



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**

---

*Autores:*

IZQUIERDO, Agustina Nahir  
37729473  
[agustina.izquierdo@mi.unc.edu.ar](mailto:agustina.izquierdo@mi.unc.edu.ar)  
SALVATIERRA, Andrés  
39611008  
[andressalvatierra@mi.unc.edu.ar](mailto:andressalvatierra@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 nuestras familias, 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 nuestros amigos y todas las personas que tuvimos el placer de conocer durante estos años de carrera.

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

Finalmente, agradecemos 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 . . . . .	21
2.4. Conurrencia y sincronización . . . . .	21
2.4.1. Conurrencia y paralelismo . . . . .	22
2.4.1.1. Problemas inherentes a la programación concurrente .	22
2.4.1.2. Exclusión mutua . . . . .	22

2.4.2.	Interbloqueo . . . . .	24
2.4.2.1.	Condiciones para producir interbloqueo: . . . . .	24
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 . . . . .	26
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 . . . . .	29
2.5.1.	Definición de S <sup>2</sup> P . . . . .	29
2.5.2.	Definición de S <sup>2</sup> PR . . . . .	30
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 v3.0 . . . . .	34
3.3.1.	Introducción . . . . .	34
3.3.2.	Objetivos . . . . .	35
3.3.3.	Desarrollo . . . . .	35
3.3.3.1.	Caso Hospital . . . . .	35
3.3.3.1.1.	Características generales . . . . .	36
	Análisis estructural . . . . .	36
	Subred derecha . . . . .	37
	Subred izquierda . . . . .	38
	Control subredes . . . . .	38
	Control red original . . . . .	40
3.3.3.2.	Caso Huang . . . . .	40
3.3.3.2.1.	Características generales . . . . .	40
	Análisis estructural . . . . .	41
	Subred izquierda . . . . .	41
	Subred derecha . . . . .	42
	Control subredes . . . . .	42
	Red controlada . . . . .	43
3.3.3.3.	Caso MFC . . . . .	43
3.3.3.3.1.	Características generales . . . . .	44
3.3.3.3.2.	Análisis estructural . . . . .	44
3.3.3.3.3.	Red controlada . . . . .	45
3.3.3.4.	Caso Portugal . . . . .	46
3.3.3.4.1.	Características generales . . . . .	46
3.3.3.4.2.	Análisis estructural . . . . .	47
	¼ de red . . . . .	47
	Control ¼ de la red . . . . .	48
	Control ½ de la red . . . . .	48
3.3.3.5.	Caso Ezpeleta v2.0 . . . . .	49
3.3.3.5.1.	Características generales . . . . .	49
	Análisis estructural . . . . .	49
	Subred izquierda . . . . .	50

Lado izquierdo del conflicto y subred derecha . . . . .	51
Lado derecho del conflicto y subred derecha . . . . .	51
Control subredes . . . . .	52
Red controlada . . . . .	53
3.3.4. Implementación del algoritmo . . . . .	54
3.3.4.1. Desarrollo . . . . .	54
3.3.5. Criterio de elección del supervisor . . . . .	56
3.3.6. Conclusión . . . . .	56
3.4. Iteración 2: Algoritmo v4.0 . . . . .	61
3.4.1. Introducción . . . . .	61
3.4.2. Objetivos . . . . .	61
3.4.3. Desarrollo . . . . .	61
3.4.3.1. Ejecución del algoritmo . . . . .	62
3.4.3.2. Pseudocódigo . . . . .	63
3.4.3.3. Diagramas de secuencia . . . . .	65
3.4.3.4. Secuencia de ejecución . . . . .	70
3.4.3.5. Ejecución en diferentes escenarios . . . . .	70
3.4.3.5.1. Caso Ezpeleta . . . . .	71
Análisis estructural . . . . .	71
Control de la red . . . . .	71
3.4.3.5.2. Caso POPN . . . . .	72
Análisis estructural . . . . .	72
Control de la red . . . . .	73
3.4.3.5.3. Caso Hospital . . . . .	74
Análisis estructural . . . . .	74
Control de la red . . . . .	75
3.4.3.5.4. Caso Huang . . . . .	76
Análisis estructural . . . . .	76
Control de la red . . . . .	77
3.4.3.5.5. Caso Ezpeleta v2.0 . . . . .	78
Análisis estructural . . . . .	79
Control de la red . . . . .	79
3.4.4. Conclusión . . . . .	81
3.4.5. Extensión: algoritmo v4.1 . . . . .	82
<b>4. Testing</b>	<b>83</b>
4.1. Nuevos escenarios . . . . .	83
4.1.1. Caso Auto y Hiuxia . . . . .	83
4.1.1.1. Características generales red Auto . . . . .	83
4.1.1.2. Análisis estructural Auto . . . . .	84
4.1.1.2.1. Control de la red . . . . .	84
4.1.1.3. Características generales red Hiuxia . . . . .	85
4.1.1.4. Análisis estructural red Hiuxia . . . . .	85
4.1.1.4.1. Control de la red . . . . .	86
4.1.2. Casos YiFan (1,2,3) . . . . .	86
4.1.2.1. Características generales YiFan1 . . . . .	87
4.1.2.2. Análisis estructural YiFan1 . . . . .	87
4.1.2.2.1. Control de la red . . . . .	87

4.1.2.3. Características generales YiFan2 . . . . .	88
4.1.2.4. Análisis estructural YiFan2 . . . . .	88
4.1.2.4.1. Control de la red . . . . .	89
4.1.2.5. Características generales YiFan3 . . . . .	89
4.1.2.6. Análisis estructural YiFan3 . . . . .	90
4.1.2.6.1. Control de la red . . . . .	90
4.1.3. Caso JianChao . . . . .	91
4.1.3.1. Características generales . . . . .	91
4.1.3.2. Análisis estructural . . . . .	92
4.1.3.2.1. Control de la red . . . . .	92
4.1.4. Caso Zhiwuli . . . . .	94
4.1.4.1. Características generales . . . . .	94
4.1.4.2. Análisis estructural . . . . .	94
4.1.4.2.1. Control de la red . . . . .	94
4.1.5. Caso Fanti . . . . .	95
4.1.5.1. Características generales . . . . .	95
4.1.5.2. Análisis estructural . . . . .	96
4.1.5.2.1. Control de la red . . . . .	96
4.1.6. Caso Hesuan Hu . . . . .	97
4.1.6.1. Características generales . . . . .	97
4.1.6.2. Análisis estructural . . . . .	98
4.1.6.2.1. Control de la red . . . . .	98
<b>5. Conclusión</b>	<b>101</b>
5.1. Trabajo a futuro . . . . .	102
<b>A. Tutorial de como utilizar el software <i>Petrinator</i></b>	<b>103</b>
A.1. Ejecución del software . . . . .	103
<b>B. Ejecución del algoritmo completa y detallada</b>	<b>109</b>
B.1. Ejecución del caso POPN . . . . .	109
<b>Bibliografía</b>	<b>119</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. . . . .	18
2.5.	Grafo de alcanzabilidad. . . . .	18
2.6.	Sifón y trampa. . . . .	18
2.7.	P-invariantes de la red de Petri. . . . .	20
2.8.	Sección crítica. . . . .	23
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. . . . .	32
3.1.	Modelado de partes del sistema Hospital . . . . .	36
3.2.	RdP Hospital y sus T-invariantes. . . . .	37
3.3.	Subred derecha Hospital contemplando el conflicto. . . . .	37
3.4.	RdP Hospital preservando lado derecho del conflicto. . . . .	38
3.5.	RdP Hospital preservando lado izquierdo del conflicto. . . . .	38
3.6.	Control RdP Hospital preservando lado izquierdo del conflicto. . . . .	39
3.7.	Control RdP Hospital preservando lado derecho del conflicto. . . . .	39
3.8.	RdP Hospital controlada. . . . .	40
3.9.	RdP Huang y sus T-invariantes. . . . .	41
3.10.	Subred izquierda Huang. . . . .	41
3.11.	Subred derecha Huang. . . . .	42
3.12.	Subred derecha Huang controlada. . . . .	42
3.13.	RdP Huang controlada. . . . .	43
3.14.	Modelado de partes del sistema MFC. . . . .	44
3.15.	RdP MFC y sus T-invariantes. . . . .	45
3.16.	RdP MFC controlada. . . . .	46
3.17.	RdP Portugal y sus T-invariantes. . . . .	47
3.18.	Porción de la RdP Portugal. . . . .	47
3.19.	Porción de la RdP Portugal controlada. . . . .	48
3.20.	Mitad de la RdP Portugal controlada. . . . .	48
3.21.	RdP Ezpeleta v2 y sus T-invariantes. . . . .	49
3.22.	Subred izquierda Ezpeleta v2. . . . .	50
3.23.	RdP Ezpeleta v2 preservando lado izquierdo del conflicto. . . . .	51

3.24. RdP Ezpeleta v2 preservando lado derecho del conflicto. . . . .	51
3.25. Control RdP Ezpeleta v2 preservando lado izquierdo del conflicto. . . . .	52
3.26. Control RdP Ezpeleta v2 preservando lado derecho del conflicto. . . . .	53
3.27. RdP Ezpeleta v2 controlada. . . . .	53
3.28. Ejecución del algoritmo v3.0. . . . .	56
3.29. Ejecución del algoritmo versión 4.0. . . . .	63
3.30. Diagrama de secuencia: procesamiento de información y tipos de ejecución. . . . .	65
3.31. Diagrama de secuencia: ejecución del primer o análisis de red con supervisores. . . . .	66
3.32. Diagrama de secuencia: función que incorpora al supervisor. . . . .	67
3.33. Diagrama de secuencia: función path conflict. . . . .	68
3.34. Diagrama de secuencia: tratamiento de conflictos y t_idle. . . . .	69
3.35. RdP Ezpeleta y sus T-invariantes. . . . .	71
3.36. RdP Ezpeleta controlada. . . . .	72
3.37. RdP POPN y sus T-invariantes. . . . .	72
3.38. RdP POPN ejecutando el análisis 1 y 2. . . . .	73
3.39. Resultado al ejecutar el análisis 3. . . . .	73
3.40. RdP POPN controlada. . . . .	74
3.41. RdP Hospital y sus T-invariantes. . . . .	74
3.42. RdP Hospital ejecutando el análisis 1 y 2. . . . .	75
3.43. Resultado al ejecutar el análisis 3. . . . .	76
3.44. RdP Hospital controlada. . . . .	76
3.45. RdP Huang y sus T-invariantes. . . . .	77
3.46. RdP Huang ejecutando el análisis 1 y 2. . . . .	77
3.47. Resultado al ejecutar el análisis 3. . . . .	78
3.48. RdP Huang controlada. . . . .	78
3.49. RdP Ezpeleta v2.0 y sus T-invariantes. . . . .	79
3.50. RdP Ezpeleta v2.0 ejecutando el análisis 1 y 2. . . . .	80
3.51. Resultado al ejecutar el análisis 3. . . . .	80
3.52. RdP Ezpeleta v2.0 controlada. . . . .	81
3.53. Incorporación del supervisor de manera automática. . . . .	82
3.54. Ejecución del análisis <i>Red con supervisores, tratamiento de conflictos y t_idle</i> de manera automática. . . . .	82
4.1. RdP Auto y sus T-invariantes. . . . .	84
4.2. RdP Auto controlada . . . . .	85
4.3. RdP Hiuxia y sus T-invariantes. . . . .	85
4.4. RdP Hiuxia controlada . . . . .	86
4.5. RdP YiFan1 y sus T-invariantes. . . . .	87
4.6. RdP YiFan1 controlada . . . . .	88
4.7. RdP YiFan2 y sus T-invariantes. . . . .	88
4.8. RdP YiFan2 controlada . . . . .	89
4.9. RdP YiFan3 y sus T-invariantes. . . . .	90
4.10. RdP YiFan3 controlada . . . . .	91
4.11. RdP JianChao y sus T-invariantes. . . . .	92
4.12. RdP JianChao controlada . . . . .	93
4.13. RdP Zhiwuli y sus T-invariantes. . . . .	94

4.14. RdP Zhiwuli controlada . . . . .	95
4.15. RdP Fanti y sus T-invariantes. . . . .	96
4.16. RdP Fanti controlada . . . . .	97
4.17. RdP Hesuan Hu y sus T-invariantes. . . . .	98
4.18. RdP Hesuan Hu controlada . . . . .	99
A.1. Software Petrinator . . . . .	103
A.2. Abrir un archivo . . . . .	104
A.3. Ventana de selección de archivo .pflow . . . . .	104
A.4. Red cargada . . . . .	104
A.5. Submenú de análisis. . . . .	105
A.6. Clasificación y propiedades de la red . . . . .	106
A.7. Exportar invariantes. . . . .	106
A.8. Exportar matrices. . . . .	107
A.9. Exportar grafo de alcanzabilidad. . . . .	107
A.10. Exportar sifones y trampas. . . . .	108
B.1. RdP POPN . . . . .	109
B.2. Primer iteración: ejecución del primer análisis de la red. . . . .	110
B.3. Lista de supervisores. . . . .	111
B.4. Elección del supervisor. . . . .	111
B.5. Reload de la red. . . . .	112
B.6. Supervisor agregado. . . . .	112
B.7. Segunda Iteración: análisis de la red con supervisores. . . . .	113
B.8. Tercera Iteración: red con supervisores, tratamiento de conflicto y t_idle. . . . .	113
B.9. Red resultante de agregar/eliminar arcos . . . . .	114
B.10. Cuarta iteración: análisis de la red con supervisores. . . . .	114
B.11. Elección del supervisor. . . . .	115
B.12. Segundo supervisor agregado. . . . .	115
B.13. Tercer supervisor agregado. . . . .	116
B.14. Cuarto supervisor agregado. . . . .	116
B.15. Última Iteración. . . . .	117
B.16. Arcos a eliminar/agregar. . . . .	117
B.17. RdP POPN controlada. . . . .	118

**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 . . . . .	7
1.5.	R3 - Algoritmo inadecuado . . . . .	7
1.6.	R4 - Herramienta inadecuada . . . . .	7
1.7.	R5 - Ausencia/escasez de datos . . . . .	7
3.1.	Supervisores: RdP Hospital (L) . . . . .	38
3.2.	Supervisores: RdP Hospital (R) . . . . .	39
3.3.	Supervisores: RdP Huang . . . . .	42
3.4.	Supervisores: RdP MFC . . . . .	45
3.5.	Supervisores: RdP Portugal . . . . .	48
3.6.	Supervisores: RdP Ezpeleta v2 (L) . . . . .	52
3.7.	Supervisores: RdP Ezpeleta v2 (R) . . . . .	52
3.8.	Análisis de los casos - Parte 1 . . . . .	57
3.9.	Análisis de los casos - Parte 2 . . . . .	58
3.10.	Análisis de los casos - Parte 3 . . . . .	59
3.11.	Supervisores: RdP Ezpeleta - Análisis 1 y 2 . . . . .	71
3.12.	Supervisores: RdP POPN - Análisis 1 y 2 . . . . .	73
3.13.	Supervisores: RdP POPN - Análisis 3 . . . . .	74
3.14.	Supervisores: RdP Hospital - Análisis 1 y 2 . . . . .	75
3.15.	Supervisores: RdP Hospital - Análisis 3 . . . . .	76
3.16.	Supervisores: RdP Huang - Análisis 1 y 2 . . . . .	77
3.17.	Supervisores: RdP Huang - Análisis 3 . . . . .	78
3.18.	Supervisores: RdP Ezpeleta v2 - Análisis 1 y 2 . . . . .	79
3.19.	Supervisores: RdP Ezpeleta v2 - Análisis 3 . . . . .	80
4.1.	Supervisores: RdP Auto . . . . .	84
4.2.	Supervisores: RdP Hiuxia . . . . .	86
4.3.	Supervisores: RdP YiFan1 . . . . .	87
4.4.	Supervisores: RdP YiFan2 . . . . .	89
4.5.	Supervisores: RdP YiFan3 . . . . .	90
4.6.	Supervisores: RdP JianChao . . . . .	93
4.7.	Supervisores: RdP Zhiwuli . . . . .	94
4.8.	Supervisores: RdP Fanti . . . . .	96
4.9.	Supervisores: RdP Hesuan Hu . . . . .	98



# 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 podemos mencionar la integración de los conocimientos adquiridos a lo largo de nuestros estudios, 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 [23]. 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 [15].

---

<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. [11] 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. [32] y Huang y et al. [9] 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 [13], 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 [13], 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 [7] 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. [16] 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. [31] 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 [28, 27] 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 [30] 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. [29] 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 nos motivó a no regirnos por una única estrategia o estudio planteado.

### 1.3. Objetivos

Tomando como base el algoritmo realizado en el proyecto integrador [10], donde se determinan supervisores para controlar los estados de deadlock alcanzables tratando de lograr la vivacidad de la red en cuestión, se seguirá investigando y testeando el mismo con nuevos tipos de redes bloqueadas que presenten diferentes características a las ya analizadas en ese documento, en busca de mejorar su alcance hacia nuevas funcionalidades.

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.

Se definen principalmente dos objetivos por mejorar del algoritmo:

- Generalizar el análisis sobre las redes sin necesidad de tener que dividirlas en caso de presentarse conflicto entre sus T-invariantes.
- Realizar una interfaz que conecta el algoritmo con el software antes mencionado, indicando mediante una retroalimentación los nuevos arcos y plazas a colocar de manera automática.

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 los objetivos principales antes mencionados.

### 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 modificar el archivo de extensión '.pflow' de la red en cuestión, para agregar plazas, arcos y quitar estos últimos en caso de ser necesario.
$R_4$	Se debe agregar una funcionalidad al Petrinator para poder recargar la red una vez modificada por el algoritmo.
$R_5$	El algoritmo debe converger en redes de Petri del tipo S <sup>3</sup> PR, sin necesidad de tener que dividirlas en caso de presentarse conflicto entre sus T-invariantes.
$R_6$	El algoritmo debe ser desarrollado para poder ser integrado al Petrinator como una nueva funcionalidad del mismo.

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****Condición:** Inexistencia del algoritmo.**Consecuencia:** No lograr alcanzar la vivacidad de una red de Petri.**Efecto:** 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 ejemplifica 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 [34]:

- *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.

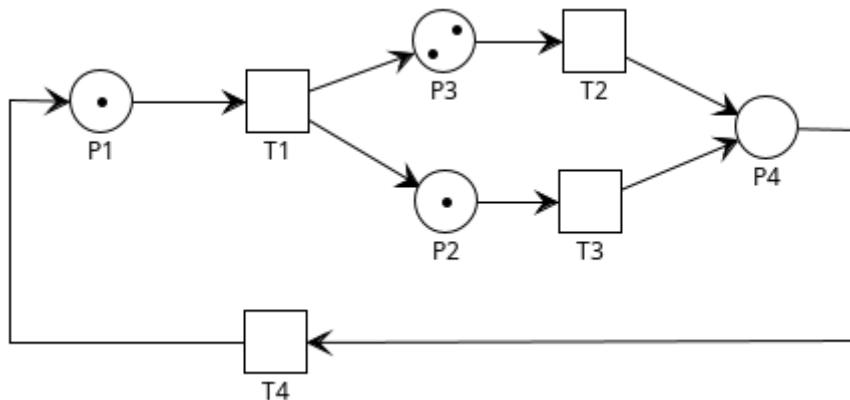


FIGURA 2.1: Red de Petri marcada.

### 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.[20]

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  $P_4 \rightarrow T_4$  ).

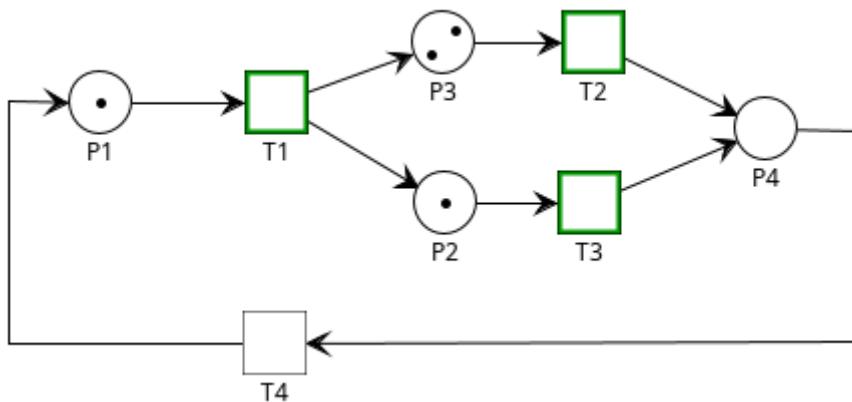


FIGURA 2.2: Transiciones sensibilizadas de la red de Petri.

### 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  $T_3$  es  $P_2$ , la cual ha **decrementado** su marcado en 1 *token* (ya que el peso del arco  $P_3 \rightarrow T_3$  es 1).
- La plaza saliente de  $T_3$  es  $P_4$ , la cual 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  $T_4$ .

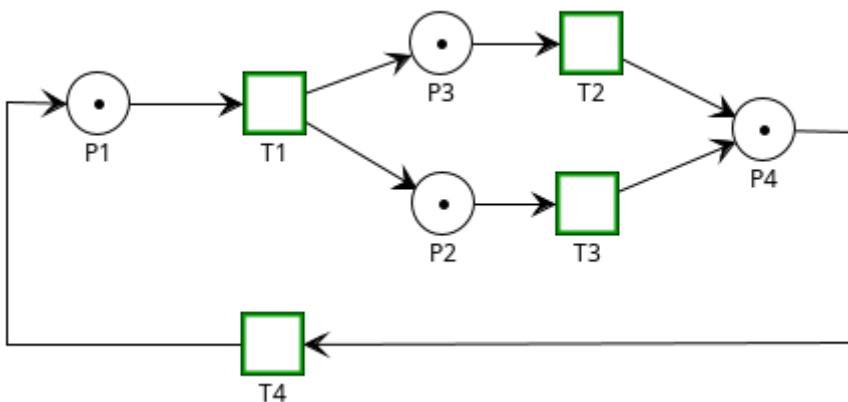


FIGURA 2.3: Nuevo marcado de la red de Petri.

## 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.<sup>1</sup> 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 deseé 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 desee 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:

---

<sup>1</sup>Cada lugar del vector corresponde a una plaza de la red, es decir, la posición 1 corresponde a la plaza P1 y así sucesivamente. Y el número de cada posición corresponde al marcado de dicha plaza.

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

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:

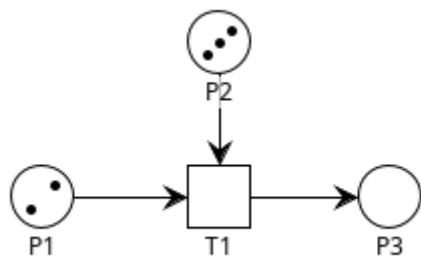


FIGURA 2.4: Red de Petri 3 estados posibles.

- 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.

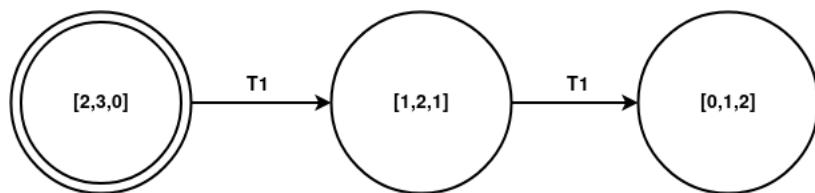


FIGURA 2.5: Grafo de alcanzabilidad.

### 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. Decimos 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.

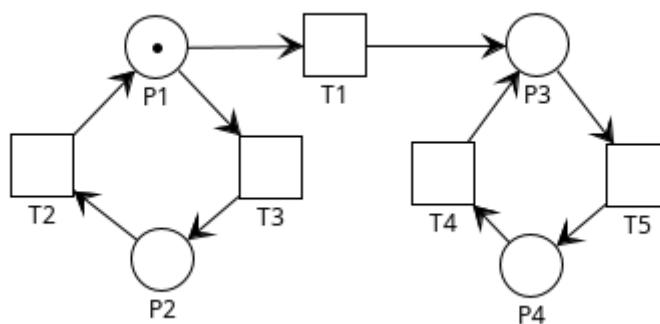


FIGURA 2.6: Sifón y trampa.

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

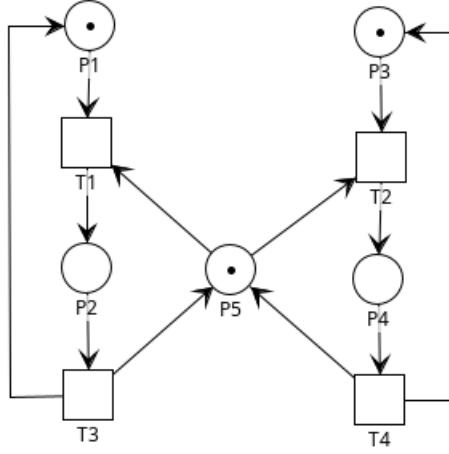


FIGURA 2.7: P-invariantes de la red de Petri.

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:

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

Si reemplazamos  $Q$  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:

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

Como los disparos son arbitrarios, podemos 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  $I \cdot x = 0$ , 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:

$$T - \text{invariantes} = \begin{pmatrix} T_0 & T_1 & T_2 & T_3 \\ 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{pmatrix} \quad (2.30)$$

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 programa 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 [22].

#### 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.**

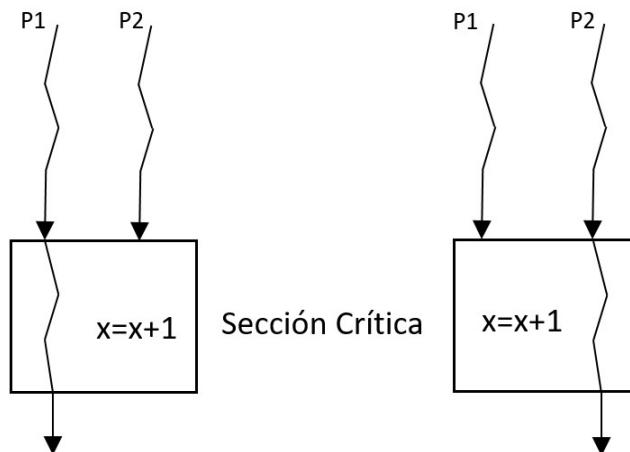


FIGURA 2.8: Sección crítica.

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 esta en la misma.

La exclusión mutua se puede representar de la siguiente manera.

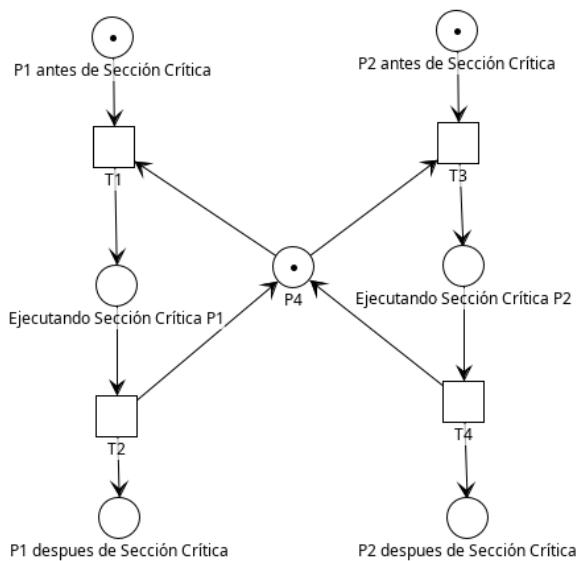


FIGURA 2.9: Sección crítica.

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  $R1$  y  $R2$ , y ambos procesos necesitan los dos recursos para proseguir, si el proceso  $A$  toma el recurso  $R1$  y el  $B$  el recurso  $R2$ , 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 [25].

#### 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*: ningún proceso podrá ser forzado a abandonar un recurso que retiene.
4. *Espera circular*: existe una cadena cerrada de procesos donde cada proceso retiene 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*: decrementa 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 ningún 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.

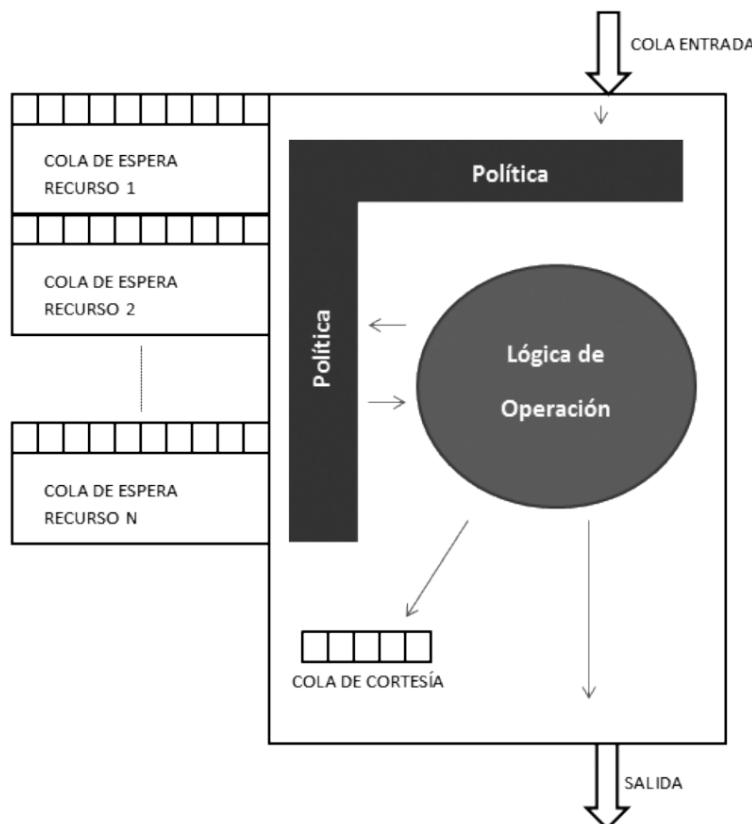


FIGURA 2.10: Estructura interna de un monitor<sup>2</sup>.

#### 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 bloquen los procesos que no tienen los recursos y que se desbloquen los que obtuvieron los recursos, y segundo, la lógica con que se administran los recursos.

<sup>2</sup>Figura adaptada del Proyecto Integrador *Desarrollo de IP cores con procesamiento de Redes de Petri Temporales para sistemas multicore en FPGA* [20, 21].

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 [20, 21].

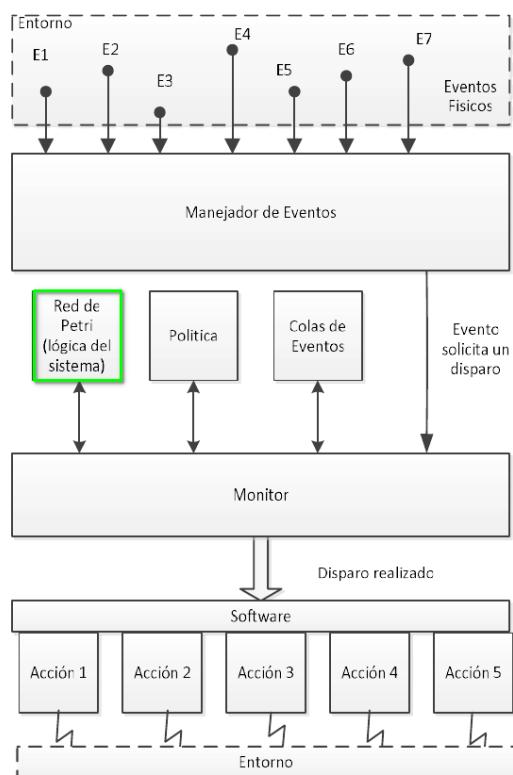


FIGURA 2.11: Monitor<sup>3</sup>.

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, mantenible 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

<sup>3</sup>Figura adaptada del paper publicado por Micolini, Orlando & Ventre, Luis & Cebollada, Marcelo & Eschoyez, Maximiliano [20]

del sistema y específicamente sobre como desbloquear las redes de Petri (del tipo  $S^3PR$ ) que la representa.

## 2.5. $S^3PR$

Definimos la clase de los procesos secuenciales simples ( $S^2P$ ); luego, lo extendemos para modelar el uso de recursos (la clase de  $S^2PR$ ) y, finalmente, definimos la clase de sistemas de procesos secuenciales simples con recursos ( $S^3PR$ ) por la composición neta de  $S^2PR$  a través de un conjunto de lugares comunes [12].

### 2.5.1. Definición de $S^2P$

Un sistema de proceso secuencial simple ( $S^2P$ ) 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
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.

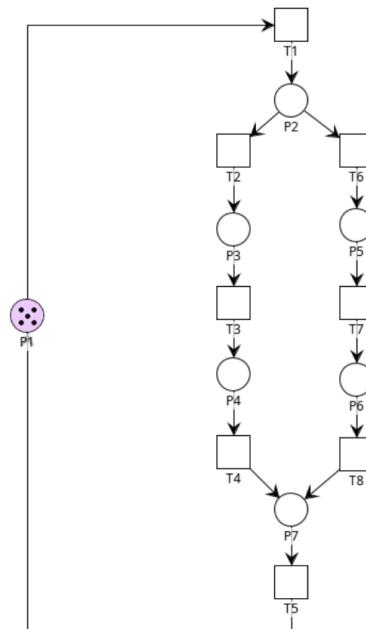


FIGURA 2.12: Red de Petri  $S^2P$ <sup>4</sup>.

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

Definimos ahora un proceso secuencial simple con recursos ( $S^2PR$ ), como un  $S^2P$  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

<sup>4</sup>Figura adaptada del libro *Deadlock Resolution in Automated Manufacturing Systems* [12].

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  $N = (P \cup \{p^0\} \cup P_R, T, F)$  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.  $\forall p \in P. \forall t \in \bullet p. \forall t' \in p \bullet \dots \bullet t \cap P_R = t' \bullet \cap P_R = \{r_p\}$
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.  $\bullet \bullet (p^0) \cap P_R = (p^0) \bullet \bullet \cap P_R = \emptyset$

$P_R$ : conjunto de plazas de recursos.

$P$ : conjunto de plazas de estado.

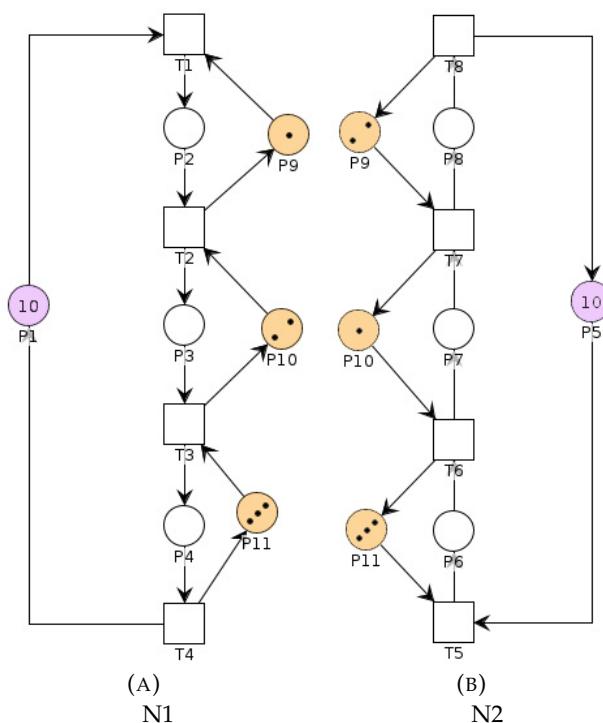


FIGURA 2.13: Redes de Petri S<sup>2</sup>PR. (A) RdP (N1,M1) - (B) RdP (N2,M2)<sup>5</sup>.

<sup>5</sup>Figura adaptada del libro *Deadlock Resolution in Automated Manufacturing Systems* [12].

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,  $r \bullet \bullet = \bigcup_{t \in r \bullet} \bullet t$  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  $\bullet \bullet p_9 = p_9 \bullet \bullet$ .

