CENTRO DE INVESTIGACIÓN Y DE ESTUDIOS AVANZADOS DEL INSTITUTO POLITÉCNICO NACIONAL

DEPARTAMENTO DE COMPUTACIÓN

VERIFICACIÓN Y ANÁLISIS DE BASE DE REGLAS ACTIVAS

Tesis que presenta la:

M. en C. Lorena Chavarría Báez

Para obtener el grado de Doctora en Ciencias en la especialidad de Ingeniería Eléctrica

Director de la tesis:

Dra. Xiaoou Li

México, D. F. Diciembre de 2008

Índice general

Agradecimientos

A Dios y a la Virgen de Guadalupe porque siempre me han guiado y ayudado a llegar hasta este punto.

A mis padres, José Luis Chavarría Silva y Lourdes Báez Rodríguez porque siempre me han apoyado y animado a salir adelante, a superarme día con día y a ser una mejor persona. Porque siempre he recibido de ellos ejemplos de superación y calidad humana.

A mis hermanos, Ivonne y Juan Franciso, por su comprensión, apoyo, ayuda y por todos los momentos que hemos pasado juntos.

A mi asesora, la Dra. Xiaoou Li, por toda la paciencia que me tuvo durante el desarrollo de la tesis, por las correcciones en mi investigación y porque siempre ha sido muy amable al atenderme.

A los revisores de mi trabajo, los Doctores Pedro Mejía Álvarez, José Guadalupe Rodríguez García, Ernesto López Mellado y Joselito Medina Marín por sus comentarios ya que estos contribuyeron a mejorar el contenido de la tesis.

A todos y cada uno de mis amigos, por su amistad, cariño, paciencia y comprensión para entender que algunas veces no podía estar con ellos.

A mi amigo Amilcar por sus comentarios y