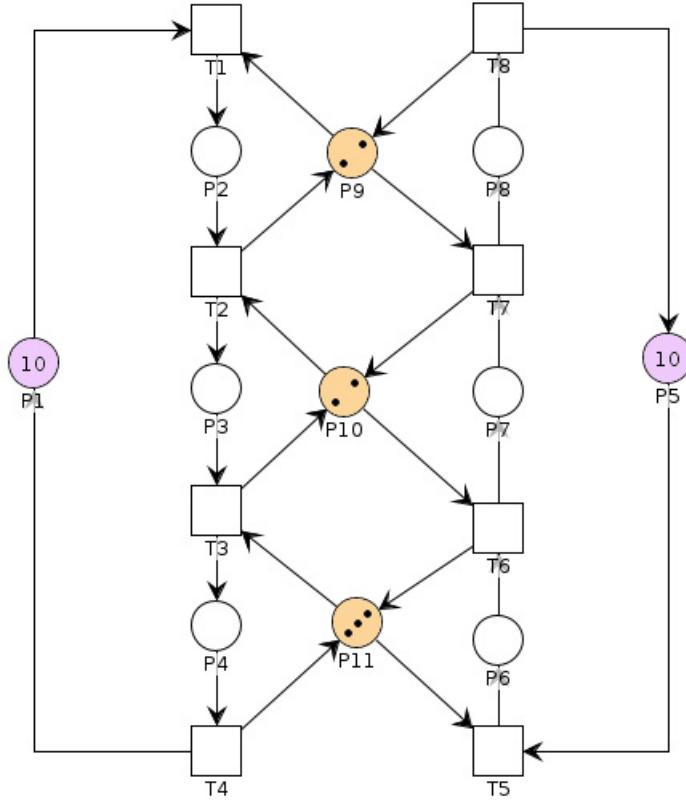
Sea  $N = (P \cup \{p^0\} \cup P_R, T, F)$  una S<sup>2</sup>PR. Un marcado inicial  $m_0$  es llamado un marcado inicial aceptable para N si:

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<sup>3</sup>PR

Entonces se puede concluir que un sistema de proceso secuencial simple con recursos  $S^3PR = \{N = \bigcup_{n=1}^k N_i = N_i = (P \cup P^0 \cup P_R, T, F)\}$  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<sup>3</sup>PR.<sup>6</sup>

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 denotando como  $(N, M)$  donde  $P = P_1 \cup P_2 = \{p_1 - p_{11}\}$ ,  $T = T_1 \cup T_2 = \{t_1 - t_8\}$ ,  $M(p_1) = M_1(p_1) = 10$ ,  $M(p_2) = M_1(p_2) = 0$ ,  $M(p_3) = M_1(p_3) = 0$ ,  $M(p_4) = M_1(p_4) = 0$ ,  $M(p_5) = M_2(p_5) = 10$ ,  $M(p_6) = M_2(p_6) = 0$ ,  $M(p_7) = M_2(p_7) = 0$ ,  $M(p_8) = M_2(p_8) = 0$ ,  $M(p_9) = \max\{M_1(p_9), M_2(p_9)\} = 2$ ,  $M(p_{10}) = \max\{M_1(p_{10}), M_2(p_{10})\} = 2$ ,  $M(p_{11}) = \max\{M_1(p_{11}), M_2(p_{11})\} = 3$ .

<sup>6</sup>Figura adaptada del libro *Deadlock Resolution in Automated Manufacturing Systems* [12].

## 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 dos iteraciones tomando como base el algoritmo antes desarrollado [10], en búsqueda de extender y mejorar su funcionamiento en redes de tipo S<sup>3</sup>PR:

- **Iteración 1:** Testeo, verificación y extensión de nuevas funcionalidades.
  1. Verificar el alcance del algoritmo sobre nuevas redes de Petri.
  2. Extensión del estudio de los T-invariantes en las redes que presentan conflicto y su influencia sobre las mismas.
- **Iteración 2:** Mantener los T-invariantes e interfaz de salida hacia el Petrinator
  1. Realizar el análisis sobre la totalidad de la red permitiendo preservar los T-invariantes de la red original.
  2. Modificar el archivo de extensión '.pflow' de la red en cuestión, para agregar plazas, arcos y quitar estos últimos en caso de ser necesario.

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

### 3.3. Iteración 1: Algoritmo v3.0

#### 3.3.1. Introducción

Se parte del algoritmo antes mencionado cuyo objetivo era la adición de las plazas de control (denominadas **supervisores**) estas 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  $S$ .

Esto se logra a partir de 3 tipos de arcos que componen las entradas y salidas del supervisor:

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.
2. Para el segundo conjunto de transiciones es necesario definir un nuevo conjunto de plazas denominadas **complemento del sifón**, estas son aquellas plazas 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. 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.

### 3.3.2. Objetivos

Verificar y validar el rendimiento del algoritmo desarrollado en nuevos escenarios, en busca de mejorar su alcance hacia nuevas funcionalidades.

### 3.3.3. Desarrollo

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.

#### 3.3.3.1. Caso Hospital

En esta red se modela el caso de un hospital. Los elementos del mismo considerados por esta serán la recepción (PW) donde se reciben los pacientes y se realiza el trabajo administrativo, la sala de consulta (CR) donde un médico da el diagnóstico a los pacientes, la sala de cirugía (S) donde se realizan las cirugías y el médico (D) que realizará todos estos procedimientos.

Además se tendrá las siguientes consideraciones:

- La persona que trabaja en la recepción y el médico podrían atender a una persona por vez.
- Tanto en la sala de consulta como en la sala de cirugía, solo un paciente a la vez podría ser tratado, es decir, podemos decir que su capacidad máxima es igual a uno.
- Si queremos que el hospital funcione en buenas condiciones, los pacientes y el personal del hospital deben respetar algunos protocolos:
  - El hospital tiene dos entradas: una normal y otra de emergencia.
    - Los pacientes que llegan a la entrada normal (IN1) tienen que ir primero a la recepción (PW) para realizar el papeleo. Dependiendo de los problemas que tengan los pacientes, pueden ser enviados a la sala

de consulta (CR) o a la sala de cirugía (S). Después de que los pacientes terminan con cualquiera de estos, van a ver al médico (Dr) para obtener el certificado de liberación. Con todas estas cosas hechas, los pacientes pueden salir del hospital (OUT1).

- El hospital tiene otra entrada (IN2): para los casos de emergencia. Los pacientes que llegaron a IN2, son revisados por el médico (D), y luego enviados a la sala de cirugía. Desde el quirófano tienen que llenar los papeles, por lo que primero deben ir a PW y luego pueden irse a casa(OUT2).

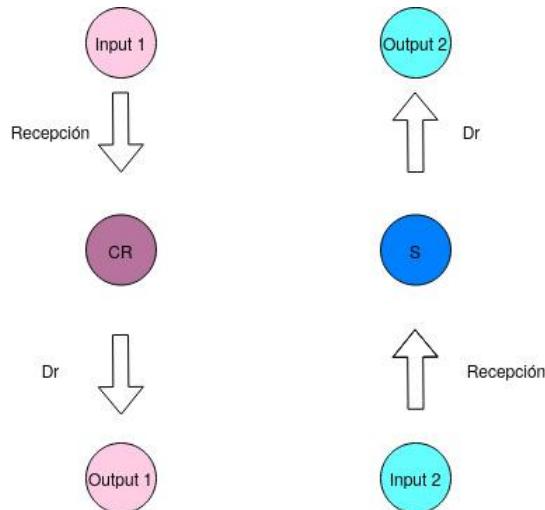


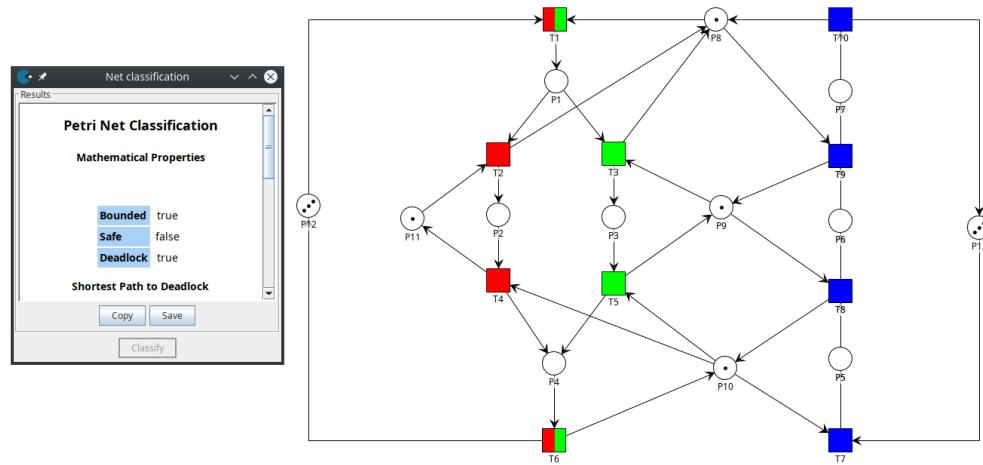
FIGURA 3.1: Modelado de partes del sistema Hospital.<sup>1</sup>

### 3.3.3.1.1. Características generales

- Presenta conflictos entre T-invariantes.
- La sala de consulta está representada por la plaza  $P_{11}$ , la recepción por la  $P_8$ , la sala de cirugía por la  $P_9$  y doctores por la  $P_{10}$ .

### Análisis estructural

<sup>1</sup>Figura adaptada del paper publicado por A. Timotei y J. Colom[26].

FIGURA 3.2: RdP Hospital<sup>2</sup> y sus T-invariantes.

En este caso particular de red en el que 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). Por esto fue necesario ejecutar el algoritmo de forma completa (5 pasos).

En un principio se buscó dividir la red preservando el conflicto en las subredes individualmente (como en los casos previos) pero las subredes resultantes no presentaban deadlock (Figura 3.3).

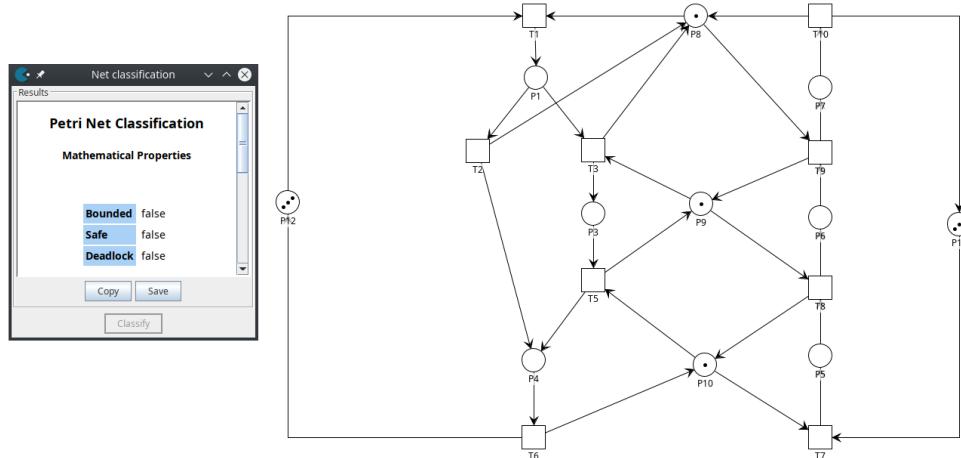


FIGURA 3.3: Subred derecha Hospital contemplando el conflicto.

Por este motivo, se optó por dividir la red contemplando en cada una de las subredes sólo uno de los caminos del conflicto y el otro T-invariante presente en la red; obteniendo dos subredes.

### Subred derecha

<sup>2</sup>Figura adaptada del paper publicado por A. Timotei y J. Colom [26].

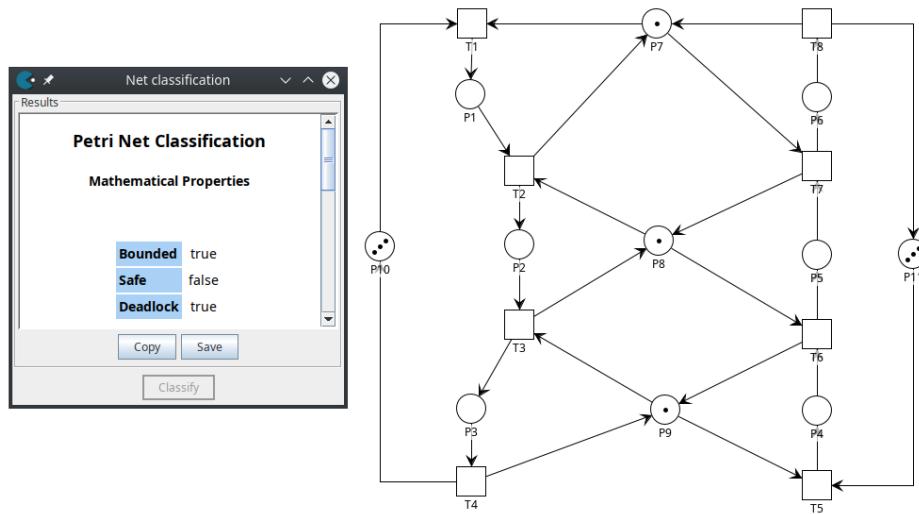


FIGURA 3.4: RdP Hospital preservando lado derecho del conflicto.

### Subred izquierda

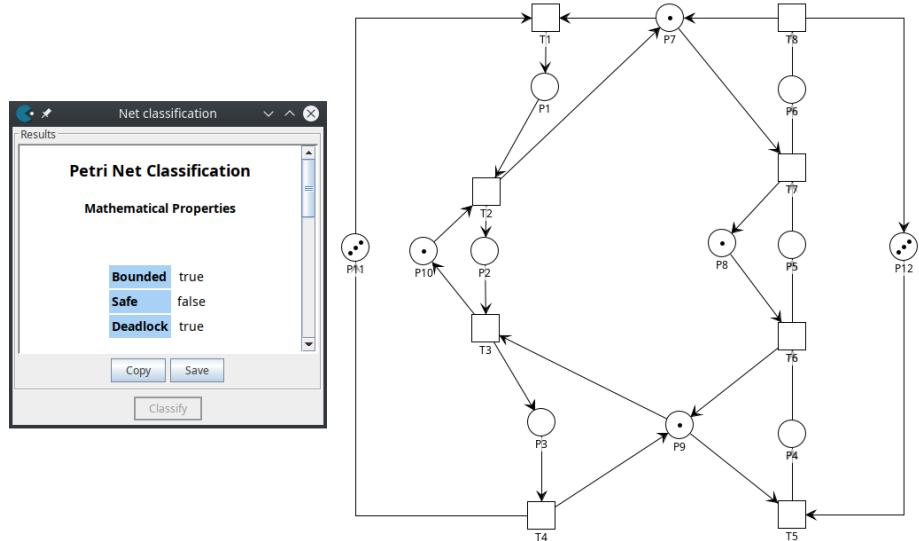


FIGURA 3.5: RdP Hospital preservando lado izquierdo del conflicto.

### Control subredes

Se ejecutaron los primeros 4 ítems del algoritmo obteniendo los supervisores correspondientes, resolviendo el deadlock en cada subred.

Supervisor	Marcado	Transiciones input	Transiciones output	Bad Siphon Controlado
$P_{15}$	3	$\{T_3, T_7\}$	$\{T_1, T_5\}$	$\{P_2, P_7, P_6, P_7, P_8\}$

CUADRO 3.1: Supervisores: RdP Hospital (L).

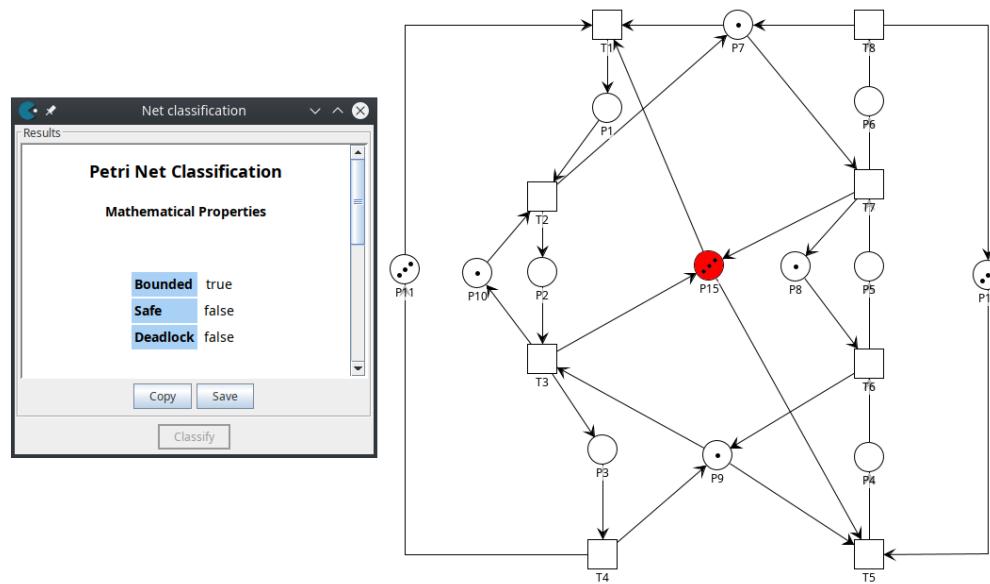


FIGURA 3.6: Control RdP Hospital preservando lado izquierdo del conflicto.

Supervisor	Marcado	Transiciones input	Transiciones output	Bad Siphon Controlado
$P_{12}$	2	$\{T_3, T_7\}$	$\{T_1, T_5\}$	$\{P_2, P_6, P_7, P_8\}$
$P_{13}$	1	$\{T_2, T_7\}$	$\{T_1, T_5\}$	$\{P_3, P_6, P_7, P_8, P_9\}$
$P_{14}$	1	$\{T_3, T_6\}$	$\{T_1, T_5\}$	$\{P_3, P_5, P_8, P_9\}$

CUADRO 3.2: Supervisores: RdP Hospital (R).

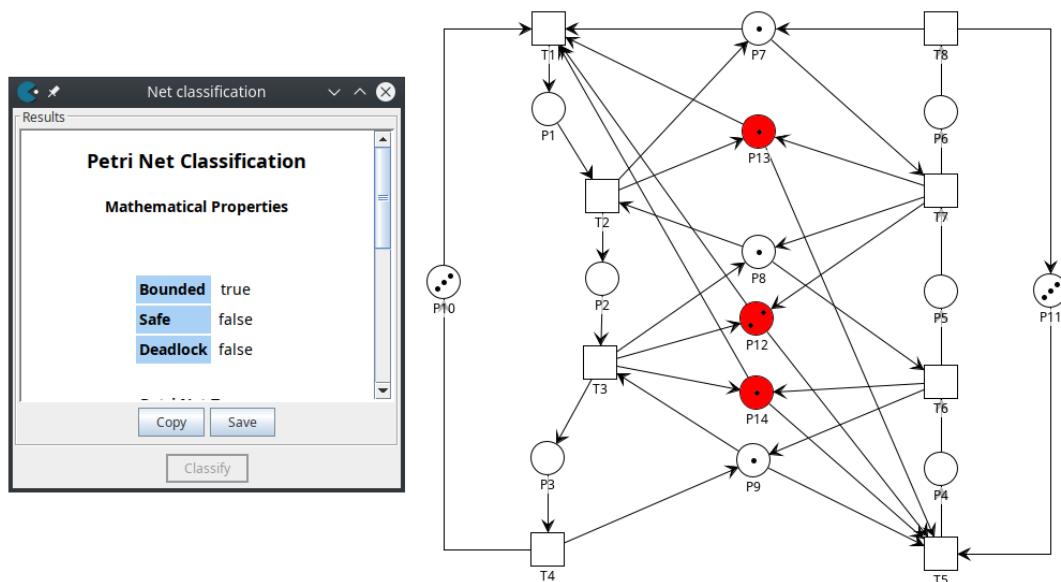


FIGURA 3.7: Control RdP Hospital preservando lado derecho del conflicto.

### Control red original

Al unir las soluciones individuales la red seguía presentando deadlock, lo que sucedió en este caso particular es que al unir ambas subredes entraron en juego nuevos supervisores que formaban parte del control de las plazas pertenecientes al otro T-invariante en conflicto. Por este motivo en caso de tomarse el camino ajeno al supervisor, la transición en conflicto debería devolver el token al mismo. Esto se puede visualizar en el caso de los supervisores representados por las plazas  $P_{14}$  y  $P_{16}$ , dado que los mismos pertenecen al control de la subred derecha, por ende no están presentes en el control de la subred izquierda; de esta manera si la red al ejecutarse toma el camino del T-invariante izquierdo, la transición  $T_2$  (que es la que da inicio a este camino) debe devolver el token consumido a estos supervisores no contemplados, conservando de esta manera la vivacidad de la red.

Además, se da el caso en el que las subredes individualmente presentan supervisores en común, por lo que el marcado del mismo (al momento de la unión) debía colocarse con el menor de ellos.

Para solucionar lo anterior se realizó lo mencionado en el punto 5.b (Sección 3.3.4.1) del algoritmo.

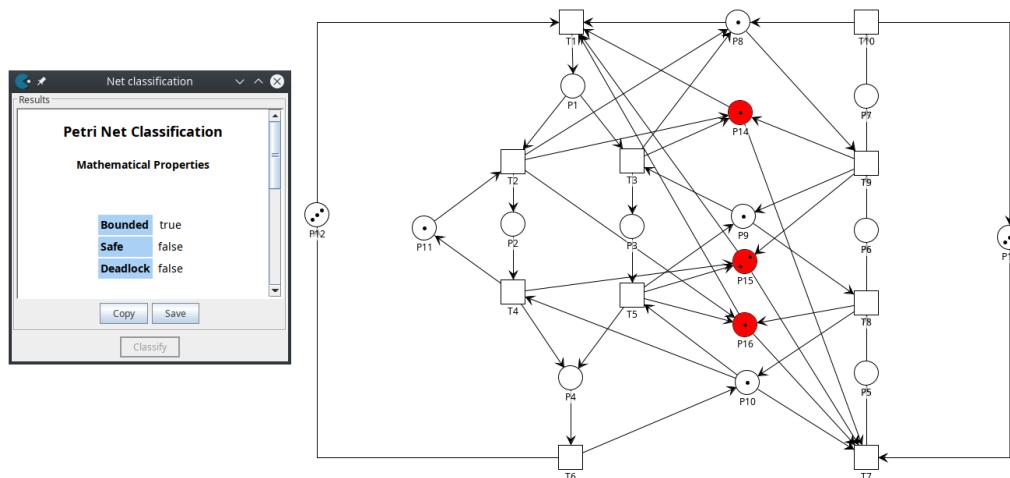


FIGURA 3.8: RdP Hospital controlada.

#### 3.3.3.2. Caso Huang

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

En la red existen plazas que simulan la disponibilidad de recursos (4) y un control incorrecto de estos en la ejecución de los procesos de trabajo, puede conducir a situaciones de deadlock.

##### 3.3.3.2.1. Características generales

- Presenta un conflicto entre T-invariantes.
- Los recursos R1-R4 están representados por las plazas  $\{P_6, P_7, P_{12}, P_{13}\}$

### Análisis estructural

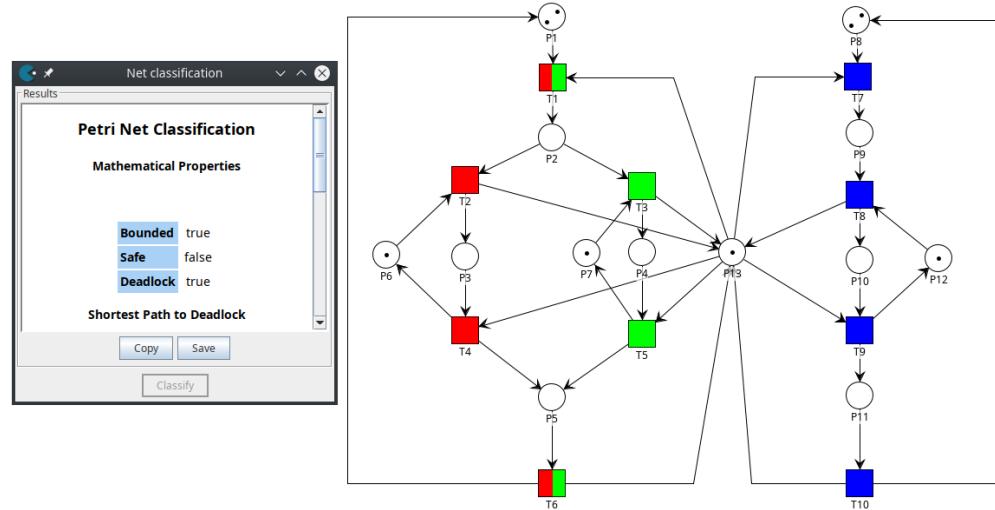


FIGURA 3.9: RdP Huang<sup>3</sup> y sus T-invariantes.

En la Figura 3.9, 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 verde, en este caso).

Dado que los primeros 4 pasos del algoritmo no lograron alcanzar una red libre de deadlock, fue necesario ejecutarlo de forma completa, es decir, incluyendo el paso 5, contemplando la división de la red. De la división resultaron dos subredes, preservando el recurso compartido ( $P_{13}$ ) en cada una de ellas.

#### Subred izquierda

Esta subred no fue necesario controlarla mediante un supervisor dado que no presenta deadlock.

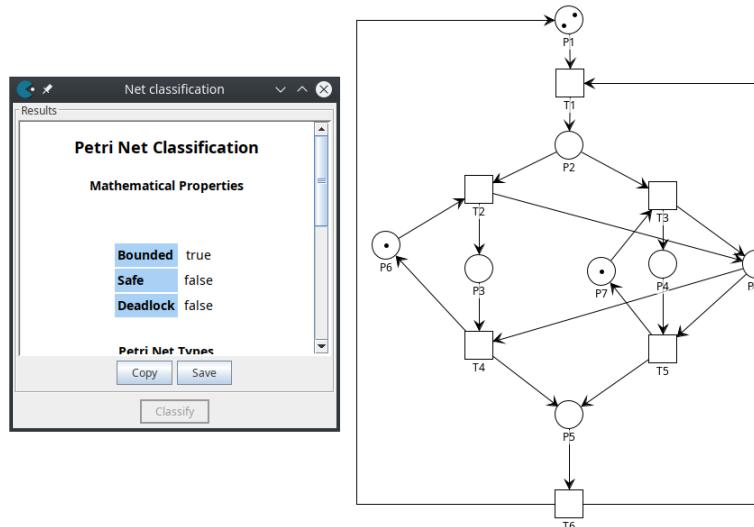


FIGURA 3.10: Subred izquierda Huang.

<sup>3</sup>Figura adaptada del paper publicado por Huang et al. [8].

### Subred derecha

Esta subred no fue necesario controlarla mediante un supervisor dado que no presenta deadlock.

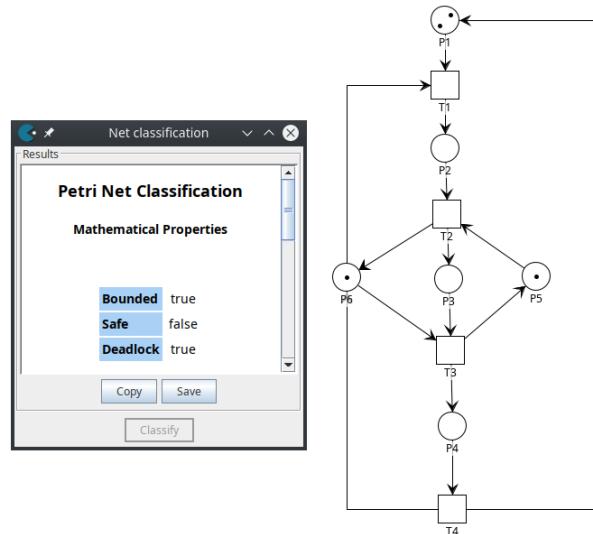


FIGURA 3.11: Subred derecha Huang.

### Control subredes

Una vez ejecutado el algoritmo (4 primeros pasos) sobre la subred derecha, se obtuvo el supervisor a agregar para su control, resolviendo así el problema de deadlock.

Supervisor	Marcado	Transiciones input	Transiciones output	Bad Siphon Controlado
$P_7$	1	$\{T_3\}$	$\{T_1\}$	$\{P_4, P_5, P_6\}$

CUADRO 3.3: Supervisores: RdP Huang.

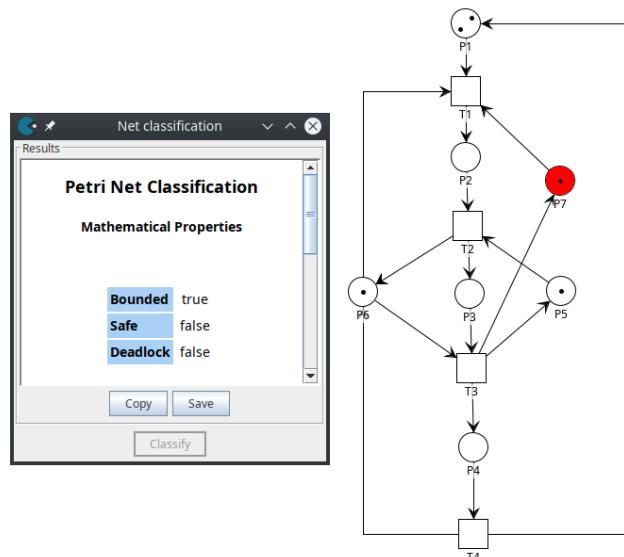


FIGURA 3.12: Subred derecha Huang controlada.

### Red controlada

Al haber contemplado en ambas subredes el recurso compartido, cuando se las integró en la red completa no fue necesario agregar otro tipo de arcos para mantener el deadlock false.

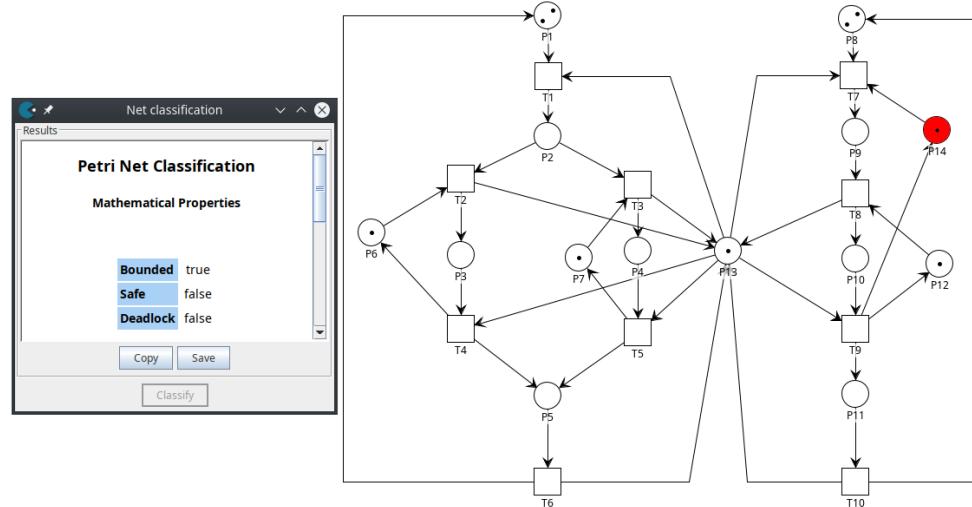


FIGURA 3.13: RdP Huang controlada.

#### 3.3.3.3. Caso MFC

En esta red se modela un FMS. El mismo consta de un conjunto de estaciones de trabajo que comparten una serie de recursos como:

- Robots R1, R2 y R3 , donde cada uno puede contener un producto a la vez.
- Máquinas M1,M2,M3 y M4 , donde cada una puede procesar dos productos a la vez.
- Accesorios de vehículos guiados automáticamente (AGV).
- Buffers tanto de carga I1 e I2 , como de descarga O1 y O2.

Con los cuales lleva a cabo la producción de dos tipos de productos: *parte<sub>1</sub>* y *parte<sub>2</sub>*.

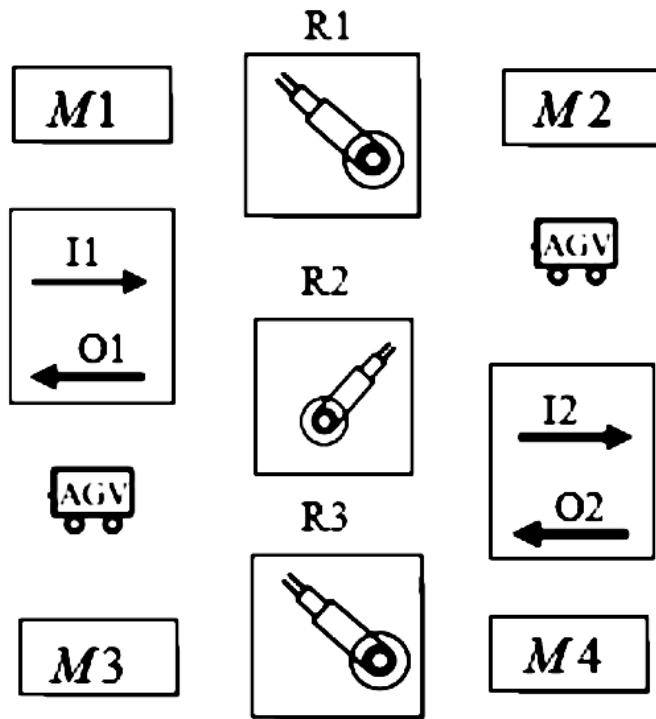


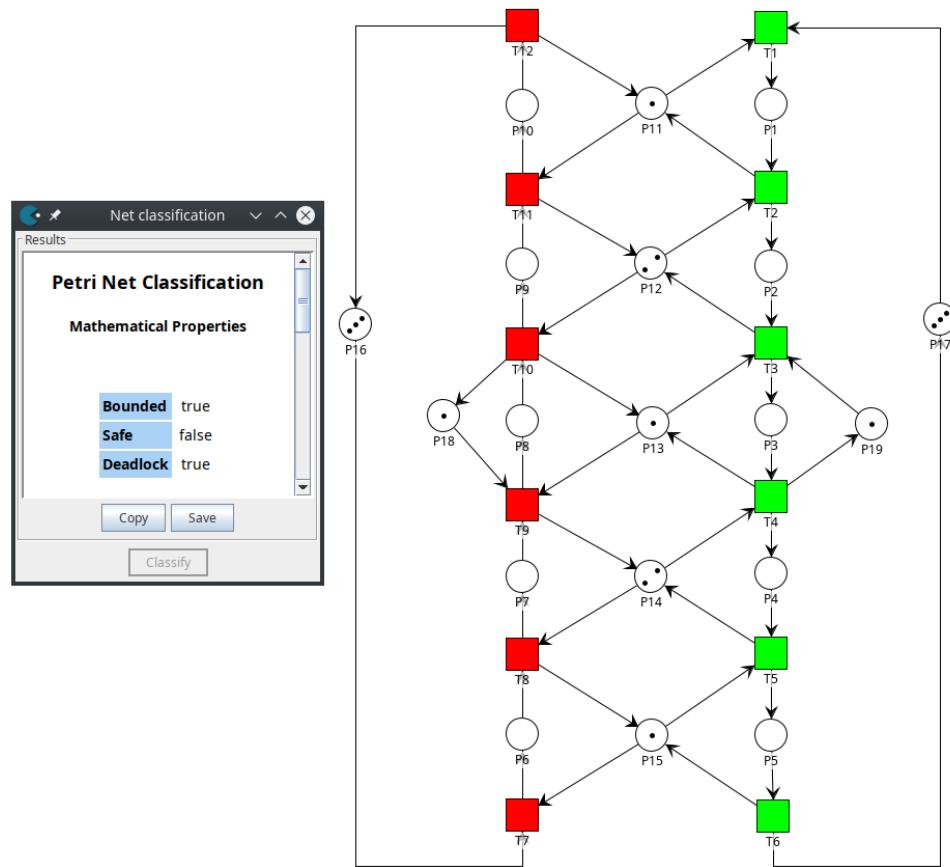
FIGURA 3.14: Modelado de partes del sistema MFC<sup>4</sup>.

### 3.3.3.3.1. Características generales

- Las máquinas M1,M2,M3,M4 están representadas por las plazas  $\{P_{12}, P_{14}, P_{18}, P_{19}\}$
- Los robots R1,R2,R3 están representados por las plazas  $\{P_{11}, P_{13}, P_{15}\}$
- La línea de producción de las *parte*<sub>1</sub> está representada por las plazas  $\{P_1 - P_5\}$ , mientras que las *parte*<sub>2</sub> por las plazas  $\{P_6 - P_{10}\}$ .

### 3.3.3.3.2. Análisis estructural

<sup>4</sup>Figura adaptada del paper publicador por *Mowafak H. Abdul-Hussin* [19] .

FIGURA 3.15: RdP MFC<sup>5</sup> y sus T-invariantes.

En la Figura 3.15, se representan los T-invariantes, en rojo y verde. Sobre esta red se ejecutaron los primeros 4 pasos del algoritmo, permitiendo encontrar el supervisor que resuelva el deadlock sin necesidad de subdividirla (paso 5); incluso realizar esta acción carecía de sentido dado que los T-invariantes por separado no representan el comportamiento de la red en su totalidad.

### 3.3.3.3.3. Red controlada

Una vez incorporado el supervisor a la red, es decir, las plazas y arcos determinados por el algoritmo, se logra su control resolviendo así el problema de deadlock.

Supervisor	Marcado	Transiciones input	Transiciones output	Bad Siphon Controlado
$P_{20}$	2	$\{T_2, T_{11}\}$	$\{T_1, T_7\}$	$\{P_2, P_{10}, P_{11}, P_{12}\}$
$P_{21}$	2	$\{T_3, T_{10}\}$	$\{T_1, T_7\}$	$\{P_3, P_9, P_{12}, P_{13}\}$
$P_{22}$	2	$\{T_4, T_9\}$	$\{T_1, T_7\}$	$\{P_4, P_8, P_{13}, P_{14}\}$
$P_{23}$	2	$\{T_5, T_8\}$	$\{T_1, T_7\}$	$\{P_2, P_{10}, P_{11}, P_{12}\}$

CUADRO 3.4: Supervisores: RdP MFC.

<sup>5</sup>Figura adaptada del paper publicador por Mowafak H. Abdul-Hussin [19].

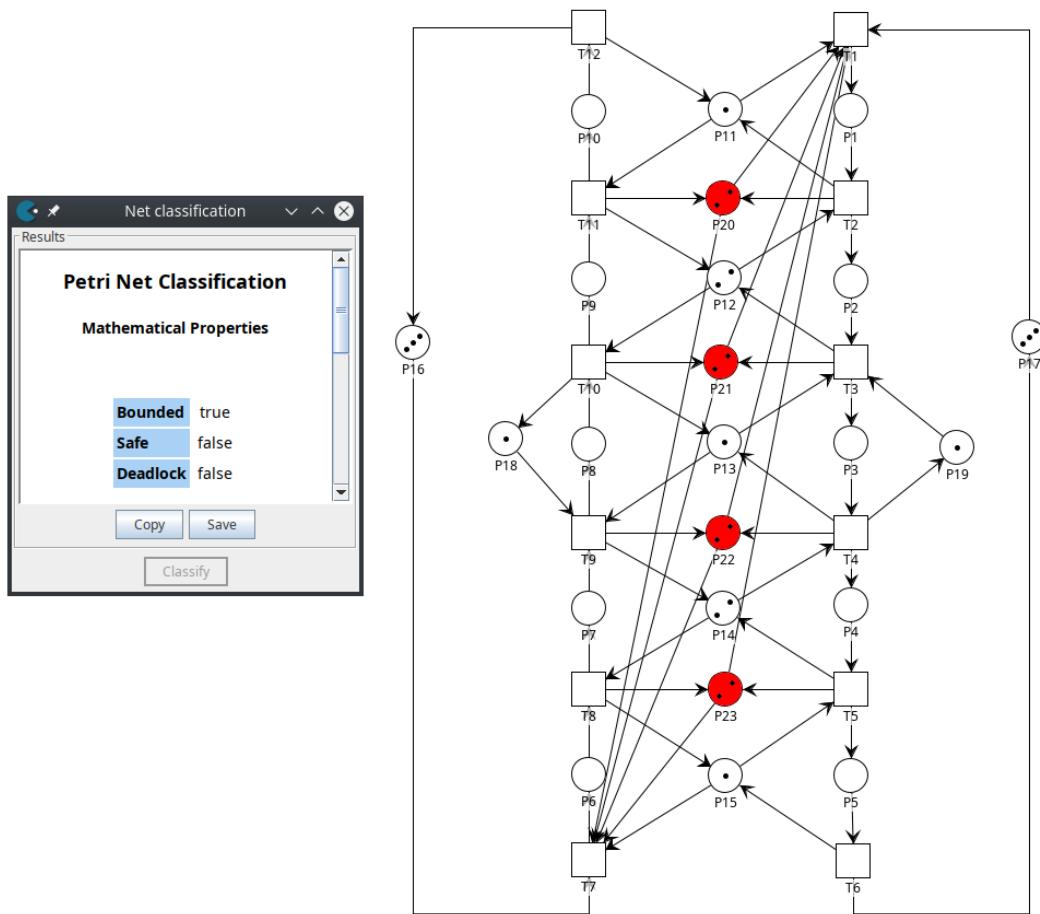


FIGURA 3.16: RdP MFC controlada.

### 3.3.3.4. Caso Portugal

Esta red modela la ejecución concurrente de procesos de trabajo en FMS, representando un sistema donde se ejecutan cuatro tipos de procesos de trabajo con estaciones de trabajo (4) idénticas, con dos recursos compartidos por todas, en caso de llevar una mala gestión de estos la red terminará en deadlock.

#### 3.3.3.4.1. Características generales

- En este caso particular la red es simétrica, pudiéndose dividir en 4 partes iguales, todas manteniendo el deadlock.
- Se trabajó sobre una de estas porciones de red, extendiendo luego la solución a las restantes partes.

### 3.3.3.4.2. Análisis estructural

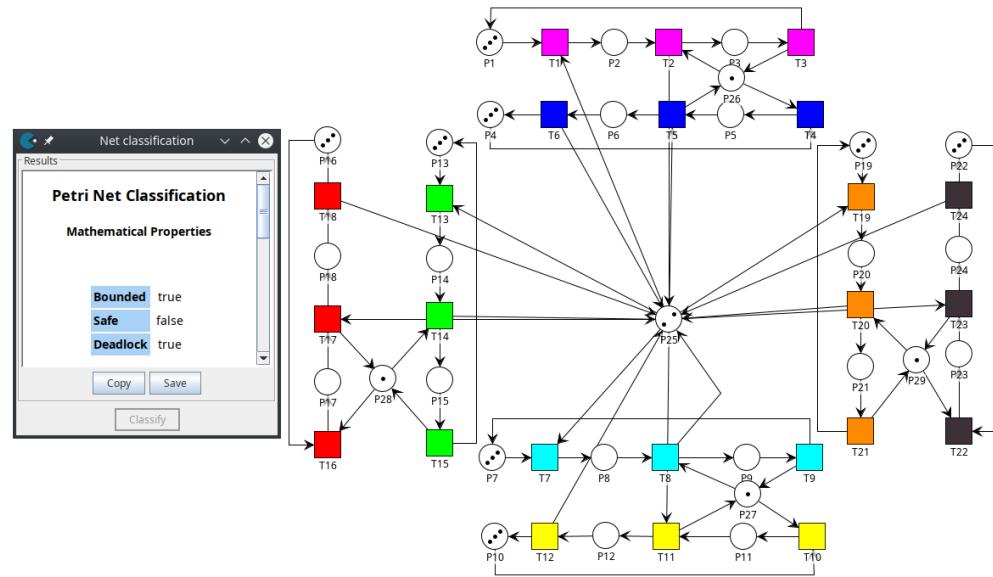


FIGURA 3.17: RdP Portugal<sup>6</sup> y sus T-invariantes.

De acuerdo con las características de la red, se trabajó sobre una de las cuatro subredes resultantes mediante el algoritmo (pasos 1 al 4), obteniendo el supervisor que controla el problema del deadlock presente en ese  $\frac{1}{4}$  de red.

Pudiéndose aplicar la misma solución a todas las restantes subredes, dada su simetría.

Al momento de unirlas con sus respectivas soluciones, hay que tener en cuenta que el recurso compartido está afectando todos los T-invariantes, por lo que se deben aplicar los arcos que componen a cada supervisor ajeno.

#### $\frac{1}{4}$ de red

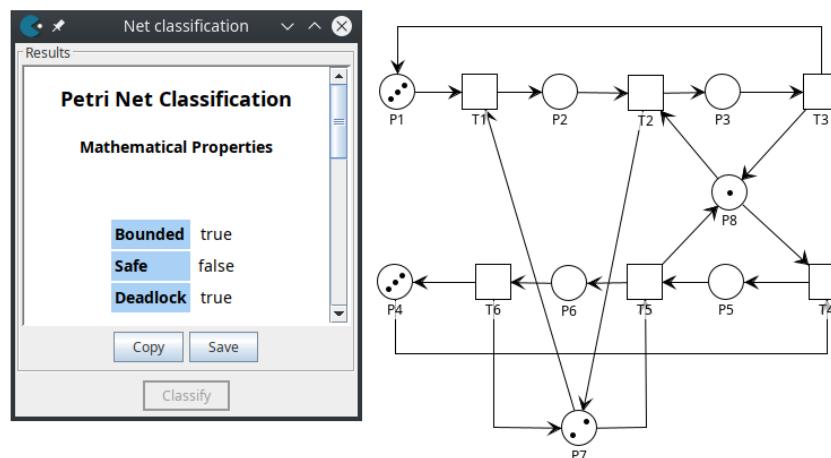


FIGURA 3.18: Porción de la RdP Portugal.

<sup>6</sup>Figura adaptada del paper publicado por S. Wang et al. [24].

### Control $\frac{1}{4}$ de la red

Una vez ejecutado el algoritmo (4 primeros ítems) sobre  $\frac{1}{4}$  de la red el problema de deadlock fue solucionado.

Supervisor	Marcado	Transiciones input	Transiciones output	Bad Siphon Controlado
$P_9$	2	$\{T_2, T_5\}$	$\{T_1, T_4\}$	$\{P_3, P_6, P_7, P_8\}$

CUADRO 3.5: Supervisores: RdP Portugal.

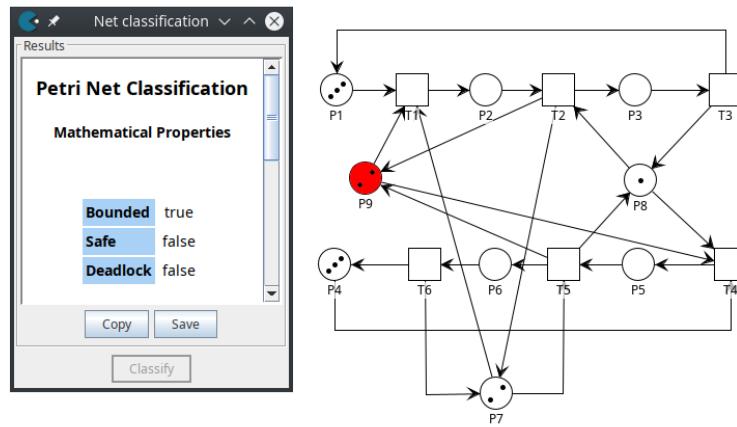


FIGURA 3.19: Porción de la RdP Portugal controlada.

### Control $\frac{1}{2}$ de la red

Realizando lo mencionado en el análisis estructural se decidió unificar las soluciones de dos cuartos de la red y como se observa en la Figura 3.20 la subred no presenta deadlock.

Lo mismo ocurriría si se unifican las 4 partes pero con el objetivo de lograr una mejor visualización del control, se optó por no desarrollarla en el presente informe.

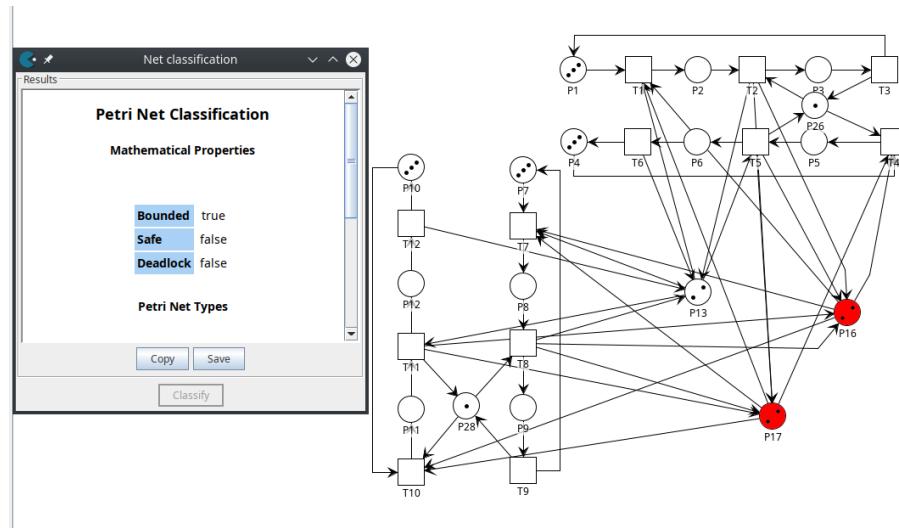


FIGURA 3.20: Mitad de la RdP Portugal controlada.

### 3.3.3.5. Caso Ezpeleta v2.0

El modelo representado en esta red es un FMS, en donde existen recursos que son compartidos por varios procesos dentro de la misma (diferentes partes de la red), los cuales se ejecutan simultáneamente compitiendo por dichos recursos, pudiendo en dicha competencia alcanzar puntos muertos indeseables, los que deberían de controlarse.

#### 3.3.3.5.1. Características generales

- Presenta un conflicto entre T-invariantes.
- Los lugares  $\{P_{15}, P_{16}, P_{17}, P_{18}, P_{19}, P_{20}\}$  denotan R1, M2, M1, R2, M3 y R3, respectivamente.
- $M_0(P_2) = 3$  y  $M_0(P_{13}) = 3$  representa el número máximo de actividades concurrentes que pueden tener lugar en cada parte de la red.

#### Análisis estructural

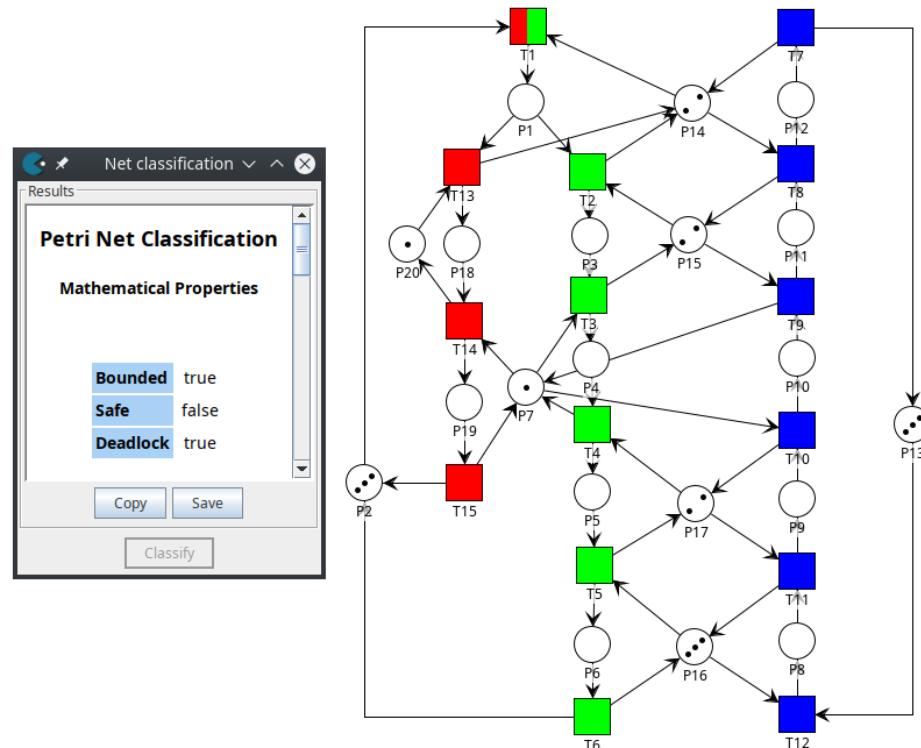


FIGURA 3.21: RdP Ezpeleta v2<sup>7</sup> y sus T-invariantes.

En este caso particular la plaza  $P_1$  de la red Figura 3.21 forma parte de un conflicto permitiendo la ejecución de un subcircuito de la red u otro, pudiendo seguir dos T-invariantes diferentes. Dado que al ejecutar los primeros 4 pasos no se alcanzo una red libre de deadlock, fue necesario ejecutar el algoritmo de forma completa (incluyendo el paso 5), es decir, dividiendo la red para lograr su control.

<sup>7</sup>Figura adaptada del paper publicado por Zhong et al. [33].

En primera instancia se dividió la red en dos subredes manteniendo por un lado los T-invariantes en conflicto (rojo y verde) y por otro lado el otro T-invariante (azul); y se observó que las mismas no presentaban deadlock Figura 3.22.

### Subred izquierda

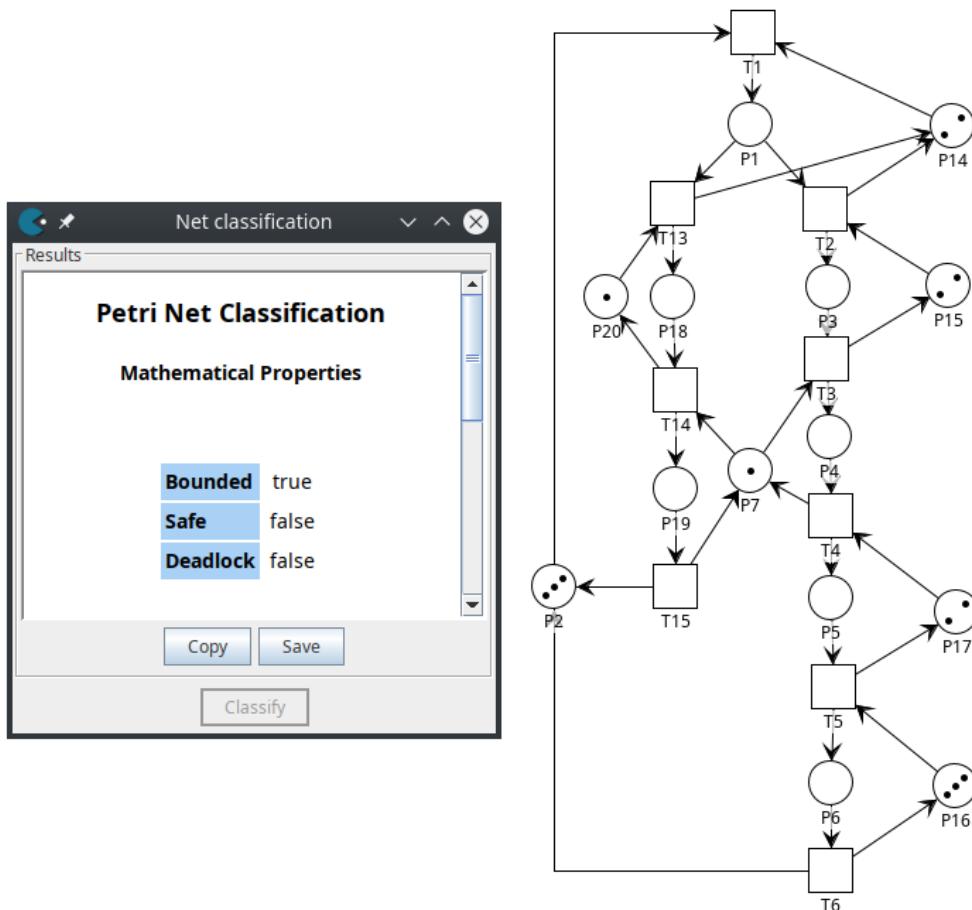


FIGURA 3.22: Subred izquierda Ezpeleta v2.

Por este motivo:

- Se dividió la red en 2 subredes, en las cuales cada una contiene sólo uno de los caminos del conflicto (por un lado rama verde y por otro rama roja) junto con el otro T-invariante (azul) presente en la red.
- Se ejecutaron los 4 ítems principales del algoritmo sobre cada una de estas, obteniendo los correspondientes supervisores y resolviendo así el problema de deadlock.

### Lado izquierdo del conflicto y subred derecha

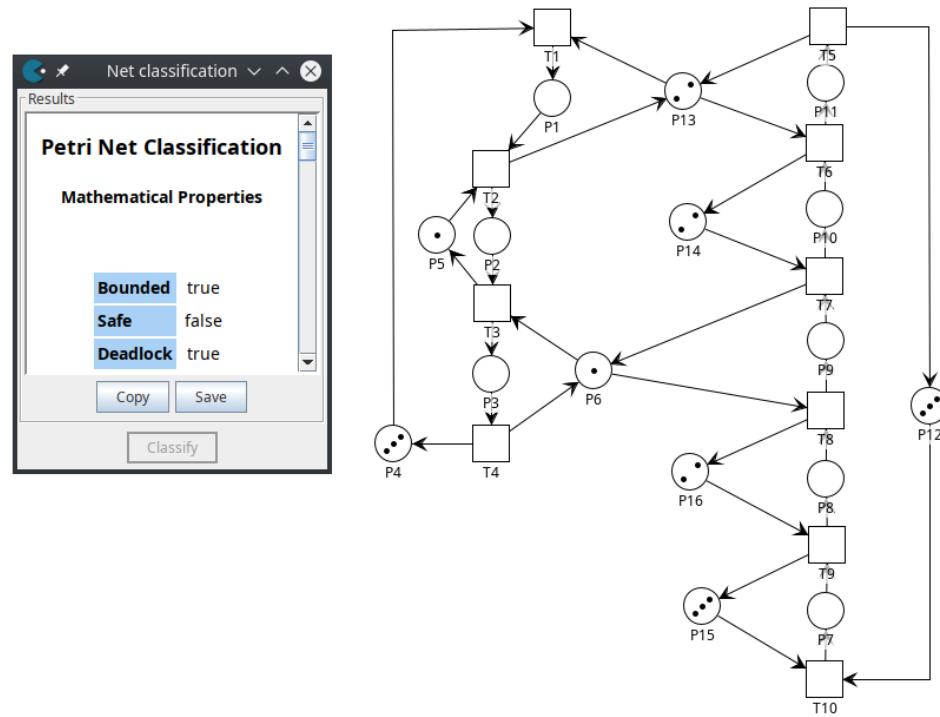


FIGURA 3.23: RdP Ezpeleta v2 preservando lado izquierdo del conflicto.

### Lado derecho del conflicto y subred derecha

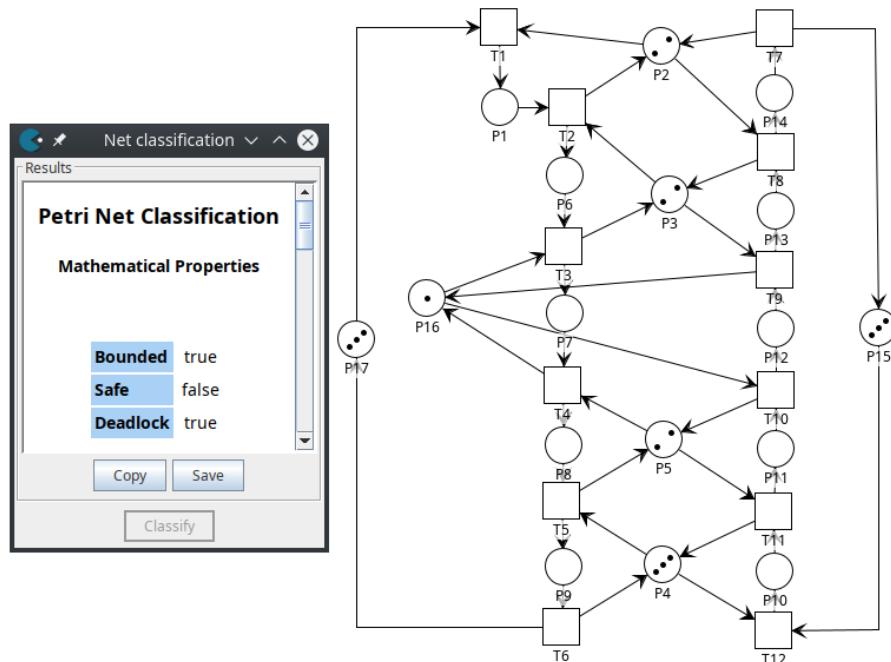


FIGURA 3.24: RdP Ezpeleta v2 preservando lado derecho del conflicto.

### Control subredes

Supervisor	Marcado	Transiciones input	Transiciones output	Bad Siphon Controlado
$P_{17}$	5	$\{T_3, T_6\}$	$\{T_1, T_{10}\}$	$\{P_3, P_5, P_6, P_{11}, P_{13}, P_{P14}\}$

CUADRO 3.6: Supervisores: RdP Ezpeleta v2 (L).

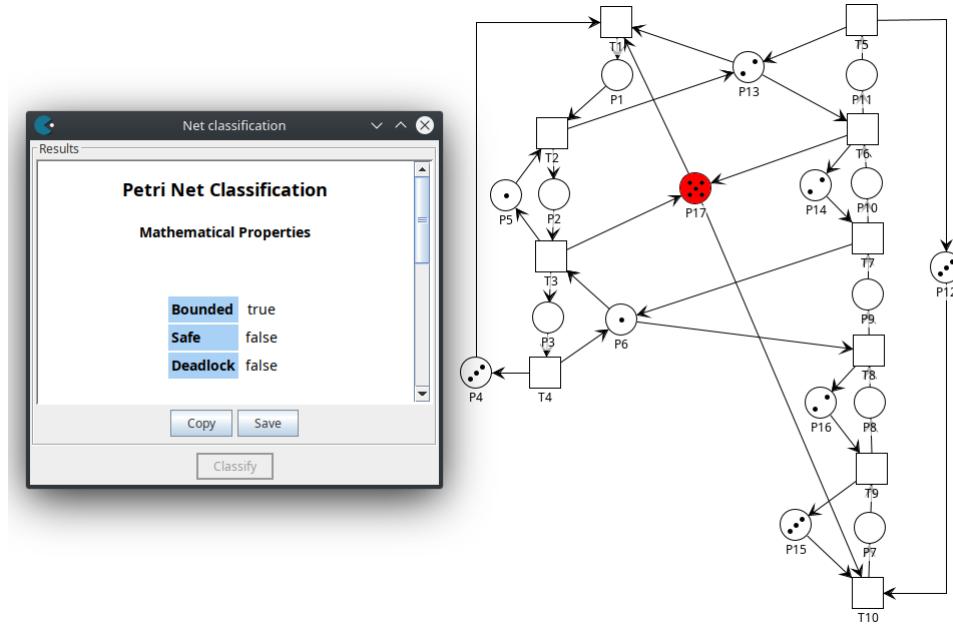


FIGURA 3.25: Control RdP Ezpeleta v2 preservando lado izquierdo del conflicto.

Supervisor	Marcado	Transiciones input	Transiciones output	Bad Siphon Controlado
$P_{18}$	4	$\{T_4, T_9\}$	$\{T_1, T_{12}\}$	$\{P_3, P_5, P_8, P_{13}, P_{16}\}$
$P_{19}$	4	$\{T_3, T_8\}$	$\{T_1, T_{12}\}$	$\{P_2, P_3, P_7, P_{14}, P_{16}\}$
$P_{20}$	2	$\{T_4, T_{10}\}$	$\{T_1, T_{12}\}$	$\{P_5, P_8, P_{12}, P_{16}\}$
$P_{21}$	2	$\{T_3, T_9\}$	$\{T_1, T_{12}\}$	$\{P_3, P_7, P_{13}, P_{16}\}$
$P_{22}$	3	$\{T_2, T_8\}$	$\{T_1, T_{12}\}$	$\{P_2, P_3, P_6, P_{14}\}$

CUADRO 3.7: Supervisores: RdP Ezpeleta v2 (R).

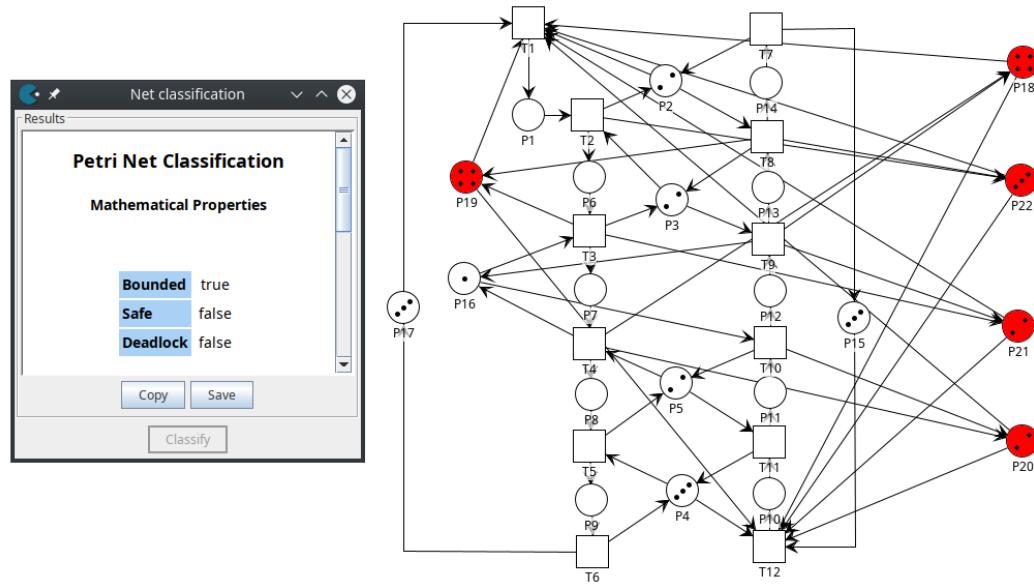


FIGURA 3.26: Control RdP Ezpeleta v2 preservando lado derecho del conflicto.

### Red controlada

Como indica el paso 5 del algoritmo al llevar a cabo la solución de la red dividiéndola como se mencionó anteriormente; al momento de unificar las soluciones parciales en la red original, se deben agregar los arcos desde las transiciones en conflicto a las plazas de los supervisores que no son propios de su subred y así devolver el token.

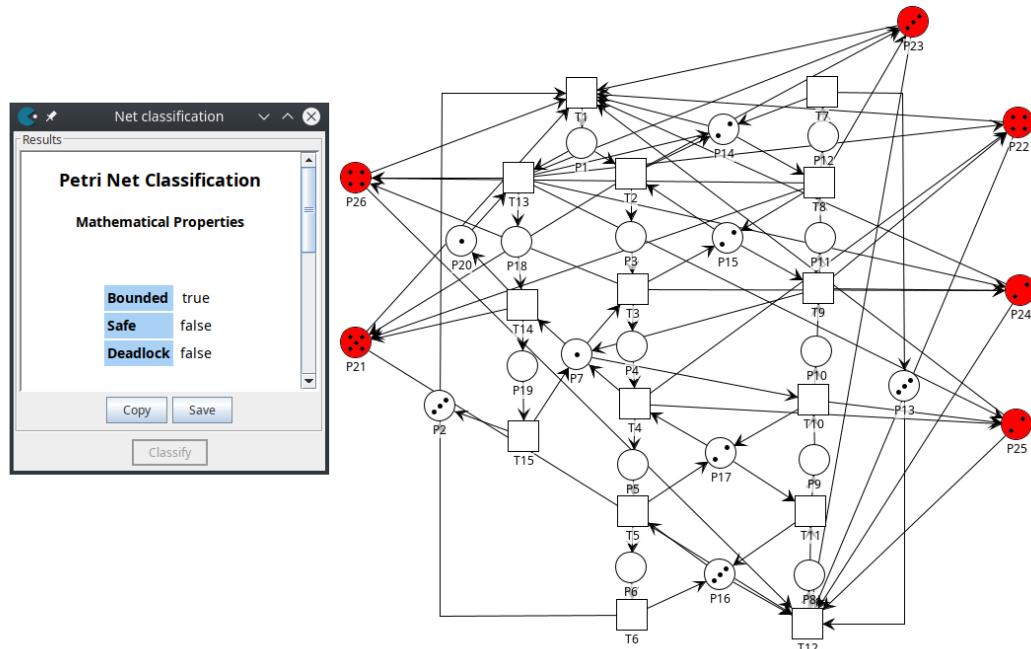


FIGURA 3.27: RdP Ezpeleta v2 controlada.

### 3.3.4. Implementación del algoritmo

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

Una de las consideraciones a tener en cuenta, la cuál emerge del estudio de las distintas redes simétricas es:

- En caso de que la red a controlar fuese simétrica, aislar en subredes y ejecutar el algoritmo en una sola de ellas obteniendo el control de la misma; el cual se podrá extender a las restantes partes. Teniendo en cuenta, al momento de integrarlas en la red completa, que los arcos pertenecientes al supervisor de cada una de las partes deben aplicarse de igual manera a los supervisores de las otras subredes (como puede observarse en el caso de la red Portugal (Sección 3.3.3.4)).

#### 3.3.4.1. 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.

4. Agregar una nueva plaza de control (perteneciente al supervisor) a la red pude 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 algunos factores:

- 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 desarrollado en algoritmo anterior [10]), las soluciones de ambas subredes se pueden unir conservando la vivacidad de la red sin problema.
- b) En caso de ser necesaria la división en subredes y la misma no pueda contemplar el conflicto en su totalidad (caso red Hospital (Sección 3.3.3.1)), notar que las transiciones pertenecientes al conflicto deben devolver el token, al momento de unir las subredes, al supervisor que no es propio de su subred; debiendo colocar de esta manera un nuevo arco en la red original.

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.  $Arcos_1 = \{(V_s, t) / t \in P_0 \bullet\}$
3.  $Arcos_2 = \{(t, V_s) / t \in C_s \bullet\}$
4.  $Arcos_3 = \{(t, V_s) / t \in conflicto \wedge t \notin T_{inv_{BS_i}}\}$

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

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 deadlock de la red desaparezca.

El algoritmo iterativo para lograr una red de Petri sin deadlock se muestra en la Figura 3.28

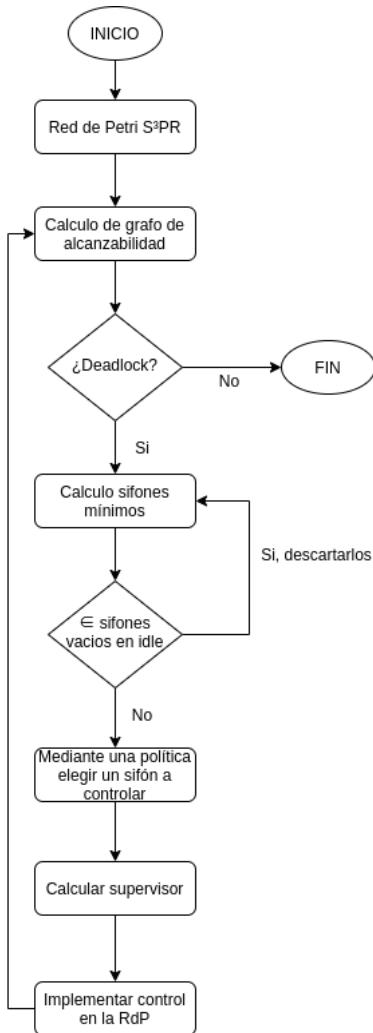


FIGURA 3.28: Ejecución del algoritmo v3.0.

### 3.3.5. 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*.

### 3.3.6. Conclusión

Con el fin de obtener una mejor visualización de la aplicación del algoritmo en los diferentes escenarios, se realizó un cuadro comparativo (incluyendo las redes desarrolladas en [10]) con las características más relevantes de cada caso.

Redes	Conflictos entre T-invariantes	Tipo de red	Símetría	¿Para la solución fue necesario dividir la red?	Solución alternativa a partir de la división del conflicto	Detalles de la solución	Recursos compartidos forman parte de un bad siphon	Recursos compartidos forman parte de un invariante	Recursos compartidos forman parte de una trampa
Panama	No	S <sup>3</sup> PR	Si	No	-	Al notar que es simétrica, se pensó en dividir la red y tratar una sola parte, pero al hacerlo se perdía el deadlock y no servía para el análisis.	Todos los recursos compartidos forman parte de algún bad siphon, solo uno se encuentra en todos los bad siphon.	Si, los tres recursos compartidos forman parte de ambos T-invariantes.	Una trampa contiene los tres recursos compartidos.
Ezpeleta	Si (pero no todos)	S <sup>3</sup> PR	No	Sí (Se dividió preservando en las subredes resultantes los T-invariantes involucrados en el conflicto con recursos compartidos).	La solución planteada fue romper el conflicto, es decir ejecutar el código en dos subredes que contenían respectivamente sólo uno de los T-invariantes en conflicto y luego, al momento de unir las soluciones, añadir un brazo que devolviera un token al otro supervisor en caso de no tomar el camino del T-invariante correspondiente.	-	No todos los recursos compartidos forman parte de algún bad siphon.	Los recursos forman parte de algún T-invariante. Solo un T-invariante incluye todos los recursos compartidos.	Ninguna trampa contiene todos los recursos compartidos. Hay una trampa que contiene 3 de los 4 recursos compartidos.
POPN	Si (pero no todos)	S <sup>3</sup> PR	No	Sí (Se dividió preservando en las subredes resultantes los T-invariantes involucrados en el conflicto con recursos compartidos).	La solución planteada fue romper el conflicto, es decir ejecutar el código en dos subredes que contenían respectivamente sólo uno de los T-invariantes en conflicto y luego, al momento de unir las soluciones, añadir un brazo que devolviera un token al otro supervisor en caso de no tomar el camino del T-invariante correspondiente.	-	No todos los recursos compartidos forman parte de algún bad siphon.	Un único recurso está compartido con todos los T-invariantes. Ningún T-invariante hace uso de todos los recursos compartidos.	Hay trampas que contienen todos los recursos compartidos.

CUADRO 3.8: Análisis de los casos - Parte 1.

Redes	Conflictos entre T-invariantes	Tipo de red	Símetría	¿Para la solución fue necesario dividir la red?	Solución alternativa a partir de la división del conflicto	Detalles de la solución	Recursos compartidos forman parte de un bad siphon	Recursos compartidos forman parte de un invariante	Recursos compartidos forman parte de una trampa
Guanjun	Si (pero no todos)	S <sup>3</sup> PR	No	No. La primera ejecución del algoritmo resolvió el problema de deadlock sin necesidad de dividirla. Se probó dividir la red teniendo en cuenta los parámetros de las anteriores (es decir a partir del conflicto y de los T-invariantes) para observar si se obtenía alguna mejora en cuanto a la cantidad de plazas supervisores y marcado. Se observó que ambas subredes no presentaban deadlock y como consecuencia de esto la división no servía.	-	-	No todos los recursos compartidos forman parte de algún bad siphon.	Los recursos forman parte de algun T-invariante. Ningun T-invariante hace uso de todos los recursos compartidos.	Ninguna trampa contiene todos los recursos compartidos.
Hospital	Si (pero no todos)	S <sup>3</sup> PR	No	Sí. Se dividió la red manteniendo el conflicto entre los T-invariantes como en el caso de las redes <sup>Ez</sup> peletaz "POPN" pero el deadlock desaparecía. La solución planteada entonces fue romper el conflicto, es decir ejecutar el código en dos subredes que contenían respectivamente sólo uno de los T-invariantes en conflicto y luego, al momento de unir las soluciones, añadir un brazo que devolviera un token al otro supervisor en caso de no tomar el camino del T-invariante correspondiente.	-	-	Todos los recursos compartidos forman parte de algún bad siphon, solo uno se encuentra en todos los bad siphon.	Los recursos forman parte de algun T-invariante. Dos T-invariantes incluyen todos los recursos compartidos.	Una trampa contiene los tres recursos compartidos.
Huang	Si (pero no todos)	S <sup>3</sup> PR	No	Sí (Se dividió preservando en las subredes resultantes los T-invariantes involucrados en el conflicto con recursos compartidos)	-	Es el primer caso en el que el supervisor de la solución no influye sobre el conflicto	El recurso compartido forma parte del bad siphon.	El recurso compartido forma parte de todos los T-invariantes.	Hay trampas que contienen el recurso compartido.

CUADRO 3.9: Análisis de los casos - Parte 2.

### 3.3. Iteración 1: Algoritmo v3.0

Redes	Conflictos entre T-invariantes	Tipo de red	Símetría	¿Para la solución fue necesario dividir la red?	Solución alternativa a partir de la división del conflicto	Detalles de la solución	Recursos compartidos forman parte de un bad siphon	Recursos compartidos forman parte de un invariante	Recursos compartidos forman parte de una trampa
MFC	No	S <sup>3</sup> PR	Si	No	-	Al notar que es simétrica, se pensó en dividir la red y tratar una sola parte, pero al hacerlo se perdía el deadlock y no servía para el análisis.	Todos los recursos compartidos forman parte de algún bad siphon.	Todos los recursos compartidos forman parte de ambos T-invariantes.	Una trampa que contiene todos los recursos compartidos.
Portugal	No	S <sup>3</sup> PR	Si	Sí. Se dividió dada la simetría tratando sólo una de las partes con el algoritmo y extendiendo la solución a las restantes. Simplicidad para el análisis.	-	Al trabajarlas por separado se tuvieron que tener en cuenta los brazos que devolvían token a los supervisores. <sup>a</sup> Jenos. <sup>a</sup> las subredes analizadas	Todos los recursos compartidos forman parte del bad siphon.	Todos los recursos compartidos forman parte de ambos T-invariantes.	Una trampa que contiene todos los recursos compartidos.
Zhao	Si (pero no todos)	S <sup>3</sup> PR	No	No	-	-	No todos los recursos compartidos forman parte del bad siphon.	Los recursos compartidos forman parte de todos los T-invariantes.	Hay trampas que contienen los recursos compartidos.
Ezpeleta v2.0	Si (pero no todos)	S <sup>3</sup> PR	No	Sí. Se dividió la red manteniendo el conflicto entre los T-invariantes como en el caso de las redes Ezpeletaz "POPN" pero el deadlock desaparecía. La solución planteada entonces fue romper el conflicto, es decir ejecutar el código en dos subredes que contenían respectivamente sólo uno de los T-invariantes en conflicto y luego, al momento de unir las soluciones, añadir un brazo que devolviera un token al otro supervisor en caso de no tomar el camino del T-invariante correspondiente.	-	-	No todos los recursos compartidos forman parte de los bad siphon.	Los recursos forman parte de algún T-invariante. Solo dos recursos compartidos afectan a todos los T-invariantes. Dos T-invariantes hacen uso de todos los recursos compartidos.	Hay trampas que contienen todos los recursos compartidos.

CUADRO 3.10: Análisis de los casos - Parte 3.

El criterio con el que se eligieron las columnas del mismo se debe a que cada red analizada estaba constituida por bad siphons, recursos compartidos y T-invariantes donde no todos se presentaban de la misma manera; y a partir de las mismas se buscaba encontrar patrones de comportamiento repetitivos o similares con el objetivo de lograr una mejora del algoritmo para obtener así uno más abarcativo.

Del análisis del mismo se pudieron realizar las siguientes observaciones:

1. Al agregar en la red las plazas y los arcos que componen al supervisor, la red modifica su comportamiento limitando su grafo de alcanzabilidad a estados deseables y evitando así que la misma evolucione a un posible estado de deadlock. Por cada supervisor incorporado en la red se generan:

- Nuevos sifones
- Nuevas trampas
- Un nuevo P-invariante

Dentro de los cuales se puede destacar la presencia de un sifón, una trampa y un P-invariante compuestos por el mismo conjunto de plazas; en los cuales siempre se incluye la plaza perteneciente al supervisor y las plazas complemento del bad siphon a controlar, entre otras plazas.

La presencia de una trampa y un sifón conformados por las mismas plazas implica que este último nunca se va a vaciar logrando así su control.

2. Al menos un recurso compartido forma parte de algún T-invariante.
3. El algoritmo fue desarrollado para analizar redes del tipo S<sup>3</sup>PR.
4. Si la red a analizar es simétrica: se busca trabajar con la red mínima (simplificando el análisis; como sucede en el caso Portugal) controlando la misma y luego extendiendo dicha solución a la totalidad de la red.  
Pueden presentarse casos en los que las redes sean simétricas y mínimas por lo que no es necesario dividirlas para llevar a cabo el análisis, como es el caso de Panamá y MFC.
5. Para el caso de las redes (Ezpeleta, POPN, Huang, Ezpeleta v2.0 y Hospital) que presentan conflicto entre T-invariantes fue necesario dividir en el análisis en diferentes subredes para lograr así controlar el deadlock. Para esta división lo que se hizo fue mantener en las diferentes subredes el conflicto y luego unir todas las soluciones. Pero en ciertas redes (Hospital y Ezpeleta v2.0) al llevar a cabo la división, antes planteada, las subredes resultantes no presentaban deadlock por lo cual el análisis no podría realizarse, para esto lo que se planteó fue dividir la red en diferentes subredes pero sin mantener el conflicto y luego a las subredes resultantes, al momento de integrarlas en una única red, se debe agregar un brazo que devuelva el token desde la subred al o a los supervisor o supervisores de la otra subred.
6. Para la división de la red, en diferentes subredes, es condición necesaria que exista conflicto entre T-invariantes pero no condición suficiente (a excepción de Huang).

7. La red Huang al igual que las que se incluyen en la generalización antes mencionada presenta conflicto, pero al dividirla y analizarla como las anteriores, los supervisores encontrados no convergen en una solución sin deadlock. Para el análisis de esta red lo que se hizo fue dividirla en dos subredes manteniendo el recurso compartido (común a todos los T-invariantes) en ambas; al realizar esto se observó que la subred que presentaba el conflicto era una subred sin deadlock por lo cual no hizo falta el agregado de supervisores. Por otro lado, la otra subred si presentaba deadlock y fue analizada para su control mediante el algoritmo. Finalmente, al unir la subred izquierda con la subred derecha ya controlada, la red resultante está libre de deadlock.

A partir de la observación 5, en la cual las redes Hospital y Ezpeleta v2.0 presentaban conflictos con el algoritmo, se decidió poner énfasis en esto y realizar una modificación del mismo (**algoritmo v3.1**) logrando que este llegue a una generalización que resuelva cada uno de los casos.

En esta extensión, se decide dividir las redes en el conflicto, pero sin mantenerlo como se hacía anteriormente. Para poder llevar a cabo esta extensión al momento de unir las redes se hizo fuerte hincapié en recorrer los T-invariantes respectivos a las transiciones en conflicto, verificando que el token extraído por alguna de estas, sea devuelto por alguna transición posterior que compone a su invariante; en caso de no cumplirse esto se debería incorporar un nuevo arco desde la transición en el conflicto hacia el supervisor y de esta manera evitar que el supervisor se vacie. Esta última solución planteada al ejecutarse en las otras redes (Ezpeleta, POPN) también resuelve el deadlock.

## 3.4. Iteración 2: Algoritmo v4.0

### 3.4.1. Introducción

A partir del análisis llevado a cabo en las secciones anteriores se realizó una última versión del algoritmo logrando generalizar su aplicación a todas las redes antes analizadas **sin la necesidad de dividir las mismas en subredes** como se presentó en diversos casos.

### 3.4.2. Objetivos

- Generalizar el algoritmo sin la necesidad de tener que dividir las redes para su análisis.
- Preservar los T-invariantes de la red original.

### 3.4.3. Desarrollo

Para alcanzar la versión final del algoritmo se analizaron las iteraciones anteriores, logrando encontrar dos puntos importantes a tener en cuenta. Uno de ellos es la generalización del algoritmo, para que pueda ser ejecutado en cada una de las redes sin necesidad de dividirlas. Y por otro lado, se observó que al agregar los arcos de los supervisores a las plazas idle, existían casos en los que dichos arcos “rompían” el T-invariante de la red original, provocando una situación de deadlock.

En busca de evitar que la red alcance este estado y lograr recuperar dichos T-invariantes, se comprendió que una de las especificaciones que planteaba Ezpeleta no se tiene que cumplir en todos los supervisores. Él plantea que las transiciones idle deben quitarle token a todos los supervisores, pero sucede que los supervisores aplican su control en un segmento acotado de la red y no en la red en su totalidad; se notó que la existencia de este brazo en las redes analizadas producía que los T-invariantes de la red original se perdieran.

Para esto, fue necesario llevar a cabo la siguiente prueba en cada uno de los t\_idle que presentaban problemas con los T-invariantes.

Se comienza con la verificación de que cada T-invariante esté realizando la devolución del token tomado del supervisor  $V_s$ , en caso contrario, probar desde cuál transición del T-invariante tendría que devolver el mismo. Esta prueba se realizó de la siguiente manera: comenzando desde la última transición que forma parte del T-invariante agregando un arco que le devuelva el token al supervisor. Si el arco que devuelve el token llega al inicio sin que se provoque deadlock, esto quiere decir, que ese T-invariante no necesita utilizar el token del supervisor  $V_s$  por lo que el arco hacia la transición idle que extrae tokens del mismo debe eliminarse.

### 3.4.3.1. Ejecución del algoritmo

El algoritmo puede ser ejecutado de diferentes maneras, dependiendo el estado del análisis de la red:

1. **Primer análisis de la red:** incluye la versión 3.1 del algoritmo y además permite extraer información relevante de la red original, como las transiciones en conflictos y los T-invariantes que servirán para el tratamiento en las próximas iteraciones. Este análisis **sólo** se realiza la primera vez.
2. **Análisis de red con supervisores:** esta ejecución se realiza de manera iterativa en busca de supervisores a incorporar hasta obtener *deadlock false*. En caso de que eso no suceda y ya no indique más supervisores por colocar; el siguiente paso es realizar la opción “3” que es un tratamiento de conflicto y t\_idle.
3. **Red con supervisores, tratamiento de conflicto y t\_idle:** el objetivo de esta ejecución es determinar que aquellas *transiciones en conflicto* le devuelvan los tokens a los supervisores en caso que el T-invariante al que pertenecen no hagan uso del mismo. Mientras que para las t\_idle en la implementación de la versión 3.1 del algoritmo todas estas le *quitan token a los supervisores*, pero no siempre al T-invariante al cual pertenecen le realizan la devolución del mismo y esto desencadena en el bloqueo de la red. Por lo tanto, el objetivo de esta ejecución es encontrar estas t\_idle y eliminar el arco que produce la extracción de esos supervisores.

```
anij@anij-HP-Notebook:~/Escritorio/banco de prueba$ python3 tesis.py
-----
Algoritmo para la solucion de deadlock para redes de petri tipo S3PR
-----
Path del archivo de Estados (html): go.html
Path del archivo de Matrices I(html): mo.html
Path del archivo de Sifones(html): so.html
Path del archivo de invariantes (html): io.html

Ingrese:
1 - Primer analisis de la red
2 - Análisis de red con supervisores
3 - Red con supervisores, tratamiento de conflicto y t_idle
```

FIGURA 3.29: Ejecución del algoritmo versión 4.0.

### 3.4.3.2. Pseudocódigo

Definiendo:

- Cantidad de sifones = cantidades de sifones total del sistema.
- SD: state deadlock.
- S: conjunto de sifones.
- BS: conjunto de bad siphons.
- VS: plaza supervisor.
- TC: transiciones en conflicto.

---

#### Pseudocódigo 1 Búsqueda de bad siphon a controlar (v4)

---

**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:          $SD_j \leftarrow \text{estado}_i$
  - 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 controle bad siphon (v4)

---

**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:     **Agregar** arco  $(VS_i, t) \forall t \in p\bullet$
  - 4:     **Agregar** arco  $(t, VS_i) \forall t \in C_S\bullet$
  - 5:     **Agregar** arco  $(t, VS_i) \forall t \in conflicto \wedge t \notin T - invariante_{BS}$
- 

---

**Pseudocódigo 3** Búsqueda de supervisor controle bad siphon (v4)

---

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

**Output:** supervisor.

- 1: **Obtener:**
    - Transiciones en conflicto.
    - Supervisores agregados.
    - T-invariantes de la red original (sin supervisores).
    - Matriz post (I+)
    - Matriz pre (I-)
  - 2: **for** i: 0 **to** len(t\_idle) **do**
  - 3:     **for** : 0 **to** Cantidad de T-invariante **do**
  - 4:         **if**  $t_{idle_i} \in T - invariante_j$  **then**
  - 5:             **for** m: 0 **to** Cantidad de supervisores **do**
  - 6:                 **if**  $\notin$  transición del  $T - invariante_j$  al cual pertenece la  $t_{idle_i}$  que devuelva token al  $VS_m$  **then**
    - 7:                     **for** i: 0 **to** len(TC) **do**
    - 8:                         **if**  $TC_k \in T - invariante_j$  **then**
    - 9:                             **Agregar** arco  $(TC_k, VS_m)$
  - 10:                 **if**  $\notin$  transición del T-invariante que  $\in$  TC **then**
  - 11:                     **if**  $\in$  arco  $(VS_m, t_{idle_i})$  **then**
  - 12:                             **Eliminar** arco  $(VS_m, t_{idle_i})$
-

### 3.4.3.3. Diagramas de secuencia

El algoritmo iterativo para alcanzar una red de Petri sin deadlock se muestra a partir de los siguientes diagramas de secuencia:

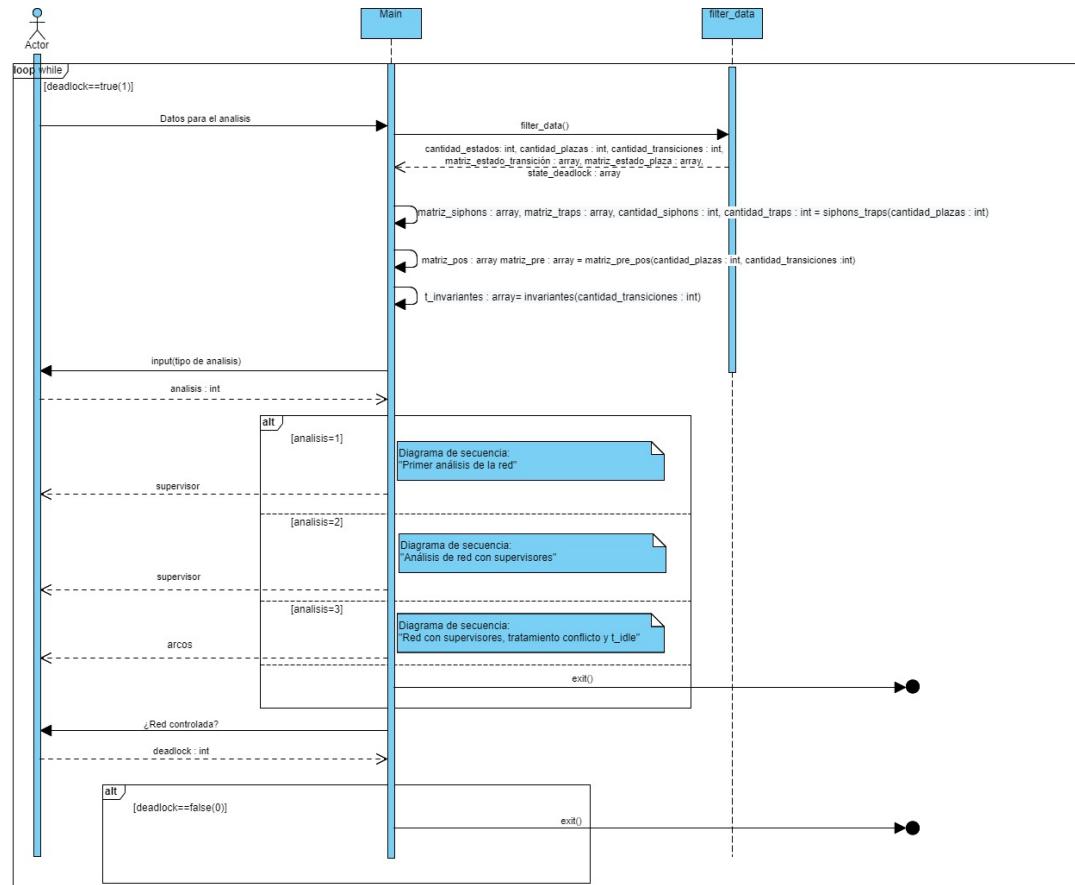


FIGURA 3.30: Diagrama de secuencia: procesamiento de información y tipos de ejecución.

En la Figura 3.30 se observa la manipulación de los datos extraídos del software utilizado para el análisis (en este caso Petrinator) desde el archivo *filter\_data.py*; junto con las opciones iniciales de ejecución del algoritmo. Cada una de las opciones luego se desarrolla individualmente para entrar más en detalle de la secuencia del mismo.

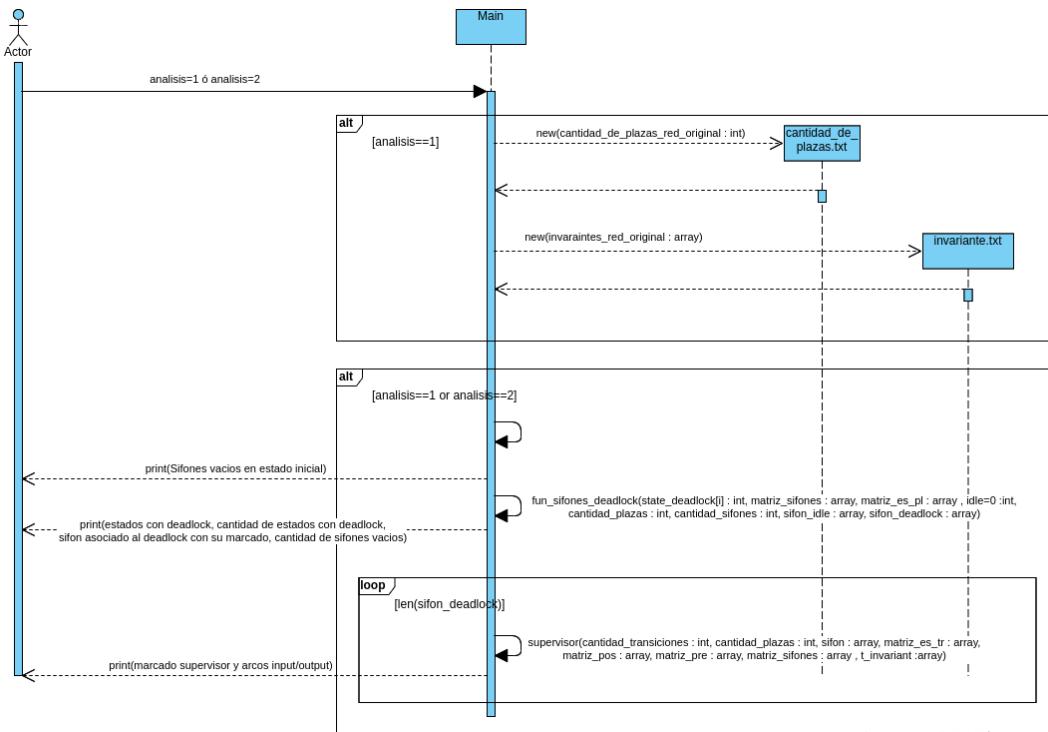


FIGURA 3.31: Diagrama de secuencia: ejecución del primer o análisis de red con supervisores.

En la Figura 3.31 se muestra la ejecución en caso de seleccionar la opción “Primer análisis de la red y Análisis de red con supervisores”, en caso de que el *análisis = 1* se exportan los archivos que serán necesarios (cantidad de plazas, T-invariantes y transiciones en conflicto de la red original, es decir, sin ningún supervisor) en caso de que la red presente conflicto entre T-invariantes y/o sea necesario el tratamiento de *t\_idle*.

La función *supervisor* que es la encargada de indicar al usuario el supervisor a colocar con el marcado y los arcos del mismo se muestra a continuación en la Figura 3.32.

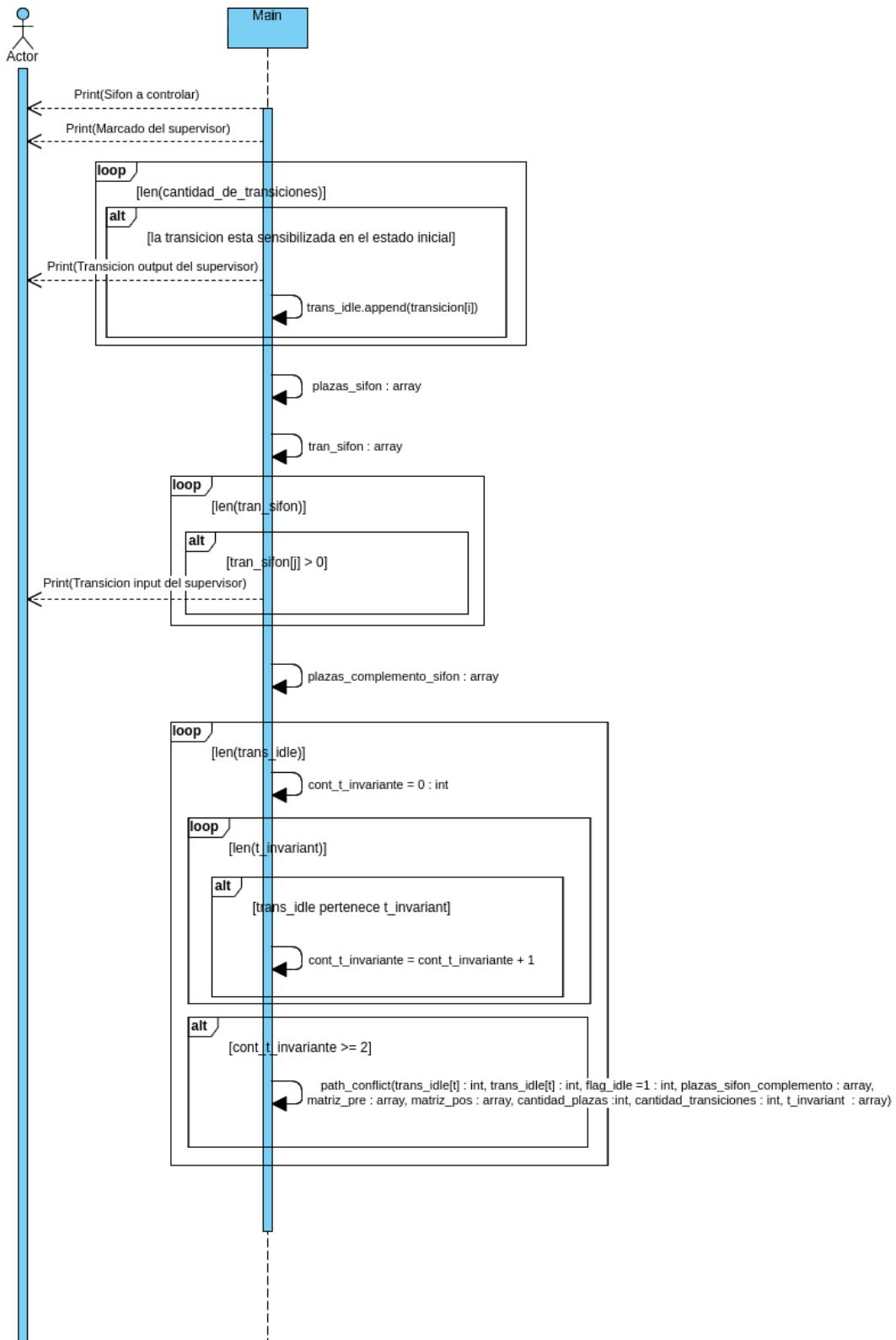


FIGURA 3.32: Diagrama de secuencia: función que incorpora al supervisor.

Para verificar si la red presenta conflicto se hace uso de la función `path_conflict`, cuya secuencia de ejecución se ve reflejada en la Figura 3.33.

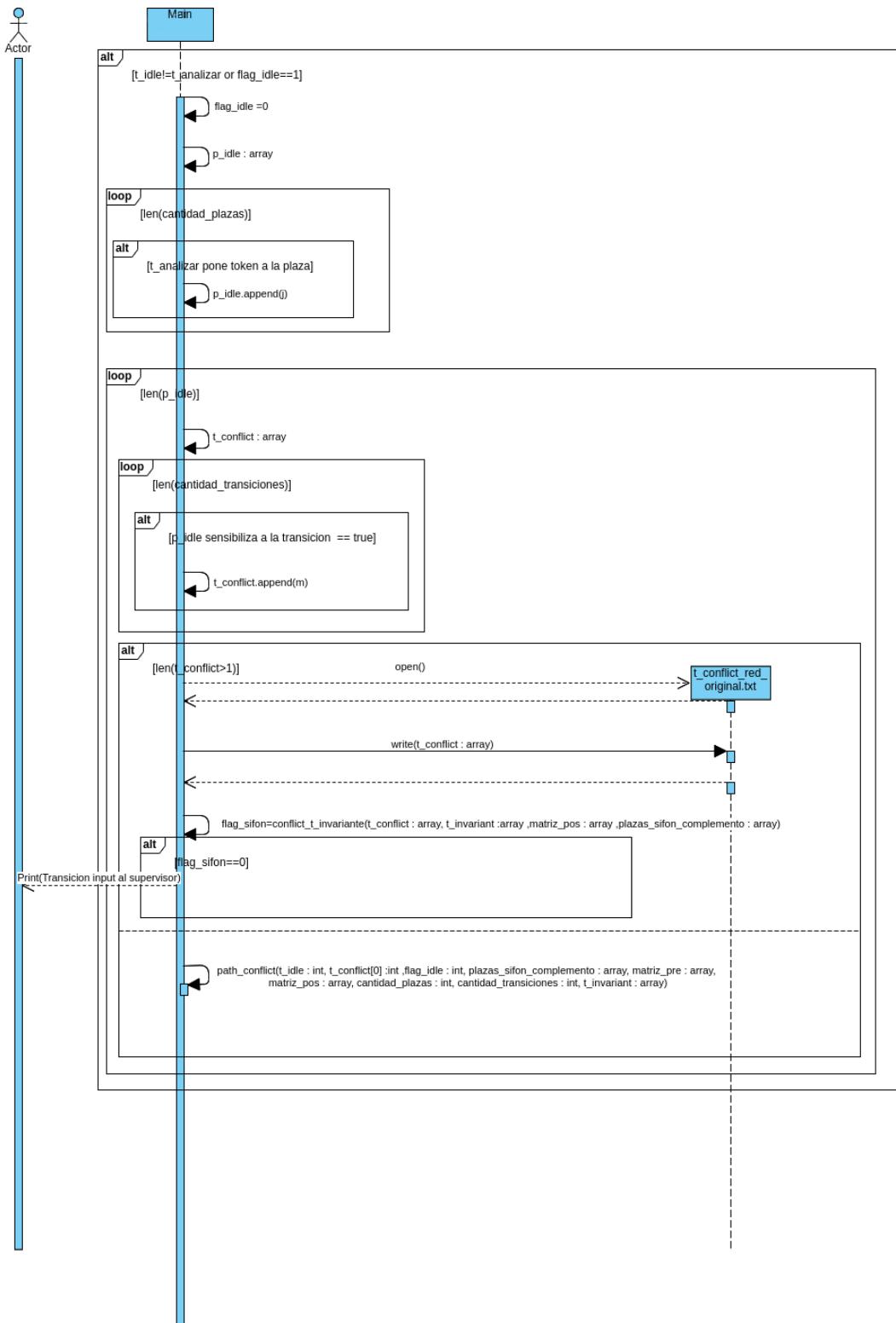


FIGURA 3.33: Diagrama de secuencia: función path conflict.

En caso de que análisis = 3, es decir, que sea necesario llevar a cabo un tratamiento de conflicto y `t_idle`, el funcionamiento del mismo se muestra a continuación.

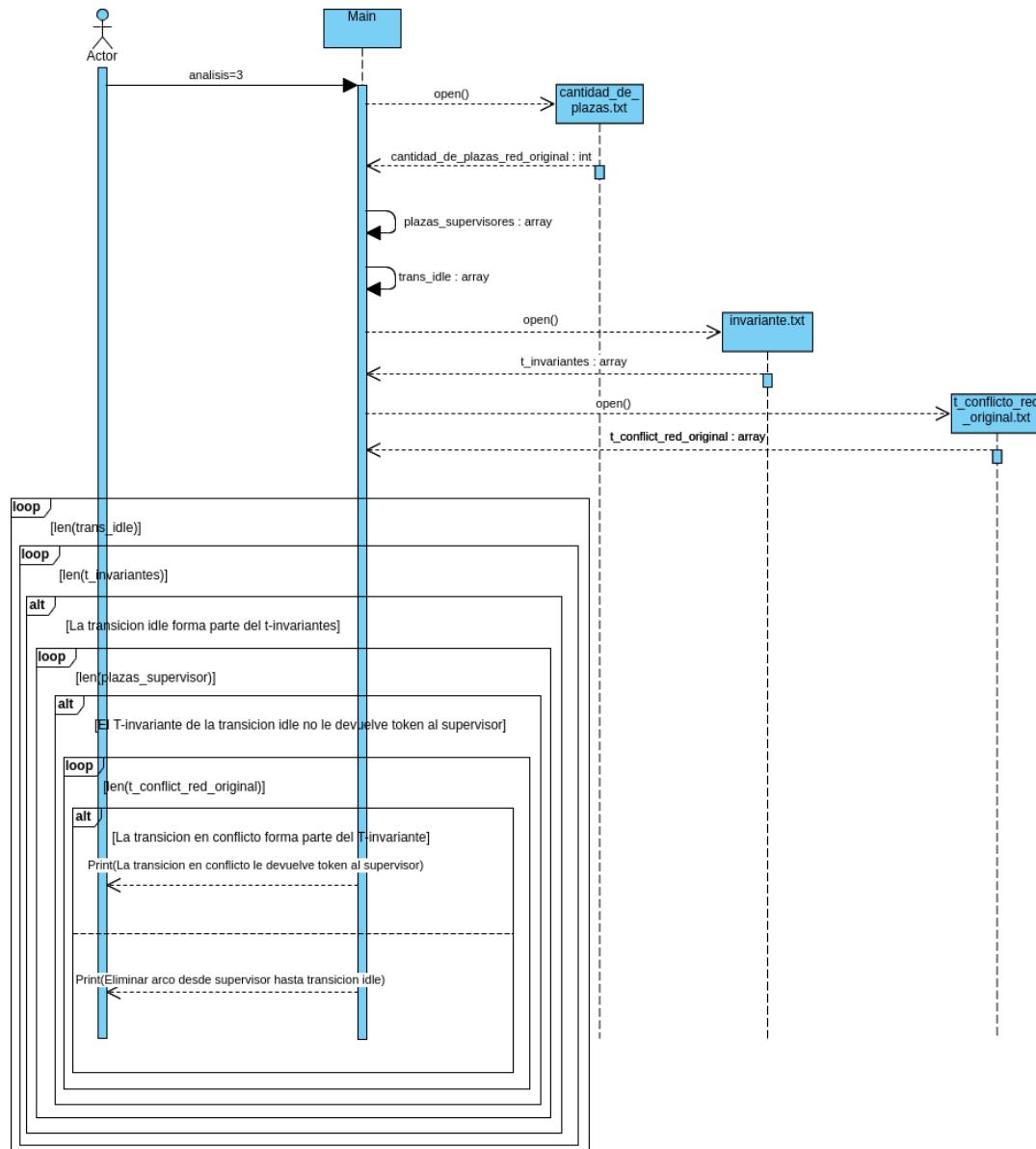


FIGURA 3.34: Diagrama de secuencia: tratamiento de conflictos y `t_idle`.

En esta sección se hace uso de los archivos exportados cuando se realizó el primer análisis de la red original; es necesario utilizar estos archivos dado que de esta manera podemos:

- Discernir cuáles son los supervisores de la red controlada, realizando una OR con las plazas de la red original.
- Mantener el comportamiento de la red original pero acotando su alcanzabilidad a estados sin deadlock, para esto es necesario conocer los T-invariantes presentes en la red sin controlar dado que al agregar los supervisores estos invariantes sufren modificaciones y hasta en ocasiones desaparecen de la misma.

#### **3.4.3.4. Secuencia de ejecución**

En primera instancia se extraen los archivos necesarios del software (Petrinator) y una vez que se obtuvieron se procede de la siguiente manera:

Llevar a cabo el “Primer análisis de la red” y luego colocar uno de los supervisores que sugiere el algoritmo.

En caso de que la red aún presente deadlock proseguir con la ejecución bajo la elección de “Análisis de red con supervisores” y continuar colocando algún supervisor sugerido hasta que la red esté controlada.

En caso de que el algoritmo continúe indicando algún supervisor:

1. Con marcado 0 (cero) ó
2. Con solo arcos de salida y sin arcos de entrada ó
3. Algún supervisor que ya existe.

Se lleva a cabo el análisis “Red con supervisores, tratamiento de conflictos y t\_idle”, el cual indica que arcos agregar/quitar para lograr alcanzar el control de la red.

Si la red no se controla puede ser que la elección de los supervisores no haya sido la correcta y debería volver a ejecutar el “Primer análisis de la red” con la red original, eligiendo otro supervisor.

#### **3.4.3.5. Ejecución en diferentes escenarios**

Se llevó a cabo un nuevo análisis en cada una de las redes mediante esta nueva revisión del algoritmo, verificando su funcionamiento y en busca de obtener mejores resultados a través de la generalización del mismo.

Este nuevo algoritmo se ejecuta sobre la totalidad de la red, permitiendo así obtener los supervisores necesarios para resolver el deadlock de la misma.

Si bien el análisis se realizó sobre todas las redes mencionadas anteriormente, en esta sección sólo se visualizan aquellas que en la versión previa fueron necesario dividirlas para analizarlas.

### 3.4.3.5.1. Caso Ezpeleta

#### Análisis estructural

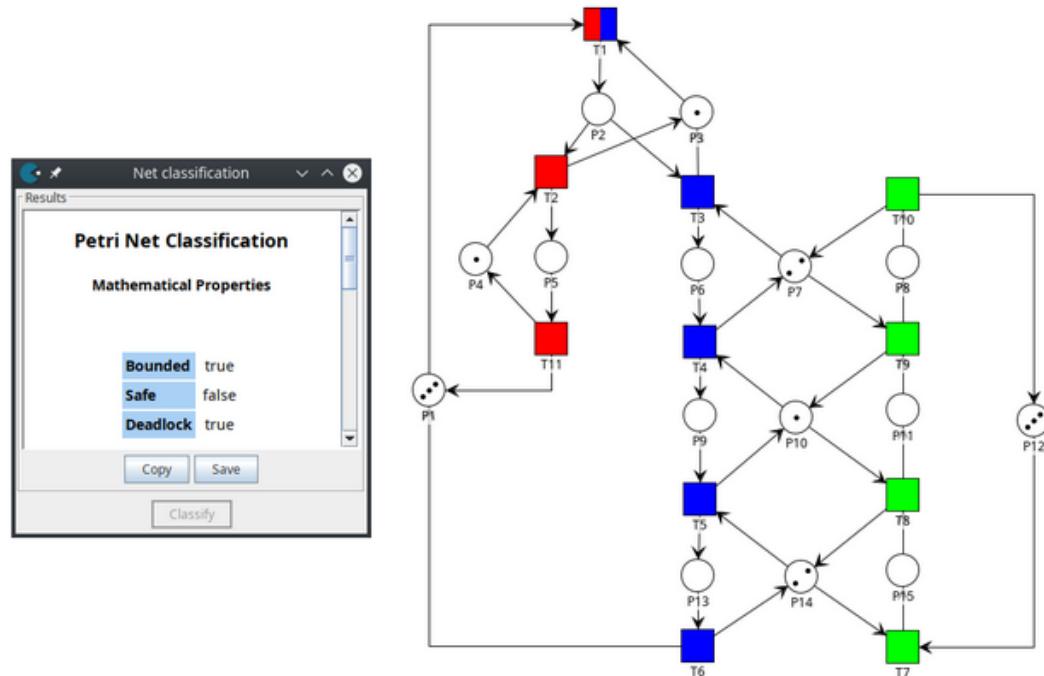


FIGURA 3.35: RdP Ezpeleta<sup>8</sup> y sus T-invariantes.

En la Figura 3.35, se observan los T-invariantes y la plaza P2 que forma parte de un conflicto.

#### Control de la red

Supervisor	Marcado	Transiciones input	Transiciones output	Bad Siphon Controlado
$P_{17}$	2	$\{T_2, T_4, T_9\}$	$\{T_1, T_7\}$	$\{P_7, P_8, P_{10}, P_{19}\}$
$P_{16}$	2	$\{T_2, T_5, T_8\}$	$\{T_1, T_7\}$	$\{P_{10}, P_{11}, P_{13}, P_{14}\}$

CUADRO 3.11: Supervisores: RdP Ezpeleta - Análisis 1 y 2.

<sup>8</sup>Figura adaptada del paper publicador por Ezpeleta et al. [11].

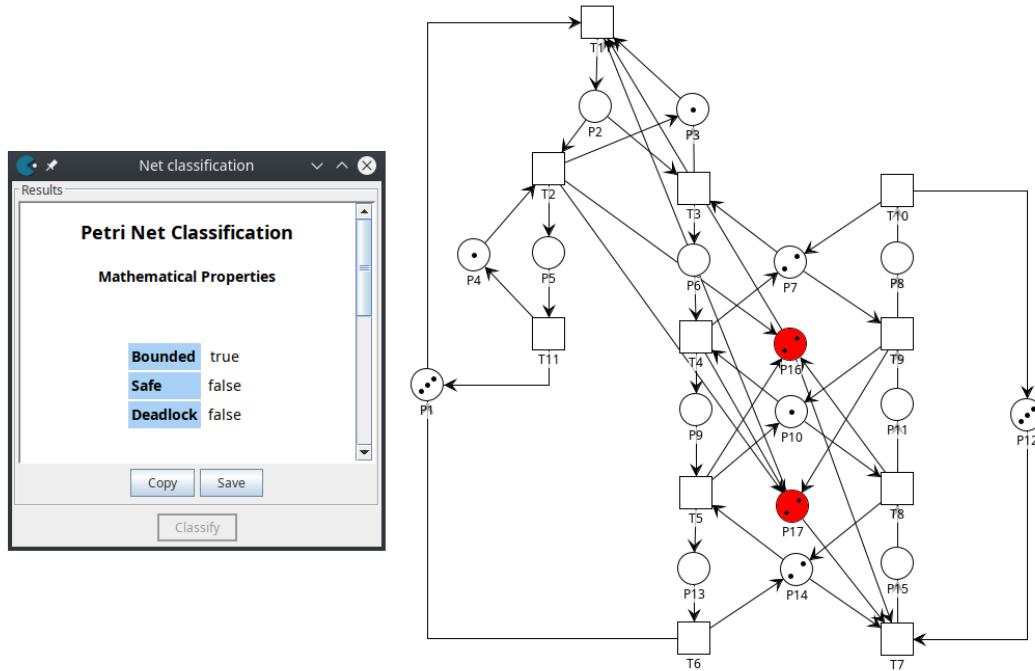
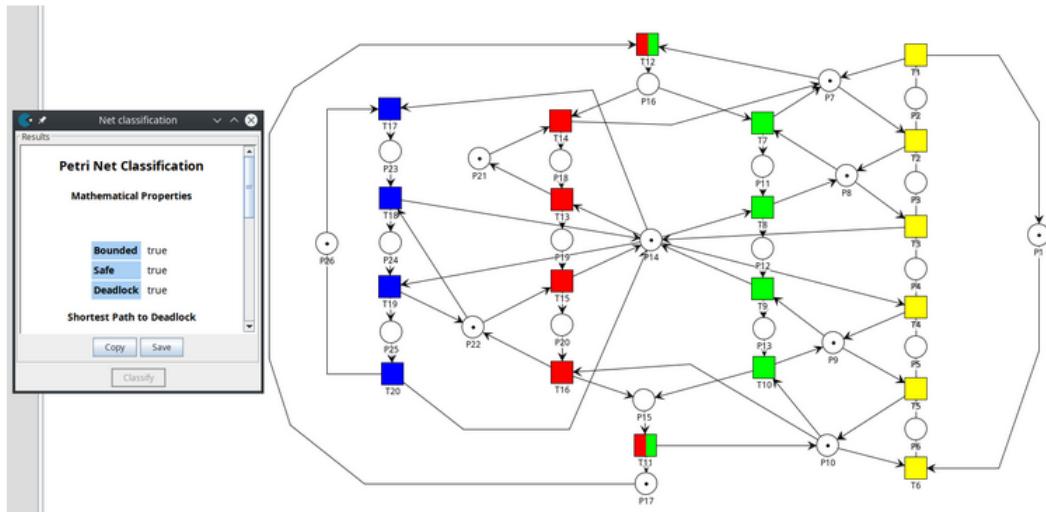


FIGURA 3.36: RdP Ezpeleta controlada.

### 3.4.3.5.2. Caso POPN

#### Análisis estructural

FIGURA 3.37: RdP POPN<sup>9</sup> y sus T-invariantes.

En la Figura 3.37, la plaza  $P_{16}$  forma parte de un conflicto, dado que el marcado de la misma determina la ejecución de un subcircuito de la red u otro, pudiendo seguir dos T-invariantes diferentes.

<sup>9</sup>Figura adaptada del libro *System Modeling and Control with Resource-Oriented Petri Nets* [34].

### Control de la red

Supervisor	Marcado	Transiciones input	Transiciones output	Bad Siphon Controlado
$P_{27}$	1	$\{T_{15}, T_{19}\}$	$\{T_6, T_{12}, T_{17}\}$	$\{P_4, P_{12}, P_{14}, P_{20}, P_{22}, P_{25}\}$
$P_{28}$	1	$\{T_3, T_8\}$	$\{T_6, T_{12}, T_{17}\}$	$\{P_3, P_8, P_{12}, P_{14}, P_{19}, P_{23}, P_{25}\}$
$P_{29}$	1	$\{T_4, T_9\}$	$\{T_6, T_{12}, T_{17}\}$	$\{P_4, P_9, P_{13}, P_{14}, P_{19}, P_{23}, P_{25}\}$
$P_{30}$	1	$\{T_5, T_{10}\}$	$\{T_6, T_{12}, T_{17}\}$	$\{P_5, P_9, P_{10}, P_{15}\}$

CUADRO 3.12: Supervisores: RdP POPN - Análisis 1 y 2.

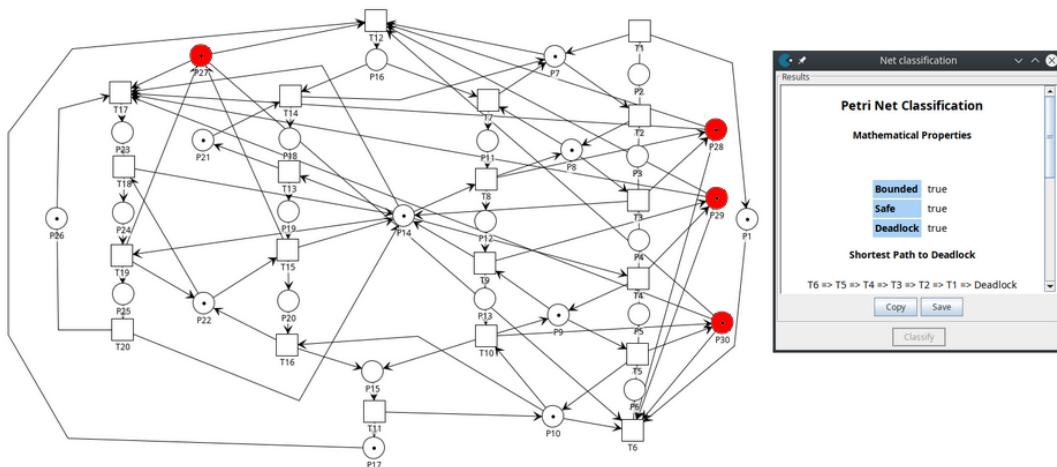


FIGURA 3.38: RdP POPN ejecutando el análisis 1 y 2.

Al agregar los supervisores en la red la misma aún presentaba deadlock, lo que se hizo fue ejecutar la parte 3 del algoritmo para resolver tanto el problema de conflicto y de t\_idle.

```

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

Ingrese:
1 - Primer analisis de la red
2 - Analisis de red con supervisores
3 - Red con supervisores, tratamiento de conflicto y t_idle

Opcion: 3

Eliminar arco desde P27 hasta T6
La transicion en conflicto T7 le tiene que devolver un token al supervisor P27

Se agrego un arco desde T7 hasta P27
Se elimino el arco desde P27 hasta T6
-----
```

FIGURA 3.39: Resultado al ejecutar el análisis 3.

Supervisor	Marcado	Transiciones input	Transiciones output	Bad Siphon Controlado
$P_{27}$	1	$\{T_7, T_{15}, T_{19}\}$	$\{T_{12}, T_{17}\}$	$\{P_4, P_{12}, P_{14}, P_{20}, P_{22}, P_{25}\}$
$P_{28}$	1	$\{T_3, T_8, T_{14}\}$	$\{T_6, T_{12}\}$	$\{P_3, P_8, P_{12}, P_{14}, P_{19}, P_{23}, P_{25}\}$
$P_{29}$	1	$\{T_4, T_9, T_{14}\}$	$\{T_6, T_{12}\}$	$\{P_4, P_9, P_{13}, P_{14}, P_{19}, P_{23}, P_{25}\}$
$P_{30}$	1	$\{T_5, T_{10}, T_{14}\}$	$\{T_6, T_{12}\}$	$\{P_5, P_9, P_{10}, P_{15}\}$

CUADRO 3.13: Supervisores: RdP POPN - Análisis 3.

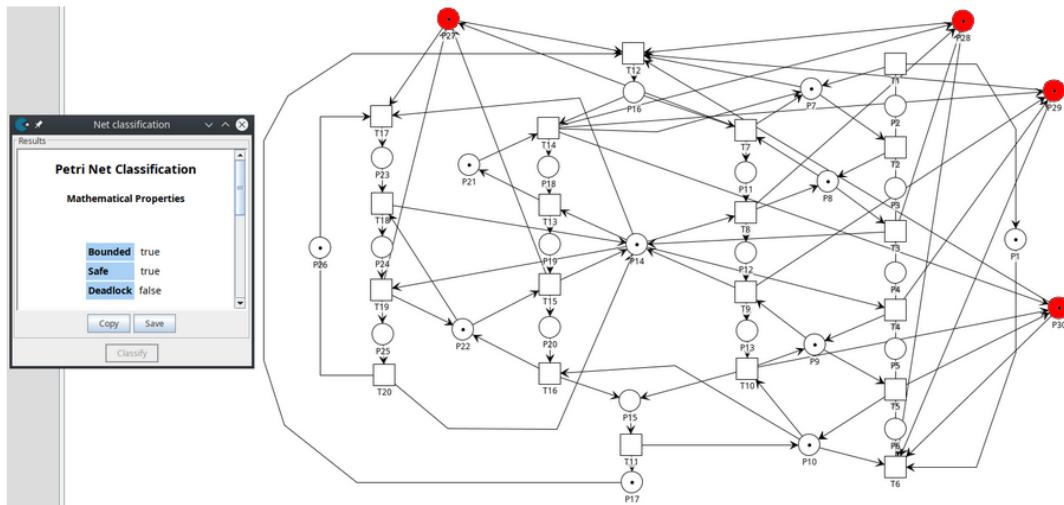
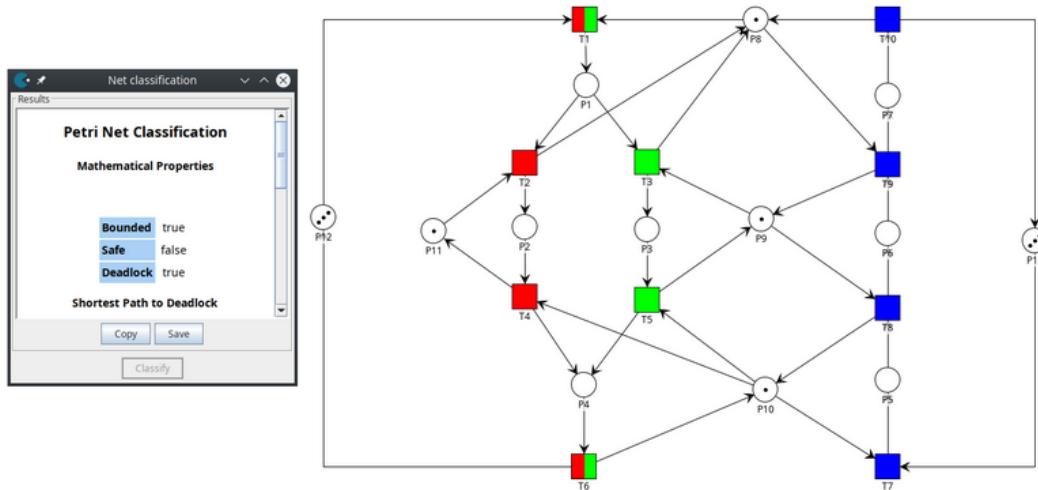


FIGURA 3.40: RdP POPN controlada.

### 3.4.3.5.3. Caso Hospital

#### Análisis estructural

FIGURA 3.41: RdP Hospital <sup>10</sup> y sus T-invariantes.

<sup>10</sup>Figura adaptada del paper publicado por A. Timotei y J. Colom [26].

En este caso particular de red en el que la plaza  $P_1$  forma parte de un conflicto, dado que el marcado de la misma determina la ejecución de un subcircuito de la red u otro, pudiendo seguir dos T-invariantes diferentes.

### Control de la red

Supervisor	Marcado	Transiciones input	Transiciones output	Bad Siphon Controlado
$P_{14}$	3	$\{T_4, T_5, T_9\}$	$\{T_1, T_7\}$	$\{P_4, P_7, P_8, P_9, P_{10}, P_{11}\}$
$P_{15}$	1	$\{T_5, T_8\}$	$\{T_1, T_7\}$	$\{P_4, P_6, P_9, P_{10}\}$

CUADRO 3.14: Supervisores: RdP Hospital - Análisis 1 y 2.

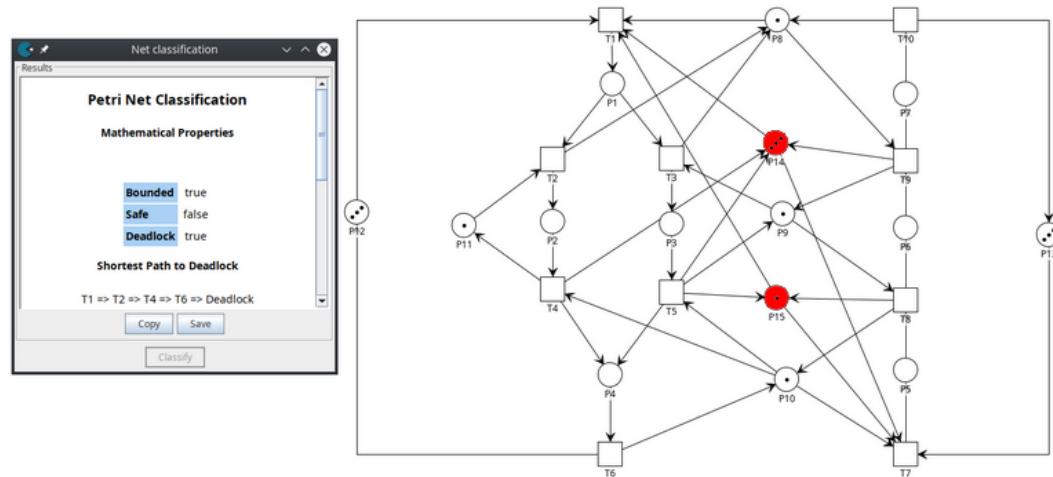


FIGURA 3.42: RdP Hospital ejecutando el análisis 1 y 2.

Al agregar los supervisores mencionados en el cuadro previo en la red, la misma aún presentaba deadlock sin indicar la posibilidad de incorporar nuevos supervisores para su control por lo tanto lo que se hizo fue ejecutar la parte 3 del algoritmo para resolver el problema de conflicto.

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

Ingresé:  
 1 - Primer análisis de la red  
 2 - Análisis de red con supervisores  
 3 - Red con supervisores, tratamiento de conflicto y t\_idle

Opcion: 3

La transición en conflicto T2 le tiene que devolver un token al supervisor P14

Se agrego un arco desde T2 hasta P14

FIGURA 3.43: Resultado al ejecutar el análisis 3.

Supervisor	Marcado	Transiciones input	Transiciones output	Bad Siphon Controlado
$P_{14}$	3	$\{T_4, T_5, T_9\}$	$\{T_1, T_7\}$	$\{P_4, P_7, P_8, P_9, P_{10}, P_{11}\}$
$P_{15}$	1	$\{T_2, T_5, T_8\}$	$\{T_1, T_7\}$	$\{P_4, P_6, P_9, P_{10}\}$

CUADRO 3.15: Supervisores: RdP Hospital - Análisis 3.

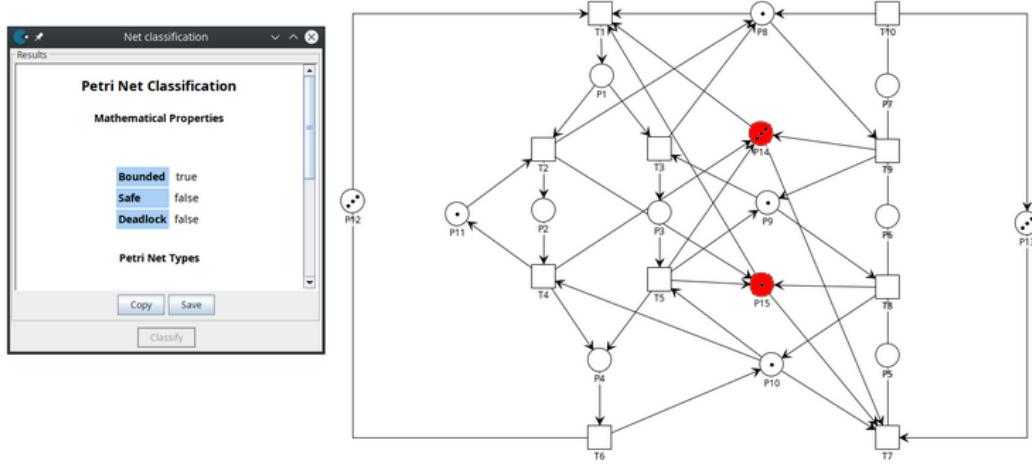
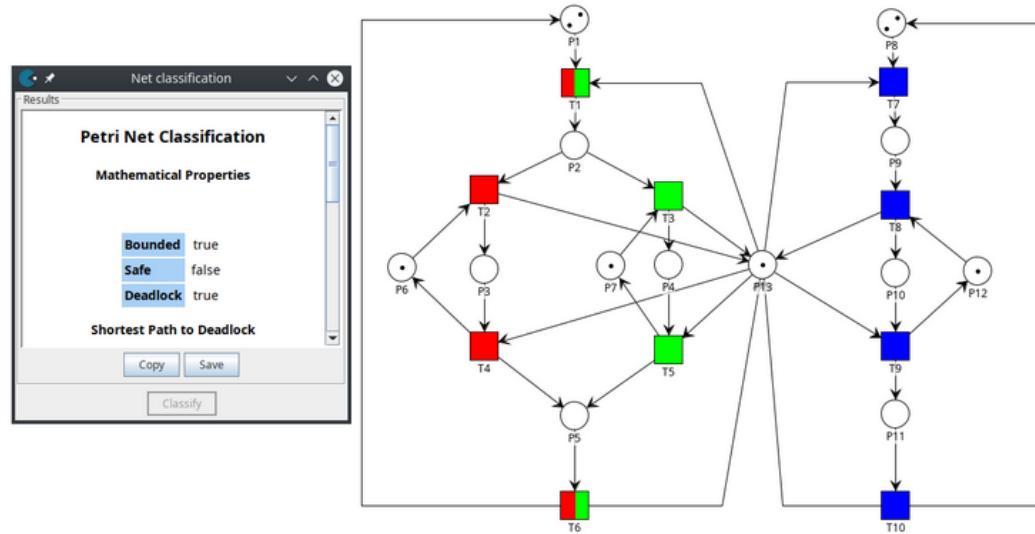


FIGURA 3.44: RdP Hospital controlada.

### 3.4.3.5.4. Caso Huang

#### Análisis estructural

FIGURA 3.45: RDP Huang<sup>11</sup> y sus T-invariantes.

En la Figura 3.45, la plaza  $P_2$  forma parte de un conflicto, dado que el marcado de la misma determina la ejecución de un subcircuito de la red u otro, pudiendo seguir dos T-invariantes diferentes (rojo o verde, en este caso).

### Control de la red

Supervisor	Marcado	Transiciones input	Transiciones output	Bad Siphon Controlado
$P_{14}$	1	$\{T_9\}$	$\{T_1, T_7\}$	$\{P_2, P_5, P_{11}, P_{12}, P_{13}\}$

CUADRO 3.16: Supervisores: RDP Huang - Análisis 1 y 2.

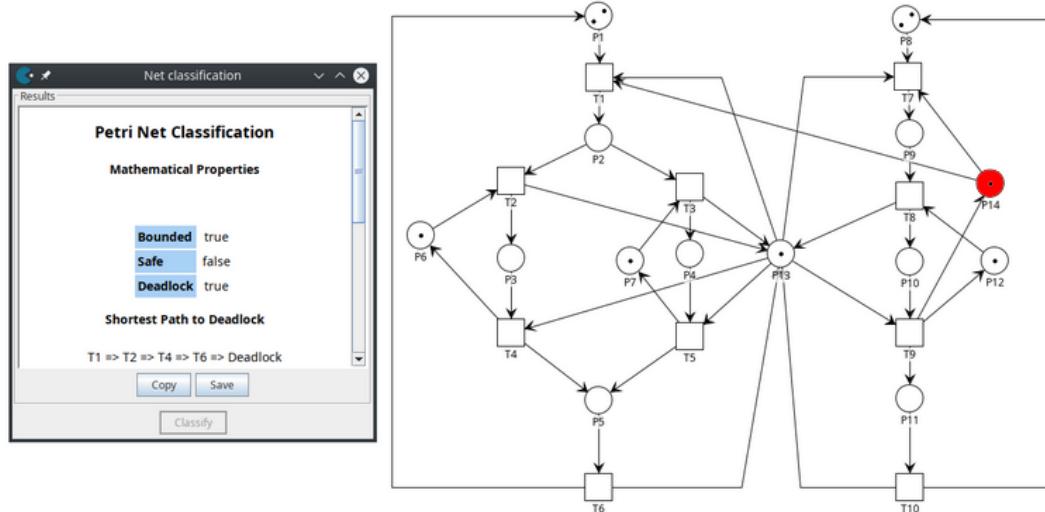


FIGURA 3.46: RDP Huang ejecutando el análisis 1 y 2.

Una vez que se agregó el supervisor mencionado en el cuadro anterior a la red, la misma aún presentaba deadlock sin indicar la posibilidad de incorporar nuevos

<sup>11</sup>Figura adaptada del paper publicado por Yi-Sheng Huang et al. [8].

supervisores para su control por lo tanto lo que se hizo fue ejecutar la parte 3 del algoritmo para resolver el problema de conflicto.

```

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

Ingrese:
1 - Primer analisis de la red
2 - Análisis de red con supervisores
3 - Red con supervisores, tratamiento de conflicto y t_idle

Opcion: 3

La transicion en conflicto T2 le tiene que devolver un token al supervisor P14
La transicion en conflicto T3 le tiene que devolver un token al supervisor P14

Se agrego un arco desde T2 hasta P14
Se agrego un arco desde T3 hasta P14

```

FIGURA 3.47: Resultado al ejecutar el análisis 3.

Supervisor	Marcado	Transiciones input	Transiciones output	Bad Siphon Controlado
$P_{14}$	1	{ $T_2, T_3, T_9$ }	{ $T_1, T_7$ }	{ $P_2, P_5, P_{11}, P_{12}, P_{13}$ }

CUADRO 3.17: Supervisores: RdP Huang - Análisis 3.

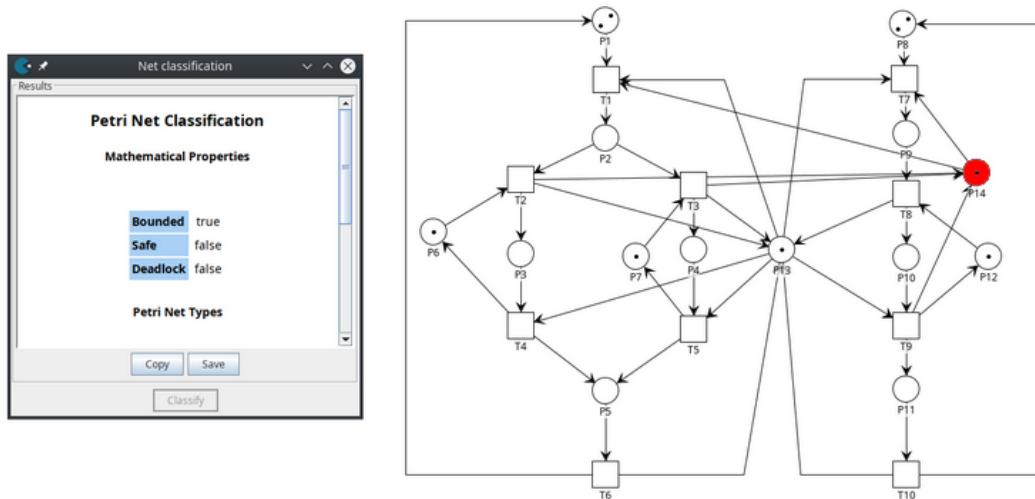


FIGURA 3.48: RdP Huang controlada.

#### 3.4.3.5.5. Caso Ezpeleta v2.0

### Análisis estructural

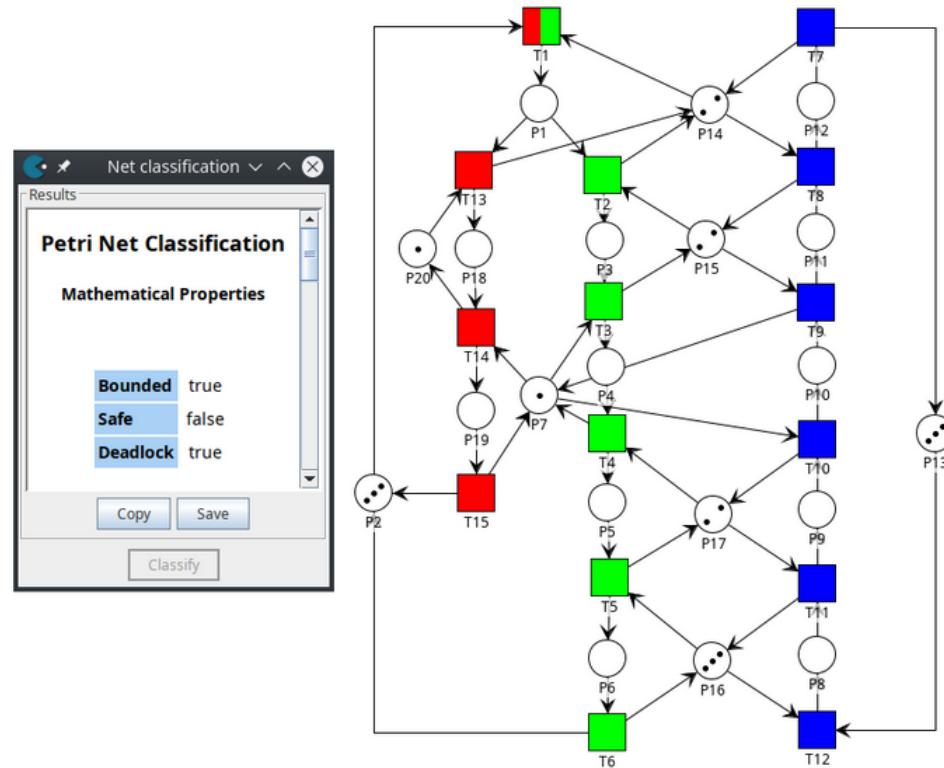


FIGURA 3.49: RdP Ezpeleta v2.0<sup>12</sup> y sus T-invariantes.

En este caso particular la plaza  $P_1$  de la red Figura 3.49 forma parte de un conflicto, dado que el marcado de la misma determina la ejecución de un subcircuito de la red u otro.

### Control de la red

Supervisor	Marcado	Transiciones input	Transiciones output	Bad Siphon Controlado
$P_{21}$	4	$\{T_4, T_9\}$	$\{T_1, T_{12}\}$	$\{P_5, P_7, P_{11}, P_{15}, P_{17}, P_{19}\}$
$P_{22}$	2	$\{T_4, T_{10}\}$	$\{T_1, T_{12}\}$	$\{P_5, P_7, P_{10}, P_{17}, P_{19}\}$
$P_{23}$	2	$\{T_3, T_9\}$	$\{T_1, T_{12}\}$	$\{P_4, P_7, P_{11}, P_{15}, P_{19}\}$

CUADRO 3.18: Supervisores: RdP Ezpeleta v2 - Análisis 1 y 2.

<sup>12</sup>Figura adaptada del paper publicado por Zhong et al. [33].

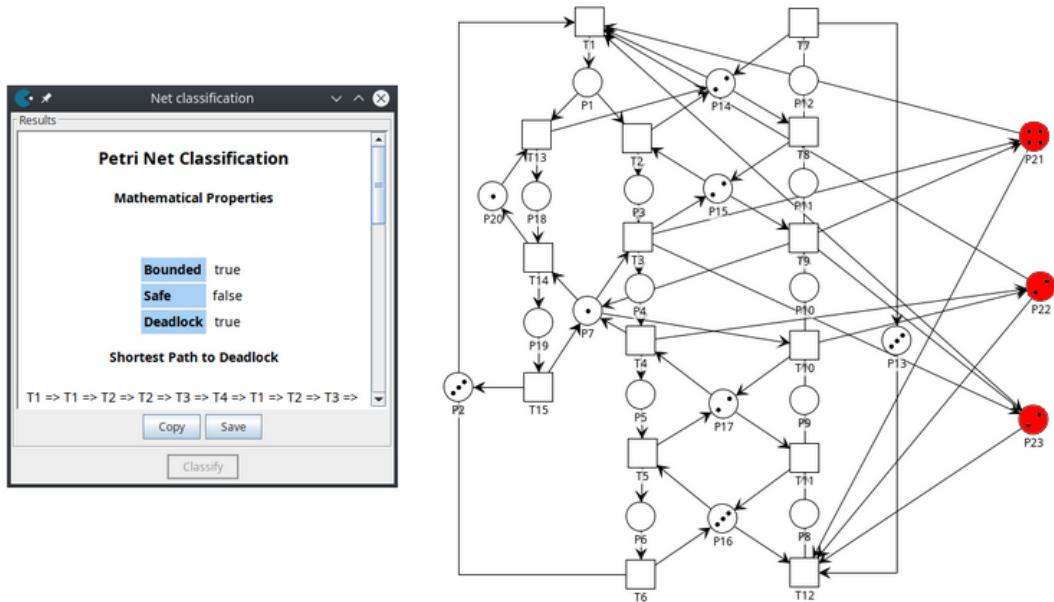


FIGURA 3.50: RdP Ezpeleta v2.0 ejecutando el análisis 1 y 2.

Una vez que se agregaron los supervisores mencionados en el cuadro anterior a la red, la misma aún presentaba deadlock sin indicar la posibilidad de incorporar nuevos supervisores para su control; por lo tanto, se ejecutó la parte 3 del algoritmo para resolver el problema de conflicto.

```
Ingrese:
1 - Primer analisis de la red
2 - Análisis de red con supervisores
3 - Red con supervisores, tratamiento de conflicto y t_idle
```

Opcion: 3

```
La transicion en conflicto T13 le tiene que devolver un token al supervisor P21
La transicion en conflicto T13 le tiene que devolver un token al supervisor P22
La transicion en conflicto T13 le tiene que devolver un token al supervisor P23
```

```
Se agrego un arco desde T13 hasta P21
Se agrego un arco desde T13 hasta P22
Se agrego un arco desde T13 hasta P23
```

FIGURA 3.51: Resultado al ejecutar el análisis 3.

Supervisor	Marcado	Transiciones input	Transiciones output	Bad Siphon Controlado
$P_{21}$	4	$\{T_4, T_9, T_{13}\}$	$\{T_1, T_{12}\}$	$\{P_5, P_7, P_{11}, P_{15}, P_{17}, P_{19}\}$
$P_{22}$	2	$\{T_4, T_{10}, T_{13}\}$	$\{T_1, T_{12}\}$	$\{P_5, P_7, P_{10}, P_{17}, P_{19}\}$
$P_{23}$	2	$\{T_3, T_9, T_{13}\}$	$\{T_1, T_{12}\}$	$\{P_4, P_7, P_{11}, P_{15}, P_{19}\}$

CUADRO 3.19: Supervisores: RdP Ezpeleta v2 - Análisis 3.

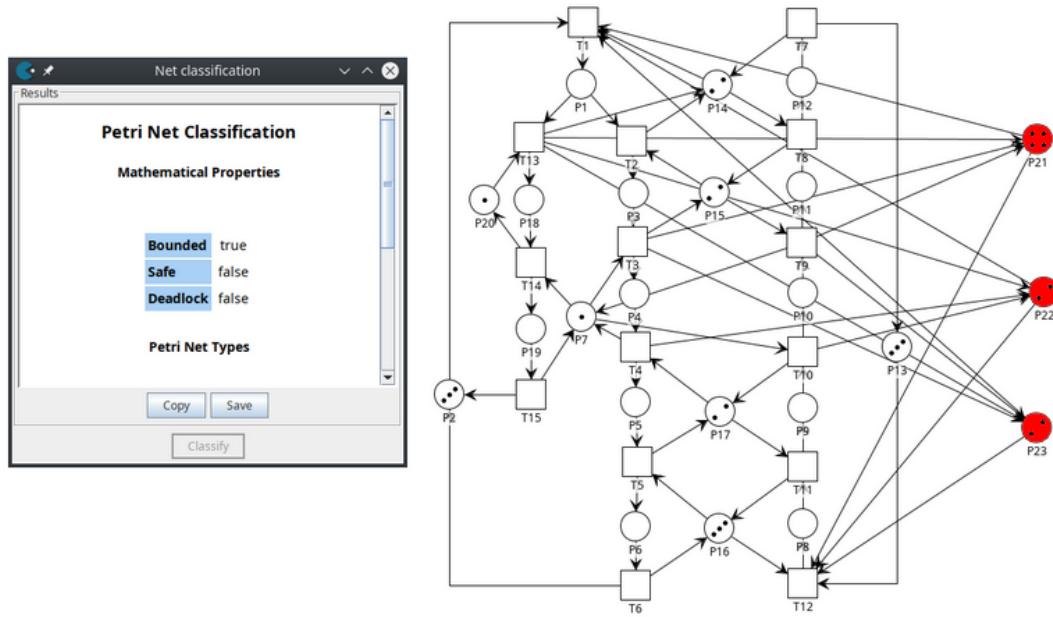


FIGURA 3.52: RdP Ezpeleta v2.0 controlada.

### 3.4.4. Conclusión

Con la incorporación de las nuevas características al algoritmo inicial, se logró generalizar el análisis de las diferentes redes de Petri propuestas para el estudio ( $S^3PR$ ).

En cada caso sometido a estudio se logró resolver los problemas de deadlock, sin tener la necesidad de aplicar el criterio de división a cada red con el que se venía trabajando anteriormente. Incluso en algunos casos la solución se optimizó al lograr controlar la red con una menor cantidad de supervisores.

Al momento de la elección de los supervisores a agregar entre los propuestos por el algoritmo, se necesita tener un orden dado que la elección incorrecta de los supervisores puede llevar a divergir el problema y no haya solución. El criterio con el que se llegó a la solución, fue tomar los supervisores que más se repetían y aquellos cuyos arcos eran mínimos (es decir, si un supervisor tiene como arcos de entrada  $\{T_1, T_5, T_9\}$ , y otro supervisor tiene  $\{T_1, T_5\}$  y presentan las mismas transiciones de salida, se elige el segundo supervisor).

Esta versión final del algoritmo sigue teniendo en cuenta el conflicto y los T-invariantes pero ya no con la finalidad antes propuesta que era la división de las mismas para su análisis; sino que se observó que en algunos casos, la incorporación de supervisores ocasionaba la pérdida de el/los T-invariante/s de la red original (sin controlar) y esto llevaba a que la red aún permaneciera bloqueada. Para solucionar esto se llevó a cabo un análisis del conflicto y de las transiciones en estado idle, para determinar qué arcos debían agregarse/quitarse para preservar el/los T-invariante/s de la red original llevando a una red sin estados de deadlock.

### 3.4.5. Extensión: algoritmo v4.1

En esta extensión se logró una interfaz para automatizar las tareas que en versiones previas se realizaban de manera manual, como la colocación de supervisores (Figura 3.53) y la tarea de incorporar/eliminar los arcos indicados por el análisis *Red con supervisores, tratamiento de conflictos y t\_idle* (Figura 3.54). La misma se realiza a través de un script que toma de base el archivo '.pflow' (extensión del Petrinator) para realizar las modificaciones indicadas por el algoritmo.

```
id= 4
Sifon a controlar: 3
Plazas del Sifon: ['P4', 'P12', 'P14', 'P20', 'P22', 'P25']
Marcado del supervisor 1
Transicion output: 6
Transicion output: 12
Transicion output: 17
Transicion input: 15
Transicion input: 19

¿Agregar supervisor?(S/N) S
AGREGA EL ID: 4
```

FIGURA 3.53: Incorporación del supervisor de manera automática.

```
Ingrese:
1 - Primer analisis de la red
2 - Análisis de red con supervisores
3 - Red con supervisores, tratamiento de conflicto y t_idle

Opcion: 3

Eliminar arco desde P27 hasta T6
La transicion en conflicto T7 le tiene que devolver un token al supervisor P27

Se agrego un arco desde T7 hasta P27
Se elimino el arco desde P27 hasta T6
```

FIGURA 3.54: Ejecución del análisis *Red con supervisores, tratamiento de conflictos y t\_idle* de manera automática.

Además, se agregó una extensión de “Reload” al software Petrinator para que vuelva a cargar el archivo '.pflow' para visualizar las modificaciones realizadas. Estas modificaciones pueden observarse en la ejecución completa y detallada del algoritmo en el Apéndice B.

Por ultimo, se observó que al extraer el grafo de alcanzabilidad los estados de deadlock presentes en la red en algunos casos pueden ser alcanzables a través de diferentes estados, es este el motivo por el que el numero de supervisores era tan alto y hasta en ciertos casos se repetían las mismas soluciones. Por este motivo, se decidió filtrar los supervisores mostrándolos una única vez evitando así redundancias.

## 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 Auto y Hiuxia

Estas redes modelan la ejecución concurrente de un AMS. En las mismas 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. Un problema AMS 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 Auto

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

#### 4.1.1.2. Análisis estructural Auto

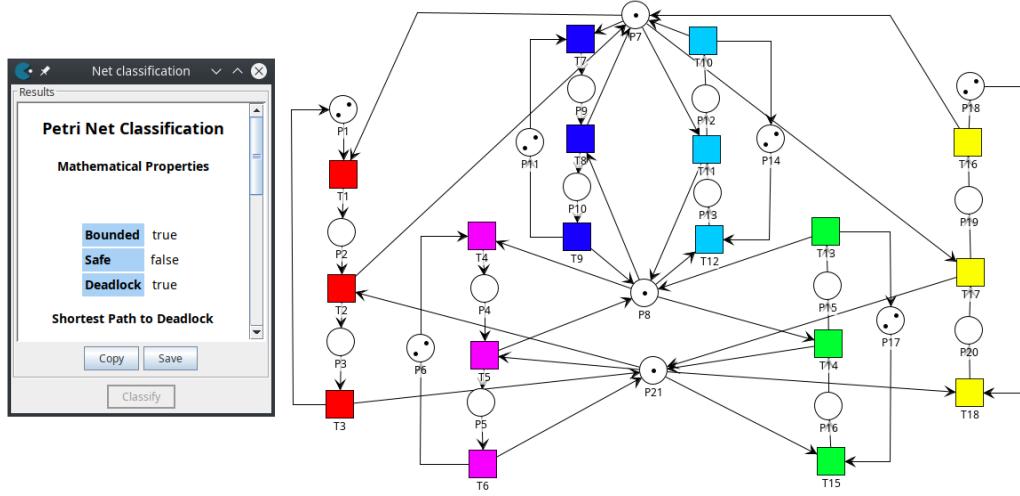


FIGURA 4.1: RdP Auto<sup>1</sup> y sus T-invariantes.

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

##### 4.1.1.2.1. Control de la red

Al agregar los supervisores en la red ésta aún presentaba deadlock, lo que se hizo fue ejecutar la parte 3 del algoritmo para resolver tanto el problema de conflicto y de t\_idle, logrando de esta manera el control de la red.

Supervisor	Marcado	Transiciones input	Transiciones output	Bad Siphon Controlado
$P_{22}$	1	$\{T_2, T_8, T_{11}, T_{13}\}$	$\{T_1, T_7, T_{12}, T_{15}\}$	$\{P_2, P_5, P_9, P_{10}, P_{13}, P_{17}, P_{19}, P_{20}\}$
$P_{23}$	1	$\{T_2, T_{13}, T_{17}\}$	$\{T_1, T_{15}, T_{18}\}$	$\{P_3, P_6, P_8, P_{10}, P_{14}, P_{17}, P_{19}, P_{21}\}$
$P_{24}$	1	$\{T_2, T_5, T_{14}\}$	$\{T_1, T_4, T_{15}\}$	$\{P_3, P_6, P_9, P_{11}, P_{13}, P_{16}, P_{20}, P_{21}\}$
$P_{25}$	2	$\{T_2, T_5, T_8, T_{11}, T_{14}, T_{17}\}$	$\{T_1, T_4, T_7, T_{12}, T_{15}, T_{18}\}$	$\{P_3, P_6, P_9, P_{10}, P_{13}, P_{17}, P_{19}, P_{20}, P_{21}\}$

CUADRO 4.1: Supervisores: RdP Auto

<sup>1</sup>Figura adaptada del paper publicado por Huixia Liu et al. [17]

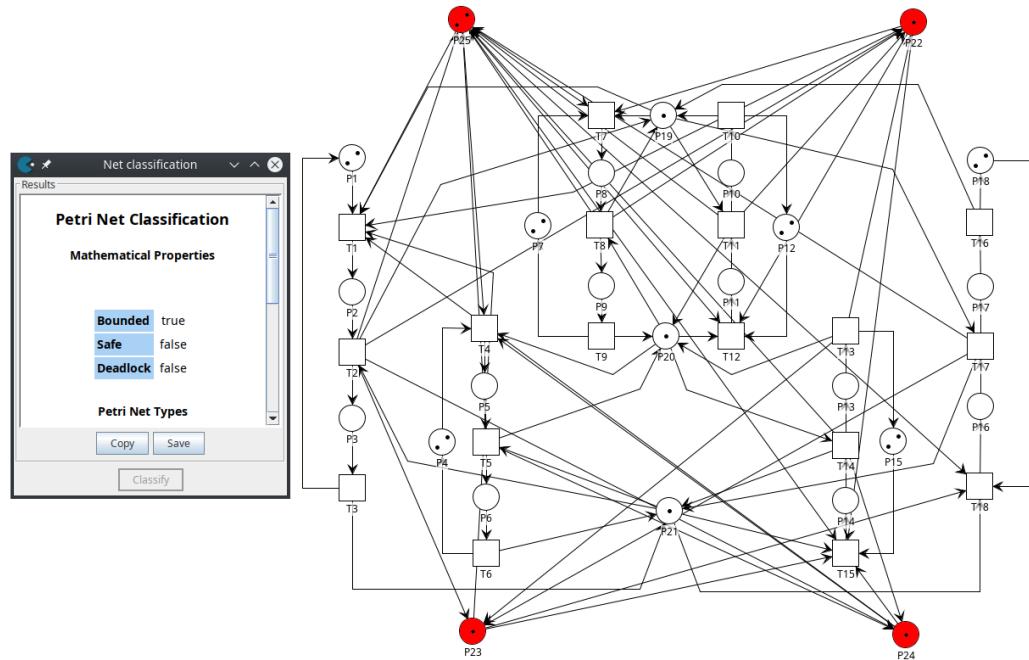
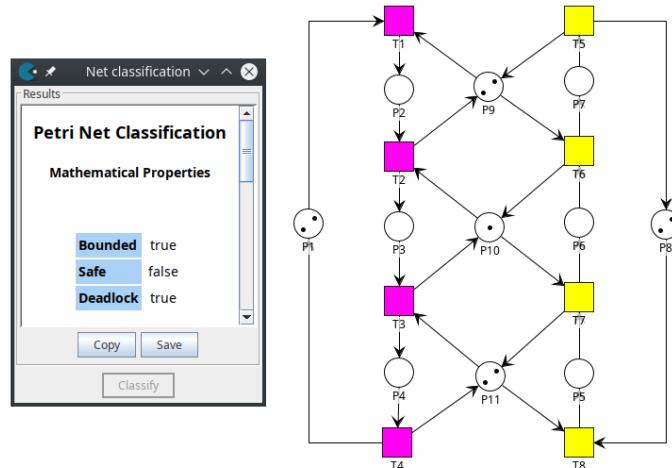


FIGURA 4.2: RdP Auto controlada.

#### 4.1.1.3. Características generales red Hiuxia

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

#### 4.1.1.4. Análisis estructural red Hiuxia

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

En la Figura 4.3, se presentan los T-invariantes en diferentes colores.

---

<sup>2</sup>Figura adaptada del paper publicado por *Huixia Liu et al.* [17]

#### 4.1.1.4.1. Control de la red

Para alcanzar la vivacidad de esta red bastó con agregar los supervisores y no fue necesario ejecutar la parte 3 del algoritmo dado que al agregar los mismos se alcanzó el control de la red, eliminando las situaciones de deadlock.

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 4.2: Supervisores: RdP Hiuxia

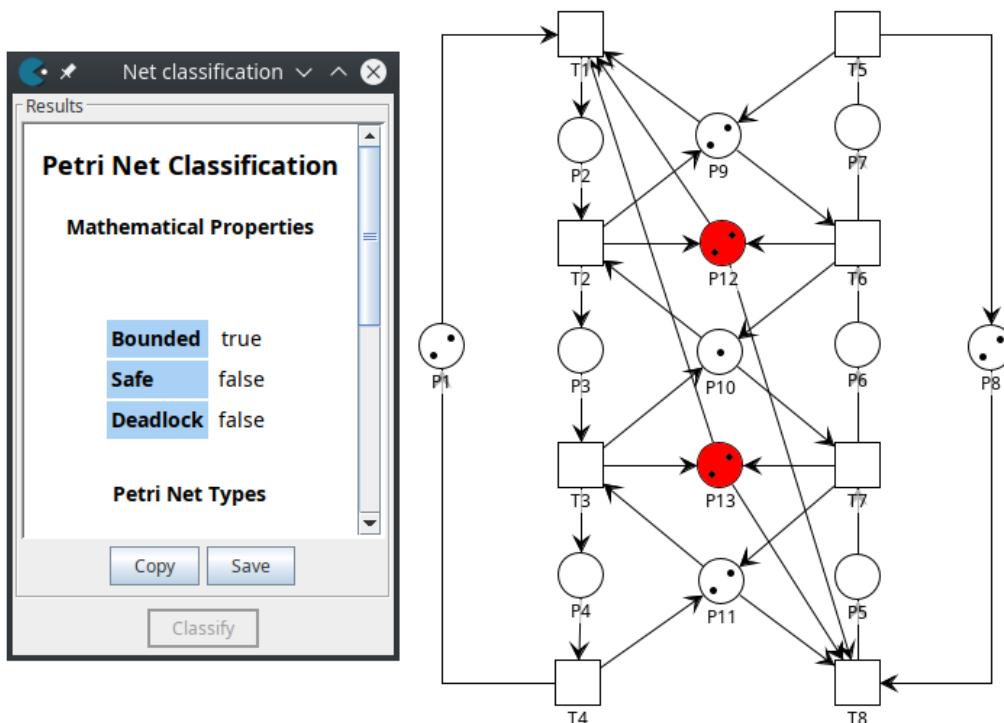


FIGURA 4.4: RdP Hiuxia controlada.

#### 4.1.2. Casos YiFan (1,2,3)

Estas tres redes<sup>3</sup> presentes en el paper de YiFan et al. [5] también son del tipo AMS. Este sistema consiste en un conjunto de estaciones interconectadas para el procesamiento de materiales que es capaz de procesar automáticamente una amplia variedad de tipos de piezas de manera simultánea y controlada por computadoras. Un AMS tiene características de alto grado de automatización, de integración y de flexibilidad. En las redes existen ciertas plazas que simulan la disponibilidad de recursos y un control incorrecto de estos en la ejecución de los procesos de trabajo, puede conducir a situaciones de deadlock.

<sup>3</sup>En estas RdP fueron modificados los pesos de los arcos reduciéndolos a uno para adaptarlas al tipo de red que admite el algoritmo.

#### 4.1.2.1. Características generales YiFan1

- Los recursos de la red están representados por las plazas  $\{P_{12}, P_{13}, P_{14}, P_{15}\}$ .

#### 4.1.2.2. Análisis estructural YiFan1

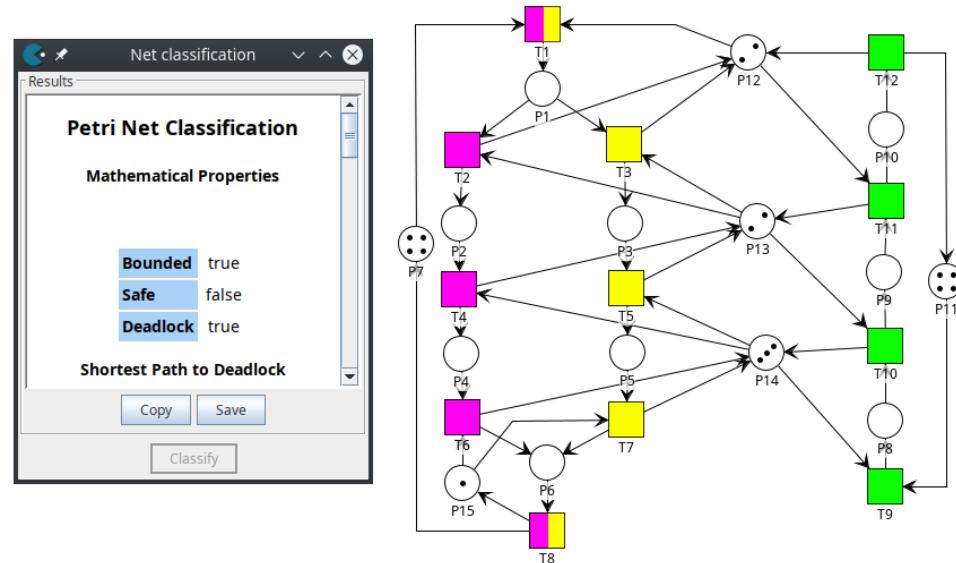


FIGURA 4.5: RdP YiFan1<sup>4</sup> y sus T-invariantes.

En la Figura 4.5, se presentan los diferentes T-invariantes en diferentes colores.

#### 4.1.2.2.1. Control de la red

Para alcanzar la vivacidad de esta red bastó con agregar los supervisores y no fue necesario ejecutar la parte 3 del algoritmo dado que al agregar los mismos se alcanzó el control de la red, eliminando las situaciones de deadlock.

Supervisor	Marcado	Transiciones input	Transiciones output	Bad Siphon Controlado
$P_{16}$	3	$\{T_2, T_3, T_{11}\}$	$\{T_1, T_9\}$	$\{P_2, P_3, P_{10}, P_{12}, P_{13}\}$
$P_{17}$	4	$\{T_4, T_5, T_{10}\}$	$\{T_1, T_9\}$	$\{P_4, P_5, P_9, P_{13}, P_{14}\}$

CUADRO 4.3: Supervisores: RdP YiFan1

<sup>4</sup>Figura adaptada del paper publicado por YiFan Hou et al. [5]

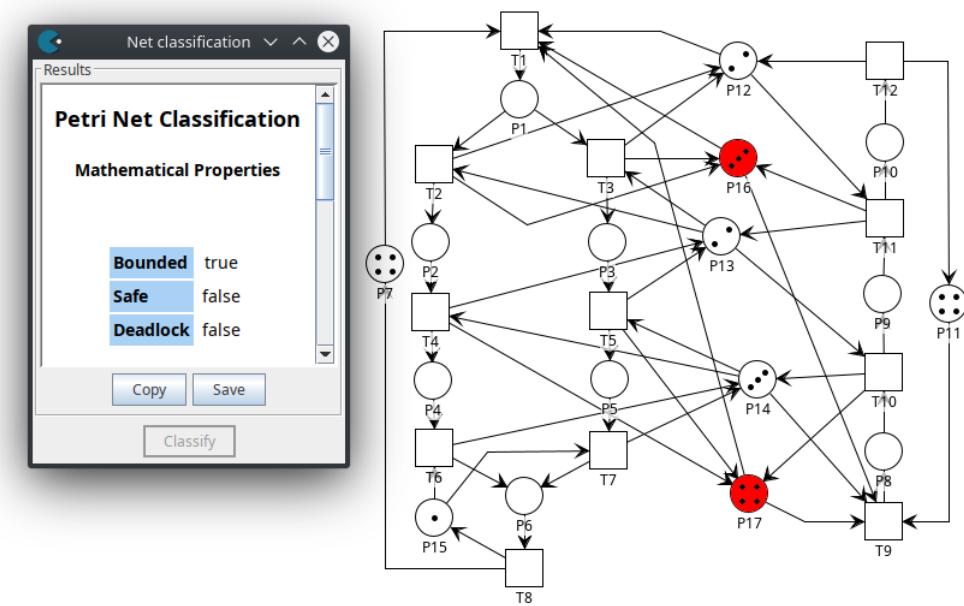
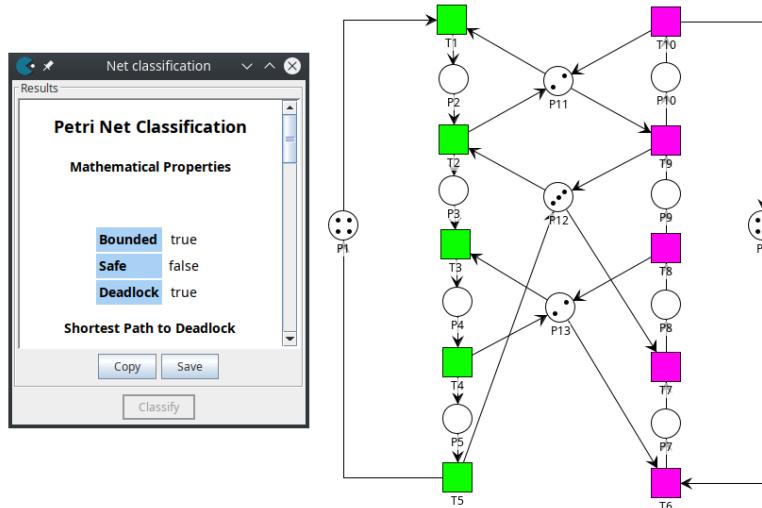


FIGURA 4.6: RdP YiFan1 controlada.

#### 4.1.2.3. Características generales YiFan2

- Los recursos de la red están representados por las plazas  $\{P_{11}, P_{12}, P_{13}\}$ .

#### 4.1.2.4. Análisis estructural YiFan2

FIGURA 4.7: RdP YiFan2<sup>5</sup> y sus T-invariantes.

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

<sup>5</sup>Figura adaptada del paper publicado por YiFan Hou et al. [5]

#### 4.1.2.4.1. Control de la red

Para alcanzar la vivacidad de esta red bastó con agregar los supervisores y no fue necesario ejecutar la parte 3 del algoritmo dado que al agregar los mismos se alcanzó el control de la red, eliminando las situaciones de deadlock.

Supervisor	Marcado	Transiciones input	Transiciones output	Bad Siphon Controlado
$P_{14}$	6	$\{T_4, T_9\}$	$\{T_1, T_6\}$	$\{P_4, P_5, P_8, P_{10}, P_{11}, P_{12}, P_{13}\}$
$P_{15}$	4	$\{T_4, T_8\}$	$\{T_1, T_6\}$	$\{P_4, P_5, P_8, P_9, P_{12}, P_{13}\}$
$P_{16}$	4	$\{T_2, T_9\}$	$\{T_1, T_6\}$	$\{P_3, P_4, P_5, P_{10}, P_{11}, P_{12}\}$

CUADRO 4.4: Supervisores: RdP YiFan2

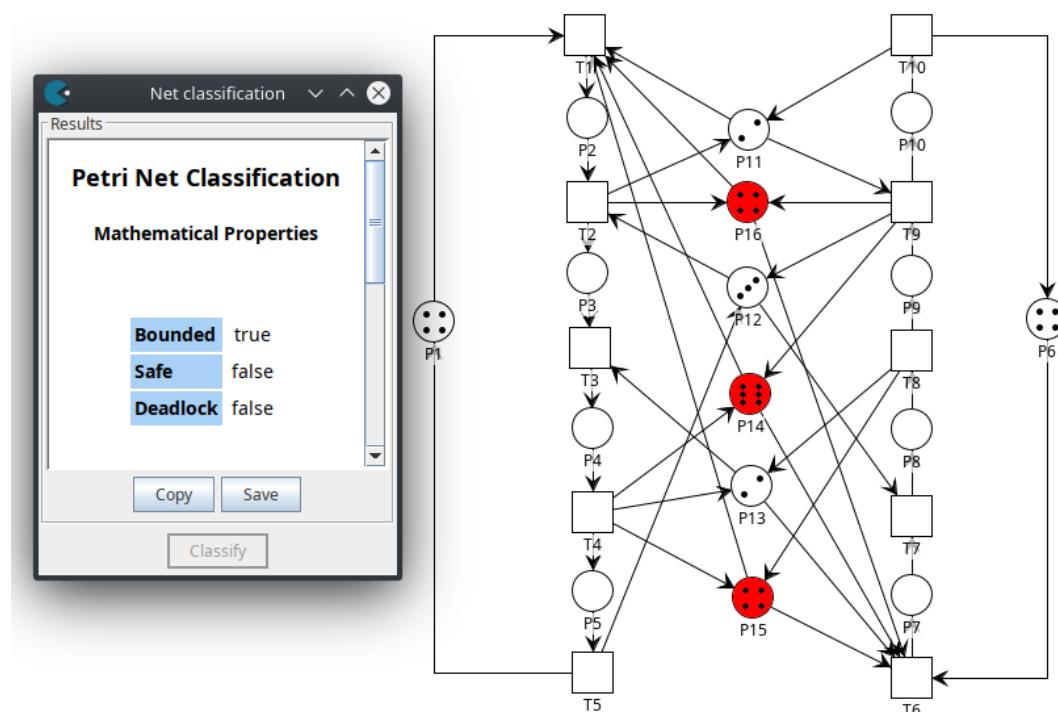


FIGURA 4.8: RdP YiFan2 controlada.

#### 4.1.2.5. Características generales YiFan3

- Los recursos de la red están representados por las plazas  $\{P_5, P_6\}$ .

#### 4.1.2.6. Análisis estructural YiFan3

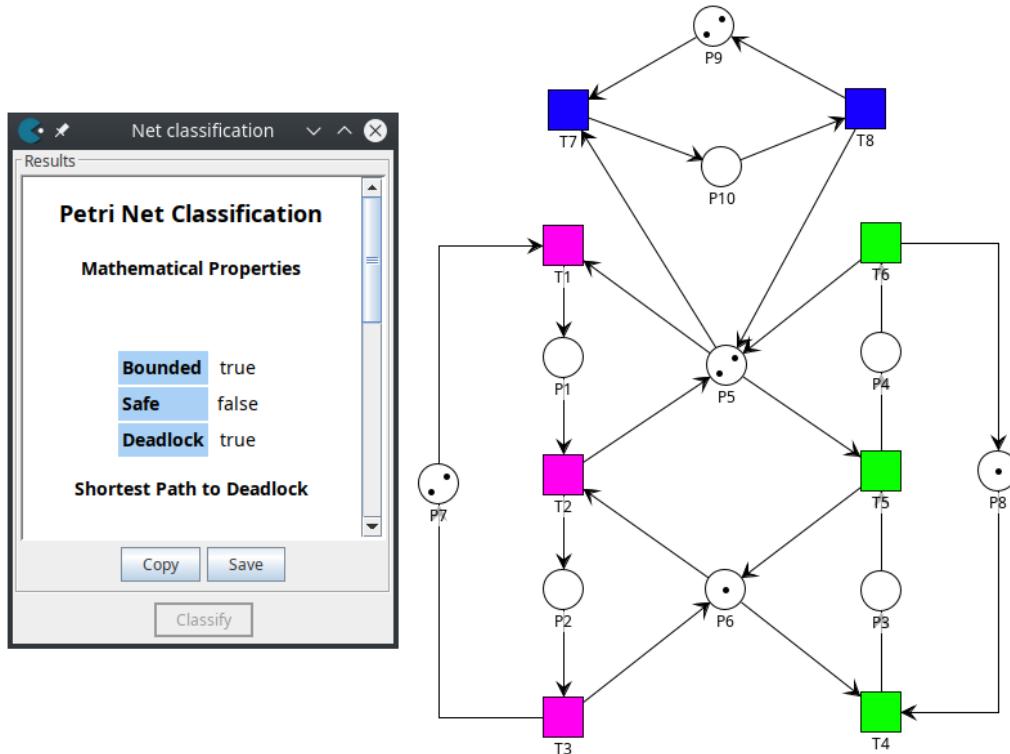


FIGURA 4.9: RdP YiFan3<sup>6</sup> y sus T-invariantes.

En la Figura 4.9, se presentan los T-invariantes en diferentes colores.

##### 4.1.2.6.1. Control de la red

Al agregar los supervisores en la red ésta aún presentaba deadlock, lo que se hizo fue ejecutar la parte 3 del algoritmo para resolver tanto el problema de conflicto y de t\_idle, logrando de esta manera el control de la red.

Supervisor	Marcado	Transiciones input	Transiciones output	Bad Siphon Controlado
$P_{11}$	2	$\{T_2, T_5\}$	$\{T_1, T_4\}$	$\{P_2, P_4, P_5, P_6, P_{10}\}$

CUADRO 4.5: Supervisores: RdP YiFan3

<sup>6</sup>Figura adaptada del paper publicado por YiFan Hou et al. [5]

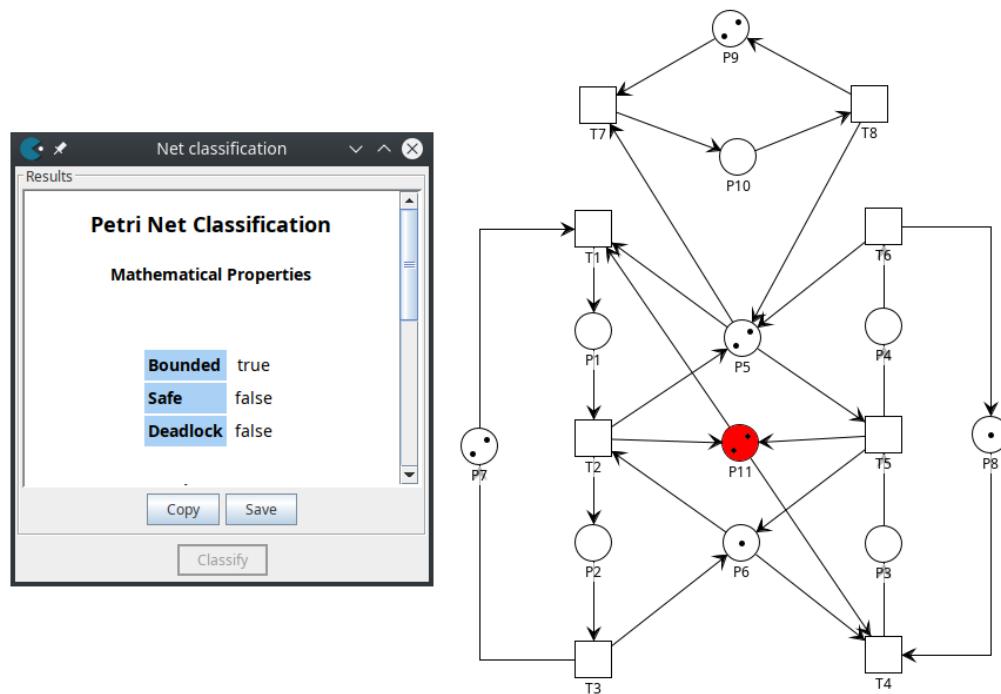


FIGURA 4.10: RdP YiFan3 controlada.

#### 4.1.3. 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.

##### 4.1.3.1. 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  $\{P_{21}, P_{23}, P_{25}, P_{27}, P_{28}, P_{29}\}$ .

#### 4.1.3.2. Análisis estructural

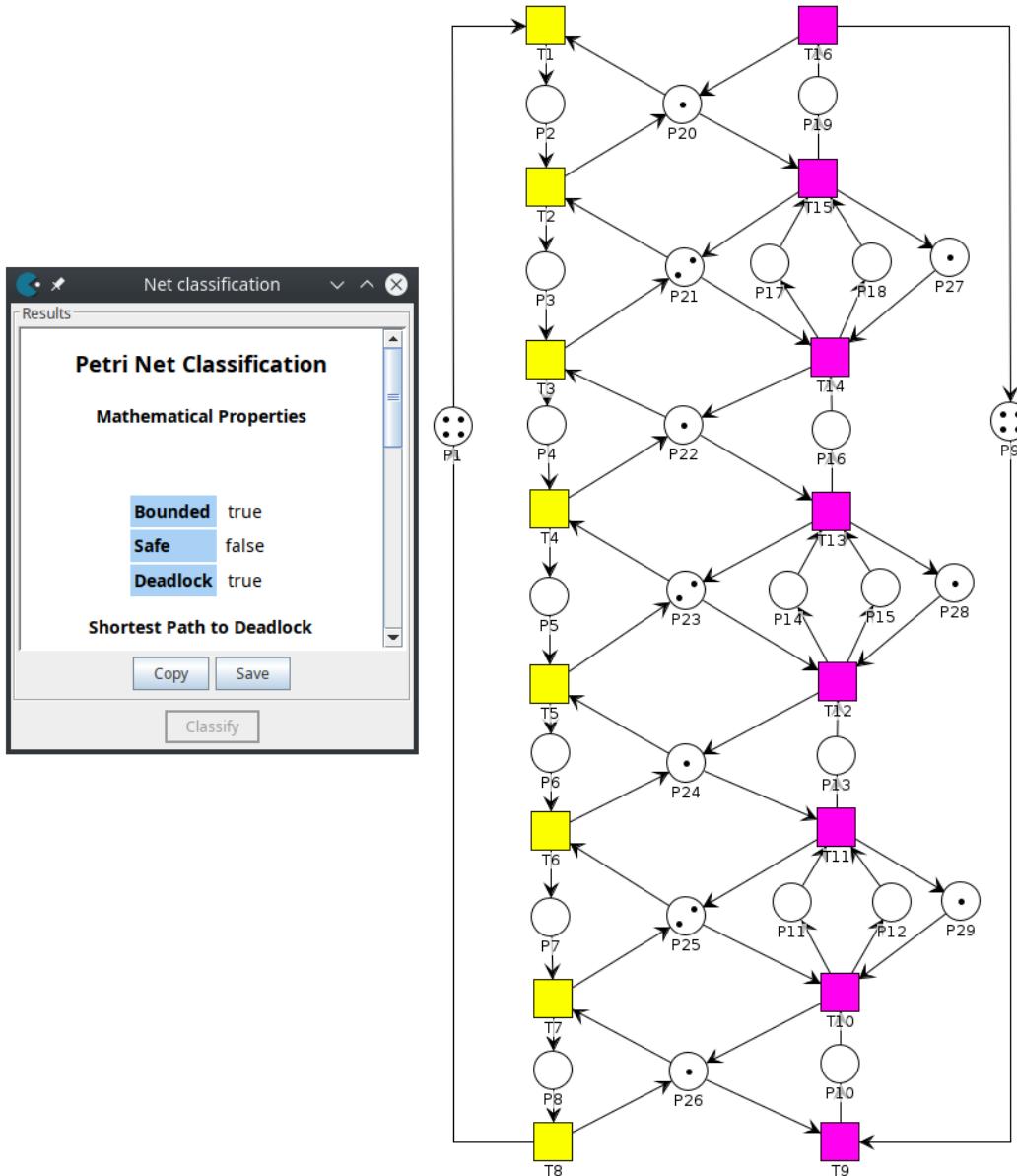


FIGURA 4.11: RdP JianChao<sup>7</sup> y sus T-invariantes.

En la Figura 4.11<sup>8</sup>, se presentan los T-invariantes en diferentes colores.

##### 4.1.3.2.1. Control de la red

Para alcanzar la vivacidad de esta red bastó con agregar los supervisores y no fue necesario ejecutar la parte 3 del algoritmo dado que al agregar los mismos se alcanzó el control de la red, eliminando las situaciones de deadlock.

<sup>7</sup>Figura adaptada del paper publicado por JianChao Lou et al. [18]

<sup>8</sup>En esta RdP fue modificado el peso de los arcos reduciéndolos a uno para adaptarlas al tipo de red que admite el algoritmo.

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 4.6: Supervisores: RdP JianChao

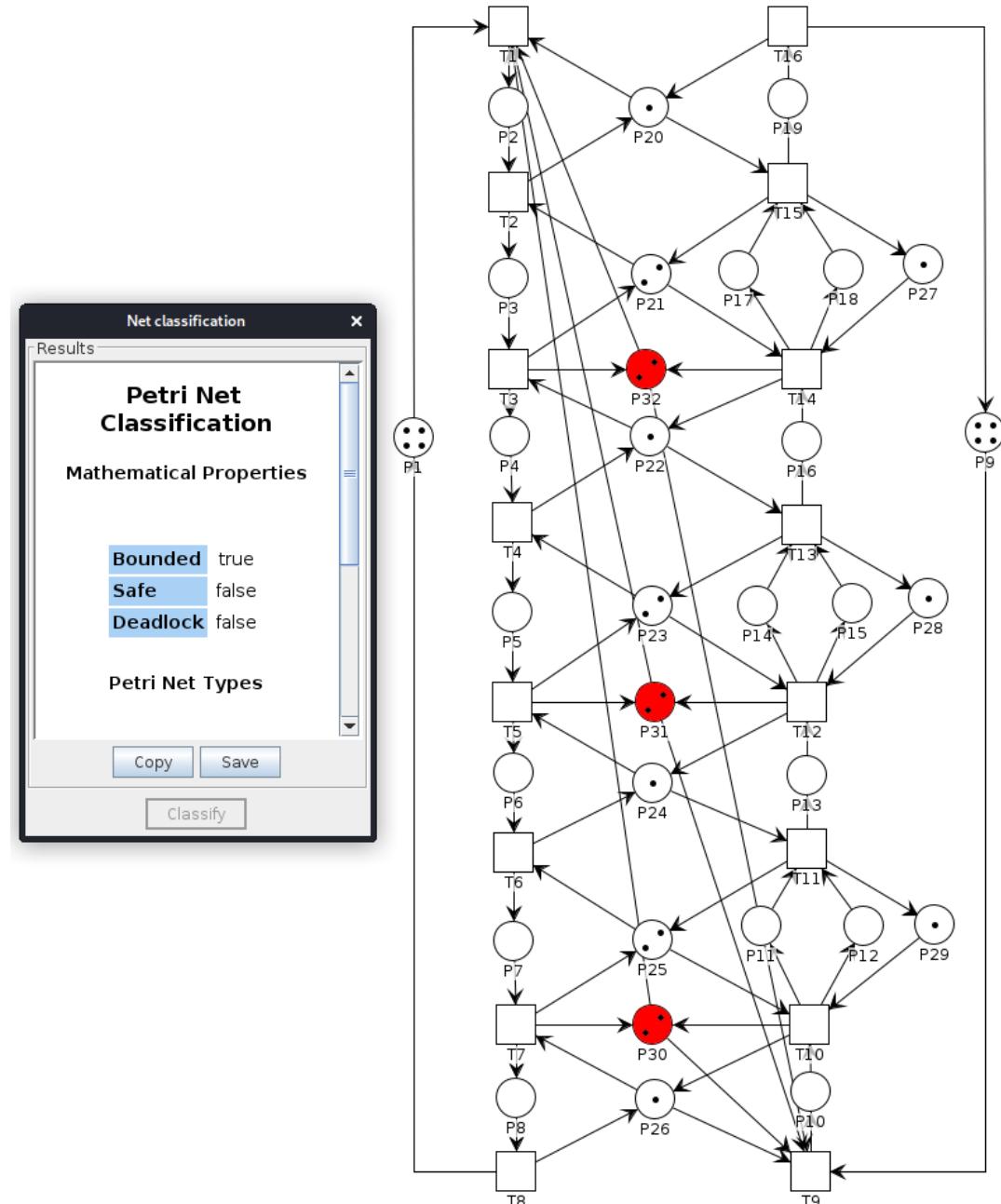


FIGURA 4.12: RdP JianChao controlada.

#### 4.1.4. Caso Zhiwuli

Esta red modela la ejecución concurrente de un FMS. El mismo consta de 3 procesos productivos que comparten 2 recursos y un control incorrecto de estos en la ejecución de los procesos de trabajo, puede conducir a situaciones de deadlock.

##### 4.1.4.1. Características generales

- Los recursos de la red están representados por las plazas  $\{P_7, P_8\}$ .

##### 4.1.4.2. Análisis estructural

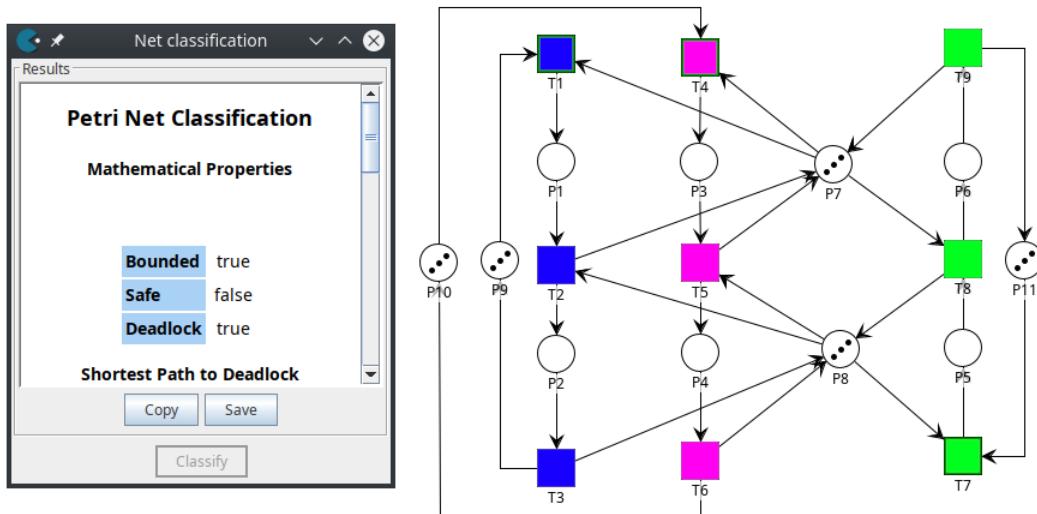


FIGURA 4.13: RdP Zhiwuli <sup>9</sup> y sus T-invariantes.

En la Figura 4.13 <sup>10</sup>, se presentan los T-invariantes en diferentes colores.

##### 4.1.4.2.1. Control de la red

Para alcanzar la vivacidad de esta red bastó con agregar los supervisores y no fue necesario ejecutar la parte 3 del algoritmo dado que al agregar los mismos se alcanzó el control de la red, eliminando las situaciones de deadlock.

Supervisor	Marcado	Transiciones input	Transiciones output	Bad Siphon Controlado
$P_{12}$	5	$\{T_2, T_5, T_8\}$	$\{T_1, T_4, T_7\}$	$\{P_2, P_4, P_6, P_7, P_8\}$

CUADRO 4.7: Supervisores: RdP Zhiwuli

<sup>9</sup>Figura adaptada del paper publicado por ZhiWu Li et al. [14]

<sup>10</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.

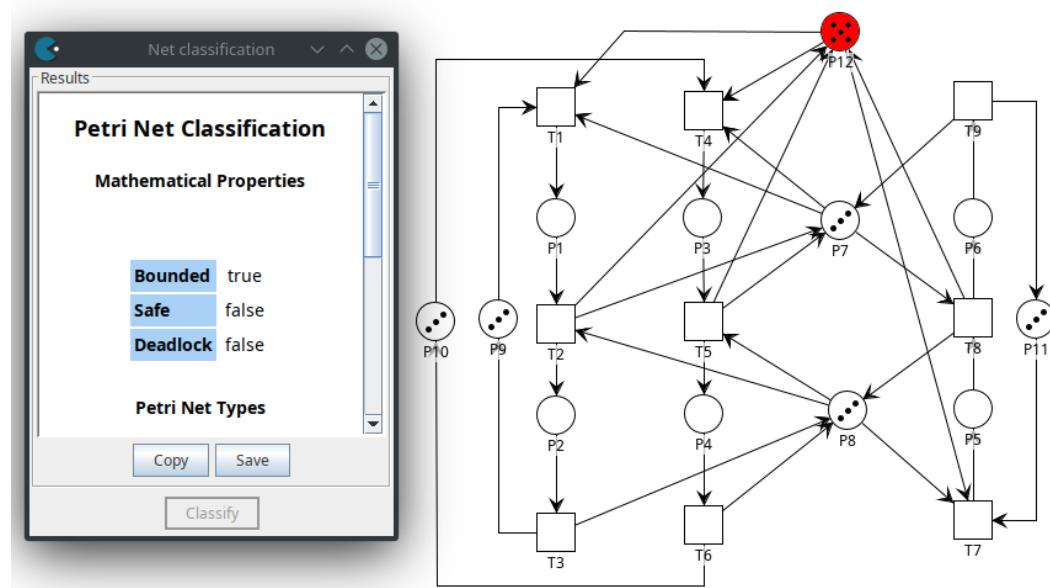


FIGURA 4.14: RdP Zhiwuli controlada.

#### 4.1.5. Caso Fanti

La red representa un AMS, la cual consta de un conjunto de estaciones de trabajo, cada una de las cuales es capaz de procesar piezas de diferente tipo de acuerdo con una secuencia de operaciones prescrita. Para el desarrollo de estas piezas la red presenta 4 recursos compartidos y una mala gestión de los mismos desencadenará el bloqueo de la red.

##### 4.1.5.1. Características generales

- Los recursos de la red están representados por las plazas  $\{P_{10}, P_{11}, P_{12}, P_{13}\}$ .

#### 4.1.5.2. Análisis estructural

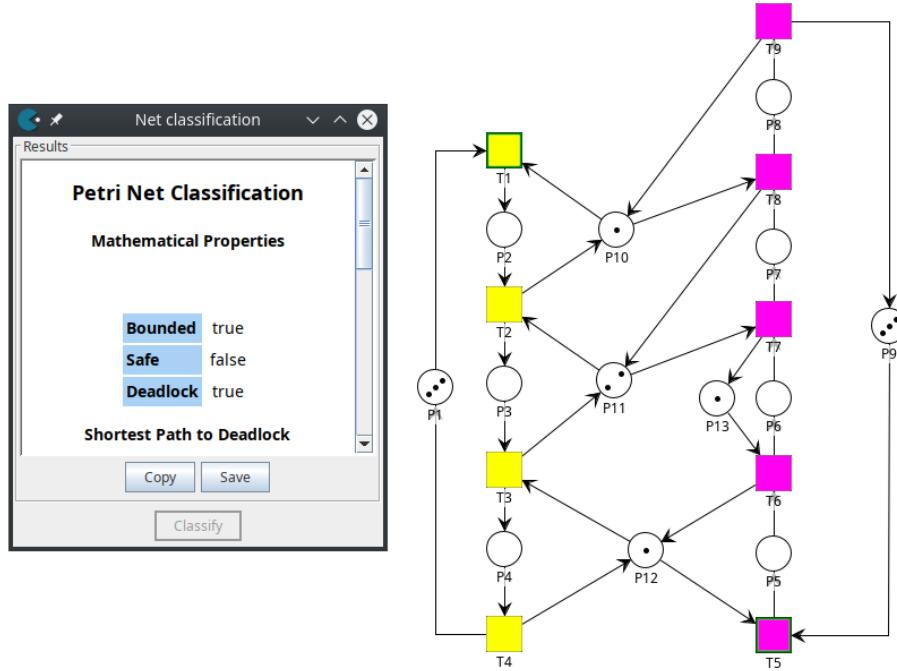


FIGURA 4.15: RdP Fanti<sup>11</sup> y sus T-invariantes.

En la Figura 4.15 , se presentan los T-invariantes en diferentes colores.

##### 4.1.5.2.1. Control de la red

Para alcanzar la vivacidad de esta red bastó con agregar los supervisores y no fue necesario ejecutar la parte 3 del algoritmo dado que al agregar los mismos se alcanzó el control de la red, eliminando las situaciones de deadlock.

Supervisor	Marcado	Transiciones input	Transiciones output	Bad Siphon Controlado
$P_{14}$	4	$\{T_3, T_8\}$	$\{T_1, T_5\}$	$\{P_4, P_8, P_{10}, P_{11}, P_{12}, P_{13}\}$
$P_{15}$	2	$\{T_2, T_8\}$	$\{T_1, T_5\}$	$\{P_3, P_8, P_{10}, P_{11}\}$
$P_{16}$	3	$\{T_3, T_7\}$	$\{T_1, T_5\}$	$\{P_4, P_7, P_{11}, P_{12}, P_{13}\}$

CUADRO 4.8: Supervisores: RdP Fanti

<sup>11</sup>Figura adaptada del paper publicado por Fanti et al. [4]

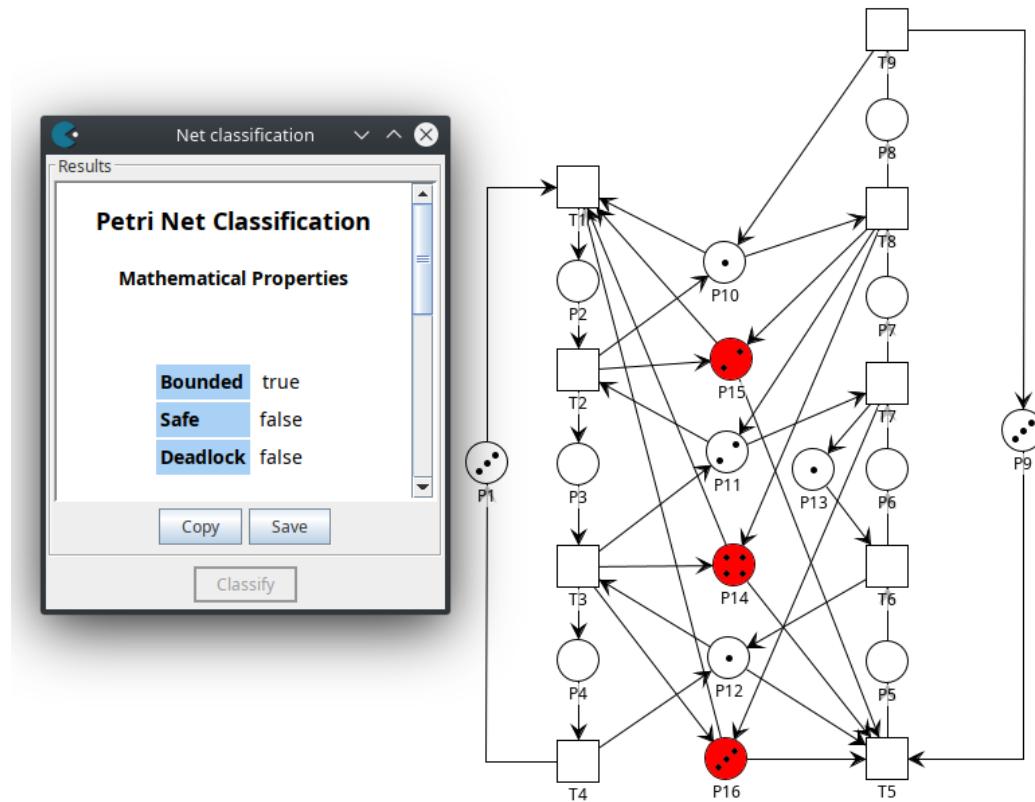


FIGURA 4.16: RdP Fanti controlada.

#### 4.1.6. Caso Hesuan Hu

La red representa un AMS donde se fabrican tres tipos de productos. El sistema esta compuesto por seis robots y cinco maquinas, cada uno de estos puede contener un solo producto.

##### 4.1.6.1. Características generales

- Los robots de la red están representados por las plazas  $\{P_{20}, P_{22}, P_{23}, P_{30}, P_{33}, P_{35}\}$ .
- Las máquinas de la red están representados por las plazas  $\{P_8, P_9, P_{21}, P_{31}, P_{32}\}$ .
- El primer producto comienza con la ejecución de la  $T_1$  y finaliza con la  $T_4$ .
- El segundo producto comienza con la ejecución de la  $T_9$  y finaliza con la  $T_{14}$ .
- El tercer producto comienza con la ejecución de la  $T_{19}$  y finaliza con la  $T_{24}$ .

#### 4.1.6.2. Análisis estructural

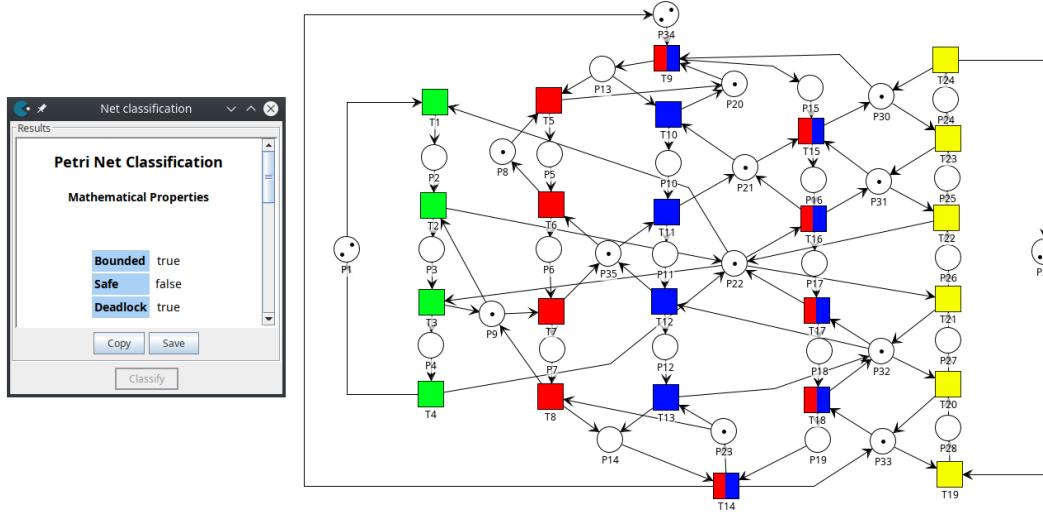


FIGURA 4.17: RdP Hesuan Hu<sup>12</sup> y sus T-invariantes.

En la Figura 4.17<sup>13</sup>, se presentan los T-invariantes en diferentes colores.

##### 4.1.6.2.1. Control de la red

Al agregar los supervisores en la red ésta aún presentaba deadlock, lo que se hizo fue ejecutar la parte 3 del algoritmo para resolver tanto el problema de conflicto y de t\_idle, logrando de esta manera el control de la red.

Supervisor	Marcado	Transiciones input	Transiciones output	Bad Siphon Controlado
$P_{36}$	2	$\{T_3, T_{16}, T_{22}\}$	$\{T_1, T_9, T_{19}\}$	$\{P_4, P_7, P_9, P_{17}, P_{22}, P_{25}, P_{31}\}$
$P_{37}$	1	$\{T_{17}, T_{21}\}$	$\{T_9, T_{19}\}$	$\{P_2, P_4, P_{12}, P_{18}, P_{22}, P_{26}, P_{32}\}$
$P_{38}$	3	$\{T_3, T_8, T_{13}, T_{21}\}$	$\{T_1, T_9, T_{19}\}$	$\{P_4, P_7, P_9, P_{12}, P_{14}, P_{22}, P_{26}, P_{32}, P_{33}\}$
$P_{39}$	1	$\{T_3\}$	$\{T_1\}$	$\{P_4, P_7, P_9, P_{17}, P_{22}, P_{26}\}$
$P_{40}$	1	$\{T_{16}, T_{22}\}$	$\{T_9, T_{19}\}$	$\{P_2, P_4, P_{17}, P_{22}, P_{25}, P_{31}\}$

CUADRO 4.9: Supervisores: RdP Hesuan Hu

<sup>12</sup>Figura adaptada del paper publicado por Hesuan Hu et al. [6]

<sup>13</sup>En esta RdP fue modificado el peso de los arcos reduciéndolos a uno para adaptarlas al tipo de red que admite el algoritmo.

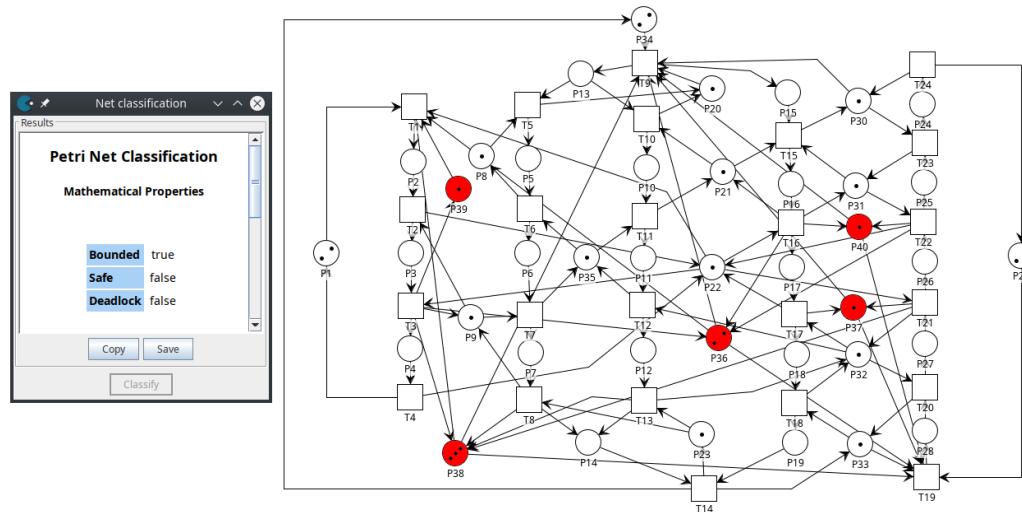


FIGURA 4.18: RdP Hesuan Hu controlada.



## 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 detectó la necesidad de extraer los T-invariantes de la red original, dado que estos debían preservarse a lo largo del análisis ya que con la incorporación de un nuevo supervisor la estructura de la red se ve afectada pudiendo perder los mismos, alterando el análisis y el objetivo es conservar el comportamiento de la red, es decir, que mantenga sus T-invariantes (que los procesos se sigan ejecutando de la misma manera) pero de una forma controlada y manteniendo la concurrencia de los procesos.

Esta modificación estructural que se presenta en la red es producto del nuevo supervisor, ya que al colocarlo no se tiene en cuenta si la red presenta conflictos o si las transiciones idle que le extraen el token son realmente necesarias. Para el primero se verificó que si la red presentaba conflictos antes de finalizar ese camino el token debía regresar el supervisor. Para el segundo se verificó que la transición idle le quitara token a los supervisores que realmente afectan al proceso que esta transición iba a desencadenar, de no ser así ese arco no debiera existir.

Por otra parte, se logró establecer una retroalimentación entre nuestro algoritmo y el software Petrinator, como ya se mencionó con anterioridad, el cual permite colocar de manera automática el nuevo supervisor y de ser necesario la actualización de sus respectivos arcos.

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 en su totalidad para todas las redes de este tipo.

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.
- Posibilidad de realizar un análisis para la determinación de un criterio de elección óptimo del orden de supervisores a colocar para el control de las diferentes redes permitiendo el mayor paralelismo a la hora de ejecutar las mismas.
- Posibilidad de integrar este proyecto junto con el que se esta desarrollando en el LAC para poder trabajar con redes de Petri cuyo grafo de alcanzabilidad supera las limitaciones de un ordenador, pudiendo ejecutarlas sin impedimentos haciendo uso de un hardware específico.
- 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
```

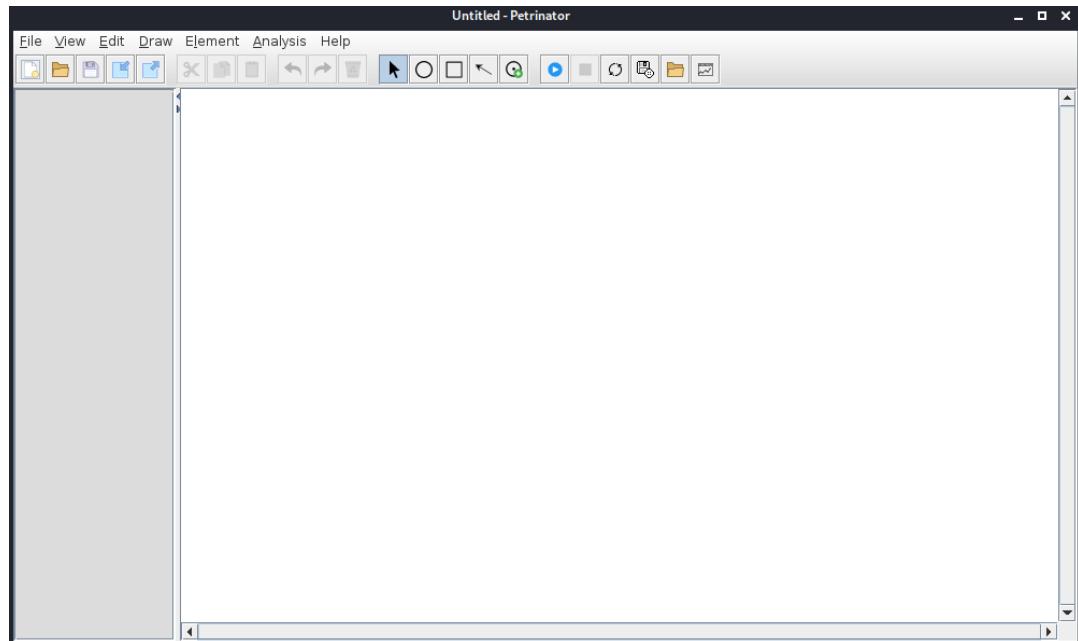


FIGURA A.1: Software Petrinator .

- **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.

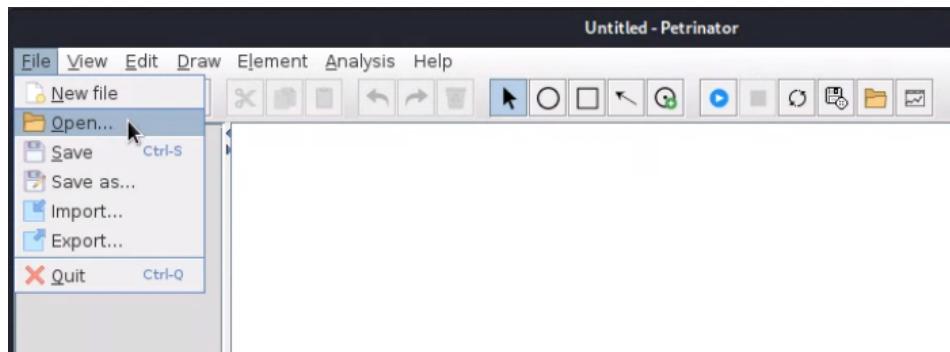


FIGURA A.2: Abrir un archivo .

Seleccionar el archivo de extensión .pflow a cargar

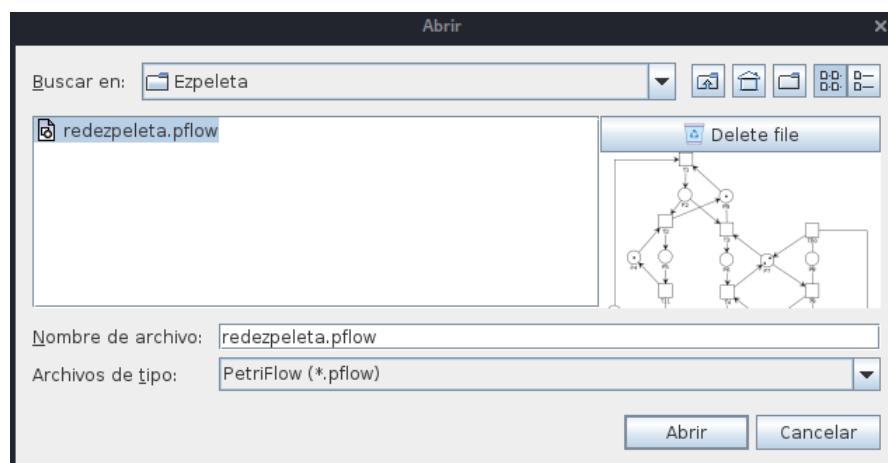


FIGURA A.3: Ventana de selección de archivo .pflow .

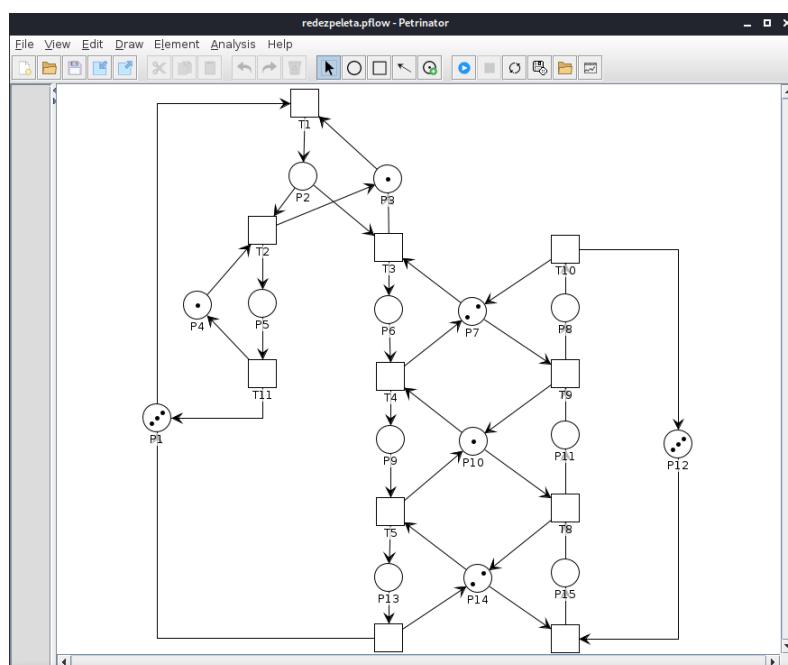


FIGURA A.4: Red cargada .

- **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.

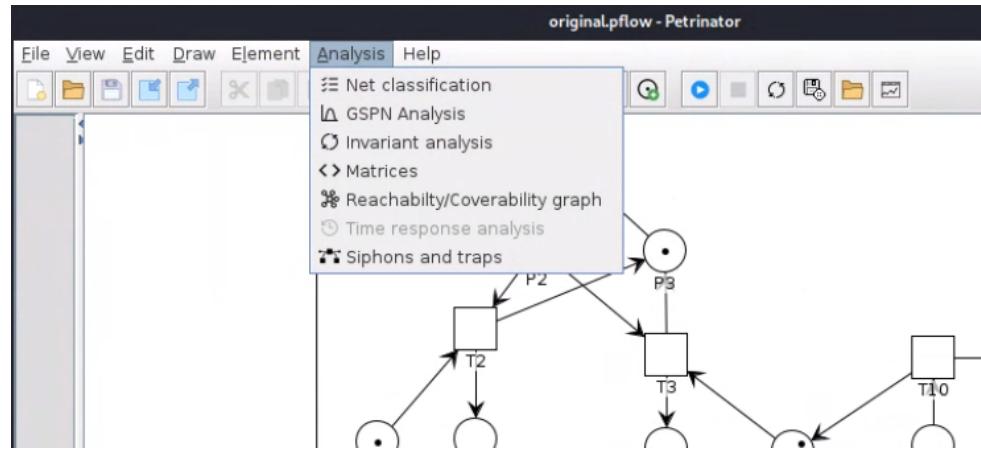


FIGURA A.5: Submenú de análisis.

- **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.

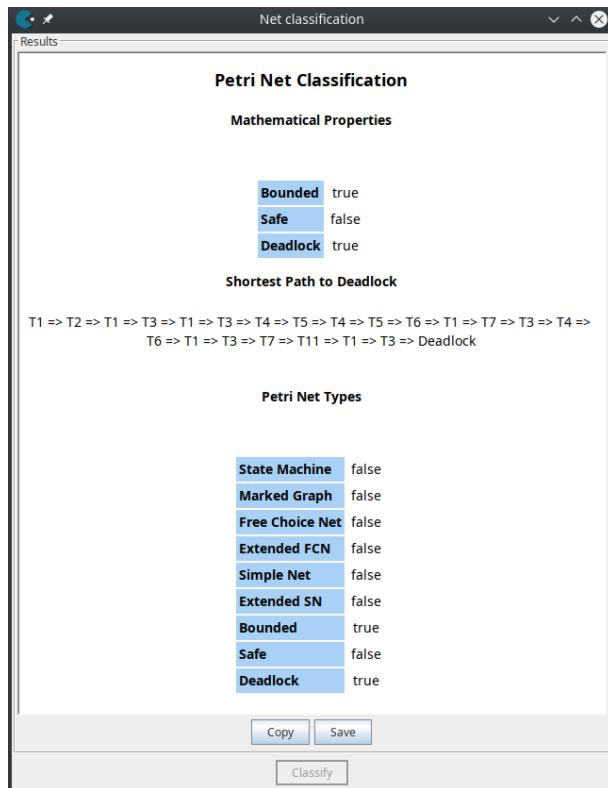


FIGURA A.6: Clasificación y propiedades de la red .

- **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*.

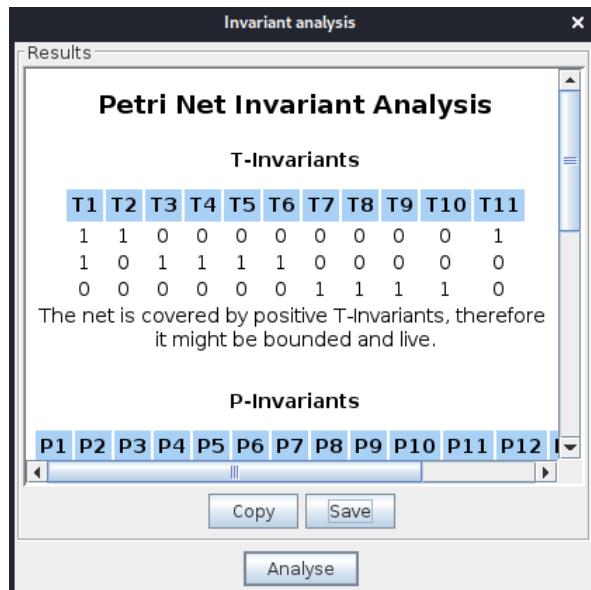
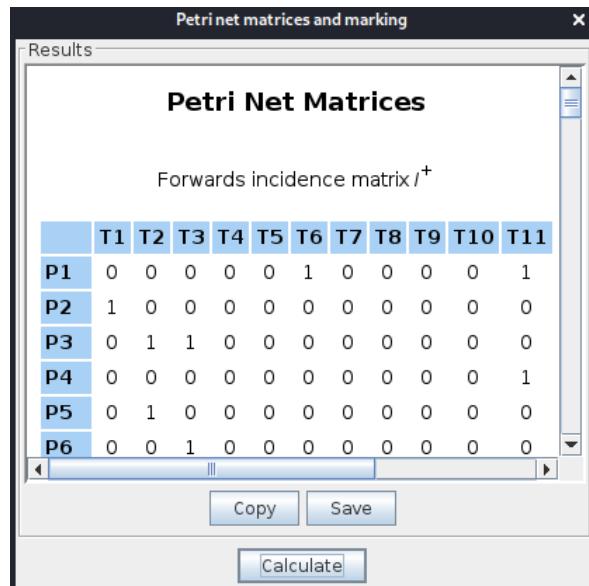


FIGURA A.7: Exportar invariantes.

- **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*.



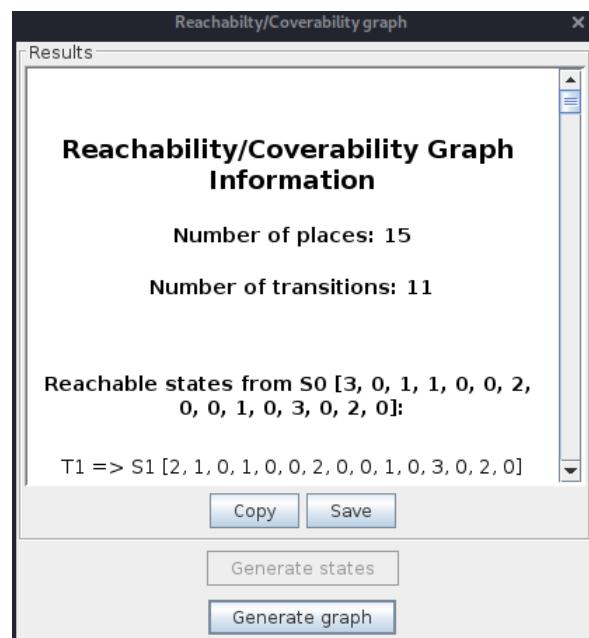
The screenshot shows a software window titled "Petri net matrices and marking". Inside, there's a section titled "Results" with a sub-section titled "Petri Net Matrices". Below this, it says "Forwards incidence matrix  $I^+$ ". A table is displayed with rows labeled P1 through P6 and columns labeled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

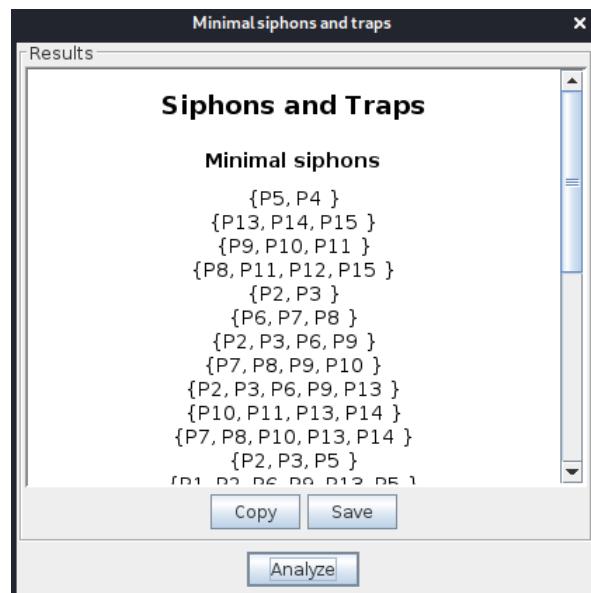


FIGURA A.10: Exportar sifones y trampas.

## Apéndice B

# Ejecución del algoritmo completa y detallada

En este apéndice, se detalla el procedimiento de ejecución de la última versión del algoritmo (sección 3.4) para un caso específico.

### B.1. Ejecución del caso POPN

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

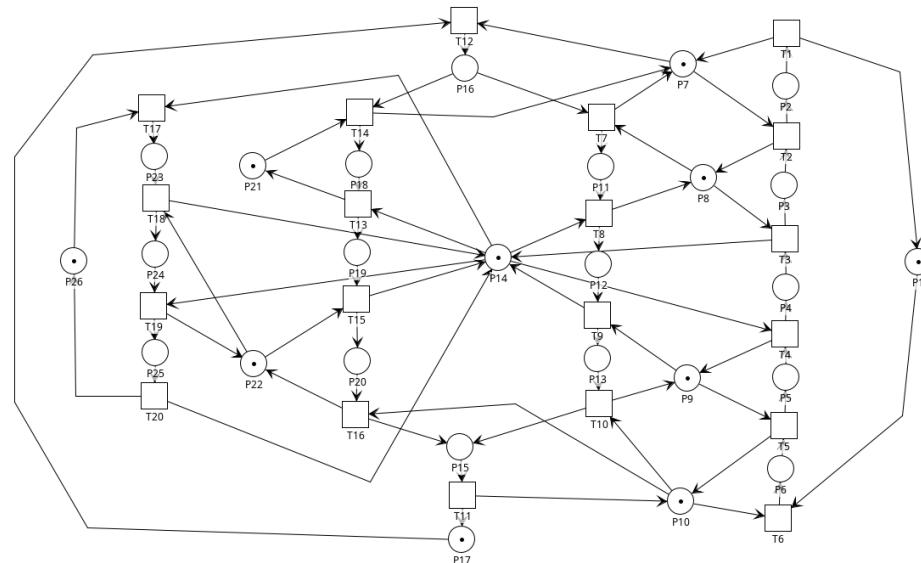


FIGURA B.1: RdP POPN<sup>1</sup>.

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.
- Matrices.
- Grafo de alcanzabilidad.

<sup>1</sup>Figura adaptada del libro *System Modeling and Control with Resource-Oriented Petri Nets* [34].

- Sifones y trampas.

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

```
$ python3 algoritmo.py
```

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

- go.html : Grafo de alcanzabilidad.
- mo.html : Matrices
- so.html : Sifones y trampas.
- io.html : Análisis de invariantes.

Al ser la primera vez que se ingresan los archivos se lleva a cabo la elección de la opción 1: *Primer análisis de la red*. El cual solicita ingresar el nombre de la red con la extensión *.pflow* a la que se le colocará el supervisor correspondiente. Una vez procesada la información y llevado a cabo el análisis, el algoritmo arroja los siguientes resultados:

```
matiasnavarro@kali:~/Facultad/Tesis/Banco_de_pruebas/Python$ python3 tesis.py
Algoritmo para la solucion de deadlock para redes de petri tipo S3PR
_____
Path del archivo de Estados (html): go.html
Path del archivo de Matrices I(html): mo.html
Path del archivo de Sifones(html): so.html
Path del archivo de Invariantes (html): io.html

Ingrrese:
1 - Primer analisis de la red
2 - Analisis de red con supervisores
3 - Red con supervisores, tratamiento de conflicto y t_idle

Opcion: 1

Ingrrese el nombre de la red(.pflow): POPN.pflow

Cantidad de estados con deadlock: 5
Cantidad de sifones vacios: 5

id= 0
Sifon a controlar: 2
Plazas del Sifon: ['P3', 'P8', 'P12', 'P14', 'P20', 'P22', 'P25']
Marcado del supervisor 2
Transicion output: 6
Transicion output: 12
Transicion output: 17
Transicion input: 3
Transicion input: 8
Transicion input: 15
Transicion input: 19
```

FIGURA B.2: Primer iteración: ejecución del primer análisis de la red.

```

id= 1
Sifon a controlar: 43
Plazas del Sifon: ['P3', 'P8', 'P12', 'P14', 'P19', 'P23', 'P25']
Marcado del supervisor 1
Transicion output: 6
Transicion output: 12
Transicion output: 17
Transicion input: 3
Transicion input: 8

id= 2
Sifon a controlar: 16
Plazas del Sifon: ['P4', 'P9', 'P13', 'P14', 'P20', 'P22', 'P25']
Marcado del supervisor 2
Transicion output: 6
Transicion output: 12
Transicion output: 17
Transicion input: 4
Transicion input: 9
Transicion input: 15
Transicion input: 19

id= 3
Sifon a controlar: 22
Plazas del Sifon: ['P4', 'P9', 'P13', 'P14', 'P19', 'P23', 'P25']
Marcado del supervisor 1
Transicion output: 6
Transicion output: 12
Transicion output: 17
Transicion input: 4
Transicion input: 9

```

FIGURA B.3: Lista de supervisores.

```

id= 4
Sifon a controlar: 3
Plazas del Sifon: ['P4', 'P12', 'P14', 'P20', 'P22', 'P25']
Marcado del supervisor 1
Transicion output: 6
Transicion output: 12
Transicion output: 17
Transicion input: 15
Transicion input: 19

¿Agregar supervisor?(S/N) S
AGREGA EL ID: 4

Ingrese:
1 - Deadlock = true - Volver a ejecutar el Algoritmo
0 - Deadlock = false - Finalizar ejecucion
Opcion: 1

```

FIGURA B.4: Elección del supervisor.

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.

Una vez seleccionado el supervisor a colocar, el propio algoritmo se encarga de agregar la plaza con su marcado y arcos correspondientes al mismo sobre el archivo *.pflow* especificado con anterioridad.

Luego se vuelve al Petrinator y se selecciona la opción *Reload* como se muestra en la figura B.5.

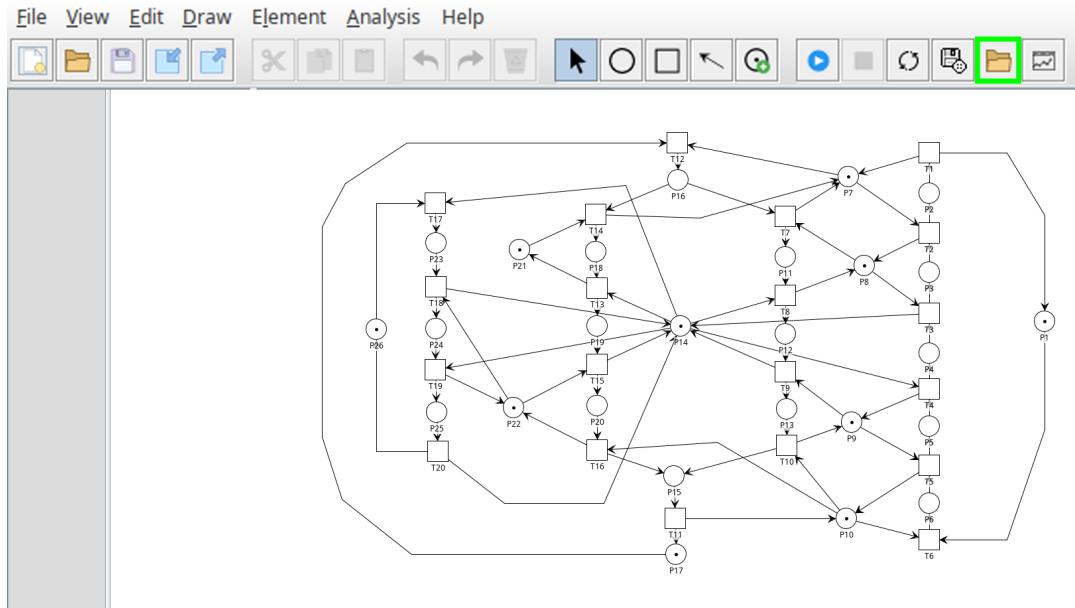


FIGURA B.5: Reload de la red.

Como se muestra en la siguiente figura B.6, la plaza resaltada  $P_{27}$  es el supervisor agregado.

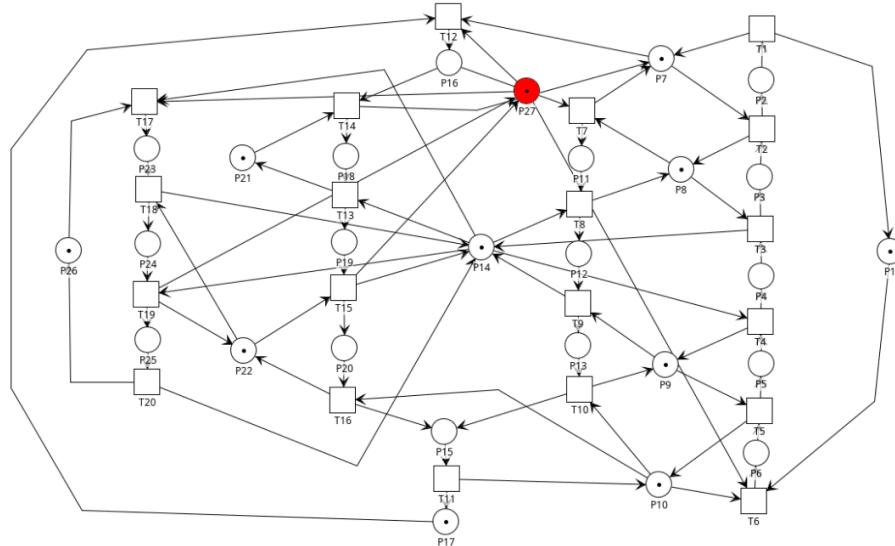


FIGURA B.6: Supervisor agregado.

Se debe verificar si el supervisor colocado soluciona el problema de deadlock realizando nuevamente el análisis de la red. En caso de no ser así, siguen existiendo el deadlock, se deberán extraer nuevamente los archivos correspondientes a la nueva red. Para seguir ejecutando el análisis se debe ingresar la opción 1 como se muestra al final de la figura B.4.

Nuevamente, se deben ingresar los archivos para realizar un nuevo análisis. En este caso la segunda opción *Análisis de red con supervisores*.

```

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

Ingrese:
1 - Primer analisis de la red
2 - Análisis de red con supervisores
3 - Red con supervisores, tratamiento de conflicto y t_idle

Opcion: 2

Cantidad de estados con deadlock: 1
Cantidad de sifones vacios: 9

id= 0
Sifon a controlar: 12
Plazas del Sifon: ['P16', 'P18', 'P19', 'P23', 'P24', 'P27']
Marcado del supervisor 0
Transicion output: 6
Transicion output: 12
Transicion output: 17

id= 1
Sifon a controlar: 21
Plazas del Sifon: ['P6', 'P16', 'P18', 'P19', 'P20', 'P23', 'P24', 'P27']
Marcado del supervisor 0
Transicion output: 6
Transicion output: 12
Transicion output: 17

```

FIGURA B.7: Segunda Iteración: análisis de la red con supervisores.

Como se puede observar en la figura B.7 la lista de supervisores sugeridos presentan marcado 0, por este motivo no se coloca ninguno de estos y se vuelve a ejecutar el algoritmo pero haciendo uso de la opción *Red con supervisores, tratamiento de conflicto y t\_idle*. Se deben eliminar/colocar los arcos indicados por el algoritmo como se muestra en la figura siguiente.

```

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

Ingrese:
1 - Primer analisis de la red
2 - Análisis de red con supervisores
3 - Red con supervisores, tratamiento de conflicto y t_idle

Opcion: 3

Eliminar arco desde P27 hasta T6
La transicion en conflicto T7 le tiene que devolver un token al supervisor P27

Se agrego un arco desde T7 hasta P27
Se elimino el arco desde P27 hasta T6

```

FIGURA B.8: Tercera Iteración: red con supervisores, tratamiento de conflicto y t\_idle.

El resultado de lo indicado con anterioridad se observa en la siguiente imagen.

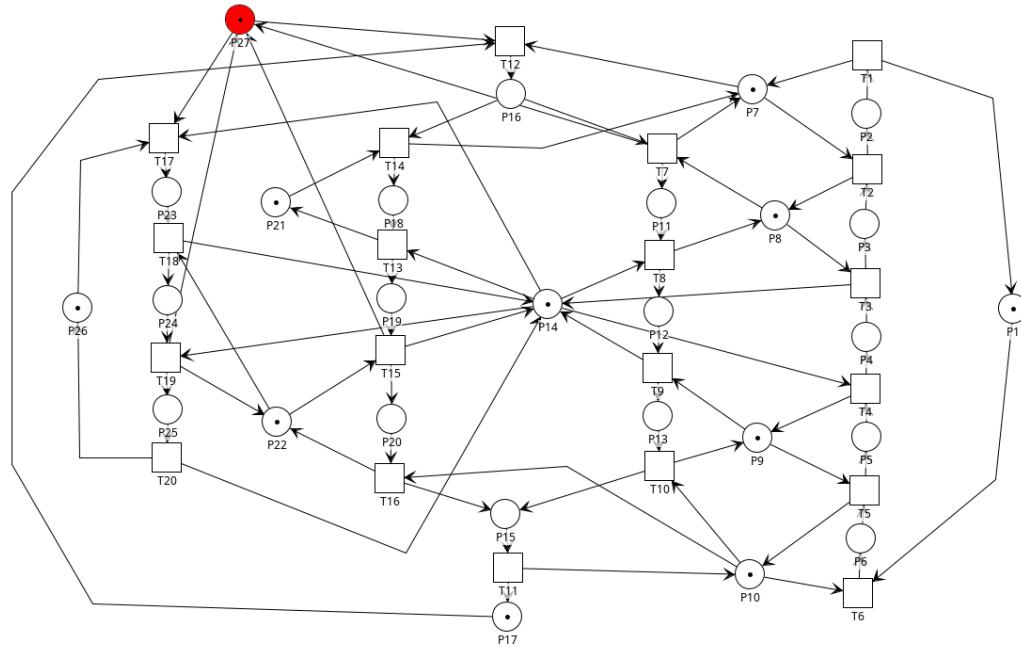


FIGURA B.9: Red resultante de agregar/eliminar arcos

Se verifica nuevamente si la red presenta deadlock y se extraen los archivos correspondientes. Para continuar con la ejecución iterativa hasta lograr el control total de la red, como se ejemplifica en las siguientes imágenes:

#### Algoritmo para la solución de deadlock para redes de petri tipo S3PR

```
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
```

Ingrese:  
 1 - Primer análisis de la red  
 2 - Análisis de red con supervisores  
 3 - Red con supervisores, tratamiento de conflicto y t\_idle

Opcion: 2

```
Cantidad de estados con deadlock: 4
Cantidad de sifones vacios: 4

id= 0
Sifon a controlar: 44
Plazas del Sifón: ['P3', 'P8', 'P12', 'P14', 'P19', 'P23', 'P25']
Marcado del supervisor 1
Transicion output: 6
Transicion output: 12
Transicion output: 17
Transicion input: 3
Transicion input: 8
```

FIGURA B.10: Cuarta iteración: análisis de la red con supervisores.

```

id= 3
Sifon a controlar: 17
Plazas del Sifon: ['P4', 'P9', 'P13', 'P14', 'P20', 'P22', 'P25']
Marcado del supervisor 2
Transicion output: 6
Transicion output: 12
Transicion output: 17
Transicion input: 4
Transicion input: 9
Transicion input: 15
Transicion input: 19

¿Agregar supervisor?(S/N) S
AGREGA EL ID: 0

Ingresar:
1 - Deadlock = true - Volver a ejecutar el Algoritmo
0 - Deadlock = false - Finalizar ejecucion
Opcion: 1

```

FIGURA B.11: Elección del supervisor.

En este caso, se selecciona el supervisor de id = 0. Como se puede observar en la siguiente figura.

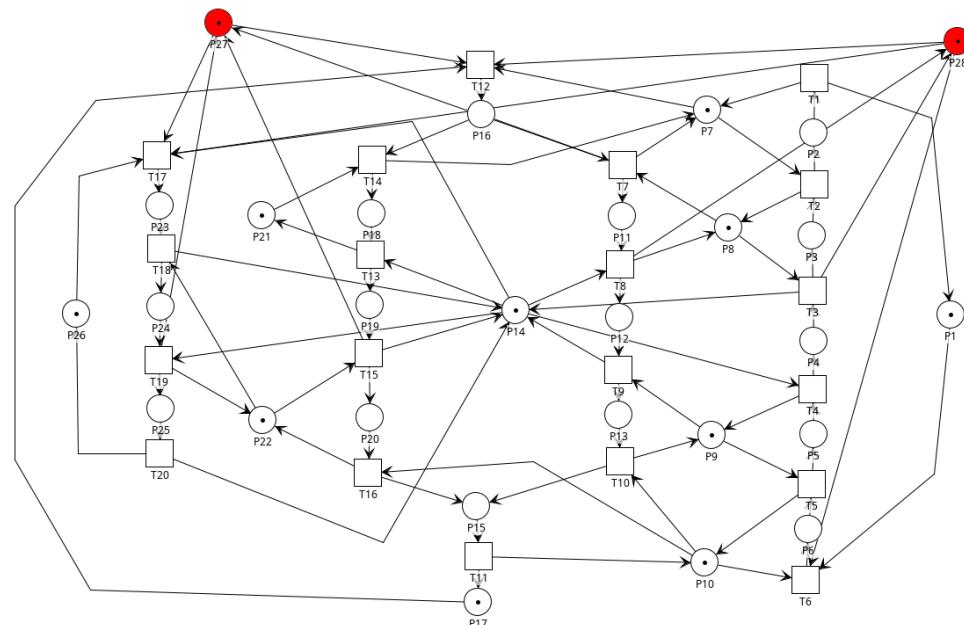


FIGURA B.12: Segundo supervisor agregado.

De manera iterativa<sup>2</sup> se colocan los demás supervisores, hasta que el análisis resulta con deadlock **false** o con supervisores de marcado 0. En el segundo caso, se debe hacer la ejecución de red con supervisores, tratamiento de conflicto y t\_idle.

<sup>2</sup>Para cada nueva iteración fue necesario verificar si la red presentaba deadlock y exportar los archivos correspondientes, como en las iteraciones anteriores.

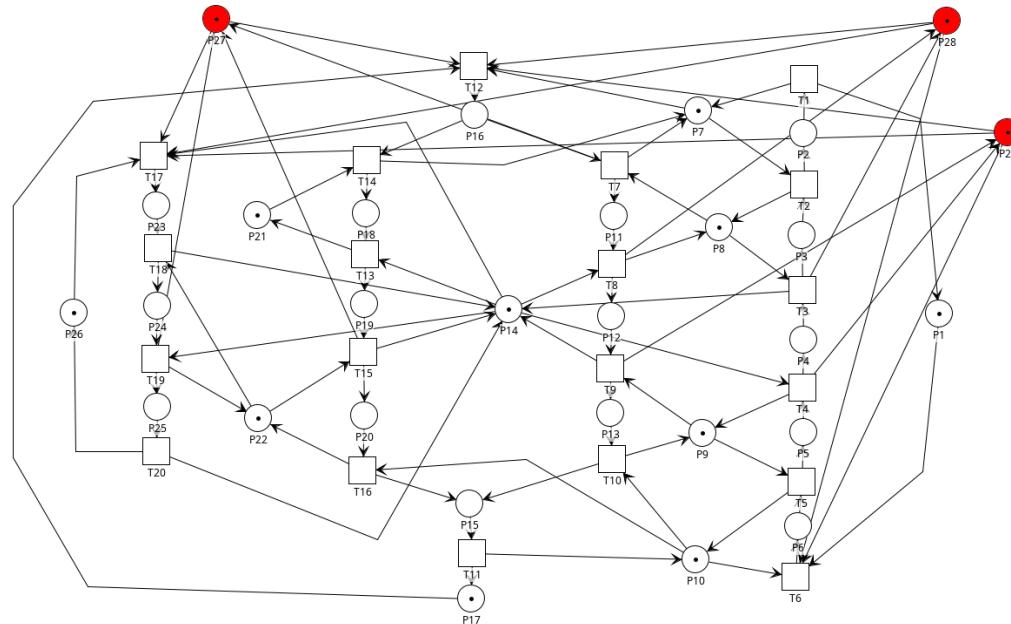


FIGURA B.13: Tercer supervisor agregado.

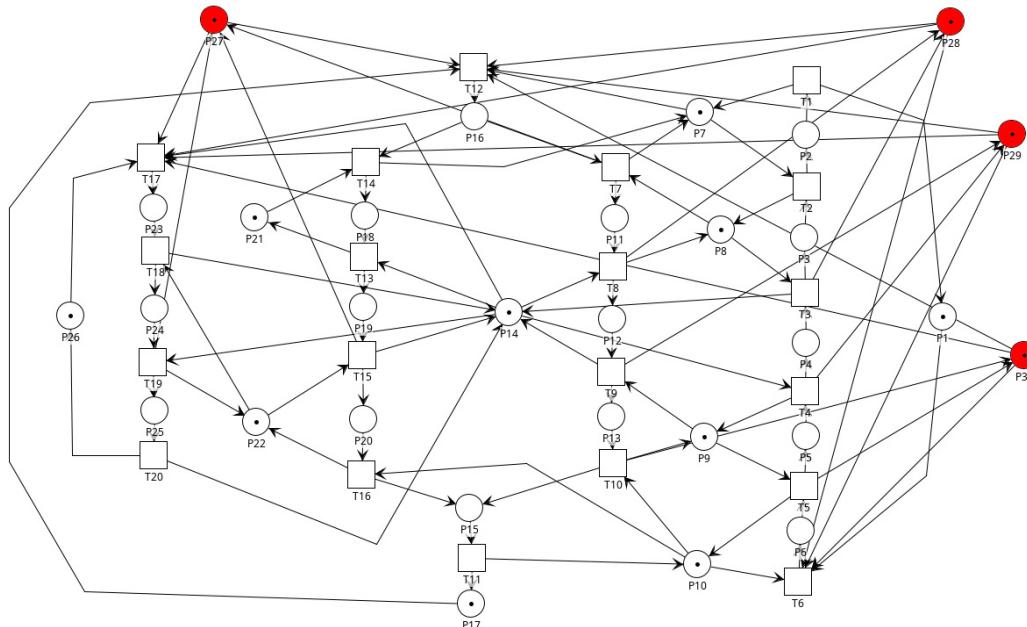


FIGURA B.14: Cuarto supervisor agregado.

```

Algoritmo para la solucion de deadlock para redes de petri tipo S3PR

Path del archivo de Estados (html): g5.html
Path del archivo de Matrices I(html): m5.html
Path del archivo de Sifones(html): s5.html
Path del archivo de Invariantes (html): i5.html

Ingrese:
1 - Primer analisis de la red
2 - Análisis de red con supervisores
3 - Red con supervisores, tratamiento de conflicto y t_idle

Opcion: 2

Cantidad de estados con deadlock: 1
Cantidad de sifones vacios: 17

id= 0
Sifon a controlar: 9
Plazas del Sifon: ['P6', 'P11', 'P12', 'P13', 'P16', 'P30']
Marcado del supervisor 0
Transicion output: 6
Transicion output: 12
Transicion output: 17

id= 1
Sifon a controlar: 10
Plazas del Sifon: ['P5', 'P6', 'P11', 'P12', 'P16', 'P29']
Marcado del supervisor 0
Transicion output: 6
Transicion output: 12
Transicion output: 17

```

FIGURA B.15: Última Iteración.

Por último, se ejecuta el algoritmo con la opción 3 como se observa en la figura B.16, indicando cuales arcos deben agregarse y cuales deben eliminarse.

```

Algoritmo para la solucion de deadlock para redes de petri tipo S3PR

Path del archivo de Estados (html): g5.html
Path del archivo de Matrices I(html): m5.html
Path del archivo de Sifones(html): s5.html
Path del archivo de Invariantes (html): i5.html

Ingrese:
1 - Primer analisis de la red
2 - Análisis de red con supervisores
3 - Red con supervisores, tratamiento de conflicto y t_idle

Opcion: 3

La transicion en conflicto T14 le tiene que devolver un token al supervisor P28
La transicion en conflicto T14 le tiene que devolver un token al supervisor P29
La transicion en conflicto T14 le tiene que devolver un token al supervisor P30
Eliminar arco desde P28 hasta T17
Eliminar arco desde P29 hasta T17
Eliminar arco desde P30 hasta T17

Se agrego un arco desde T14 hasta P28
Se agrego un arco desde T14 hasta P29
Se agrego un arco desde T14 hasta P30
Se elimino el arco desde P28 hasta T17
Se elimino el arco desde P29 hasta T17
Se elimino el arco desde P30 hasta T17

```

FIGURA B.16: Arcos a eliminar/agregar.

Se agregan/eliminan los arcos y se comprueba la clasificación de la red observando que el deadlock es false.

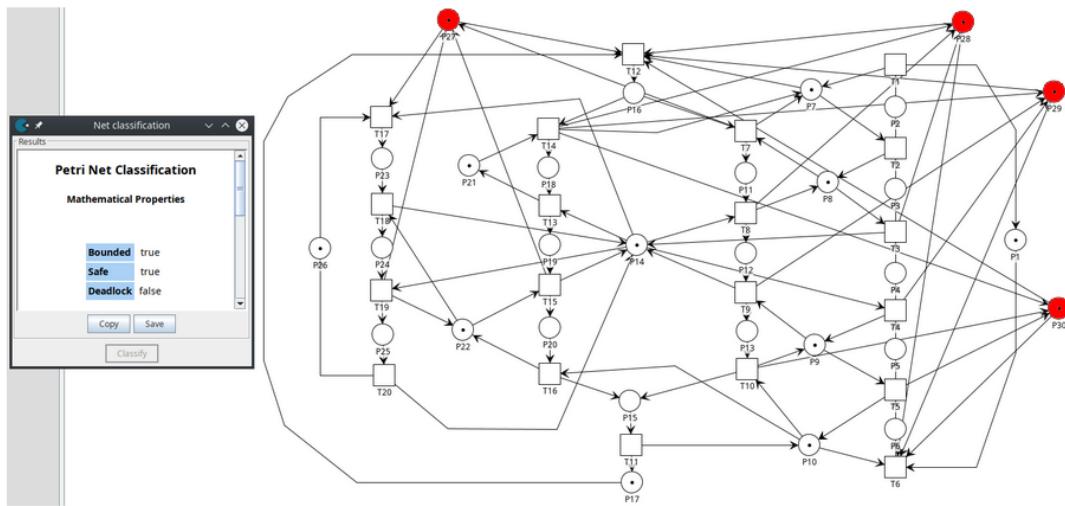


FIGURA B.17: RdP POPN controlada.

Una vez que se verifica que la red no presenta deadlock, para finalizar la ejecución del programa, se debe ingresar “0” en el menú de opciones que se muestra al final de la figura B.4.

# 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.
- [4] Maria Pia Fanti **and** Mengchu Zhou. «Deadlock Control Methods in Automated Manufacturing Systems». **in:** *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on* 34 (**february** 2004), **pages** 5–22. DOI: 10.1109/TSMCA.2003.820590.
- [5] Yifan Hou **and** Kamel Barkaoui. «Deadlock analysis and control based on Petri nets: A siphon approach review». **in:** *Advances in Mechanical Engineering* 9 (**may** 2017), **page** 168781401769354. DOI: 10.1177/1687814017693542.
- [6] H. Hu **and others**. «Deadlock-Free Control of Automated Manufacturing Systems With Flexible Routes and Assembly Operations Using Petri Nets». **in:** *IEEE Transactions on Industrial Informatics* 9.1 (2013), **pages** 109–121. DOI: 10.1109/TII.2012.2198661.
- [7] 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. URL: <https://doi.org/10.1007/s00170-006-0708-y>.
- [8] Yi-Sheng Huang, Yen-Liang Pan **and** Pin-June. Su. «Transition-Based Deadlock Detection and Recovery Policy for FMSs Using Graph Technique». **in:** *ACM Transactions on Embedded Computing Systems* 12 (2012). DOI: 10.1145/2406336.2406347.
- [9] 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.
- [10] Agustina Nahir Izquierdo Joffre, Matias Alejandro Navarro **and** Andres Salvatierra. «Determinación automática del control de una Red de Petri». Tesis de grado. UNC - Facultad de Ciencias Exactas Físicas y Naturales, **december** 2020.
- [11] J. M. Colom J. Ezpeleta **and** J. Martinez. «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.

- [12] 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.
- [13] 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.
- [14] Zhiwu Li, Naiqi Wu **and** Mengchu Zhou. «Deadlock Control of Automated Manufacturing Systems Based on Petri Nets—A Literature Review». in: *IEEE Transactions on Systems, Man, and Cybernetics - TSMC* 42 (july 2012), **pages** 437–462. DOI: 10.1109/TSMCC.2011.2160626.
- [15] 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>.
- [16] 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.
- [17] 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.
- [18] 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.
- [19] Abdul-Hussin M.H. «On Structural Conditions of S3PR based Siphon to Prevent Deadlocks of Manufacturing Systems». in: *International Journal of Simulation, Systems, Science. 17.33* (2016), **pages** 32.1–32.8. DOI: 10.5013/IJSSST.a.17.33.32.
- [20] Orlando Micolini **and others**. «Ecuación de estado generalizada para redes de Petri no autónomas y con distintos tipos de arcos». in: october 2016.
- [21] 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.
- [22] Sanchez Figueroa Fernando Palma Méndez José Tomas Garrido Carrera Maria del Carmen **and** Quesada Arenciaba Alexis. *Programación Concurrente*. Editorial Paraninfo, 2003.
- [23] 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>.
- [24] C. Wang S. Wang **and** M. Zhou. «Optimal siphon-based deadlock prevention policy for a class of Petri nets in automation.» in: *IEEE International Conference on Systems, Man, and Cybernetics, Anchorage, AK* (2011), **pages** 826–831. DOI: 10.1109/ICSMC.2011.6083755.

- [25] William Stallings. *Sistemas operativos: aspectos internos y principios de diseño*. 5ta. Madrid: Pearson Educación, S.A., 2005. ISBN: 978-84-205-5796-0.
- [26] A. Timotei **and** J. Colom. «A New Approach to Prevent Deadlock in S3PR Nets with Unreplicable Resources». in: *Proceedings of the 2nd International Conference on Operations Research and Enterprise Systems* 1 (2013), pages 252–257. DOI: 10.5220/0004275702520257.
- [27] 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.
- [28] 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.
- [29] 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.
- [30] 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.
- [31] 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.
- [32] 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.
- [33] C.-F. Zhong, Li **and** Z.-W. «Design of liveness-enforcing supervisors via transforming plant petri net models of FMS». in: *Asian Journal of Control* 12 (2010), pages 240–252. DOI: 10.1002/asjc.182.
- [34] MengChu Zhou **and** Naiqi Wu. *System Modeling and Control with Resource-Oriented Petri Nets*. 1st. USA: CRC Press, Inc., 2009. ISBN: 1439808848.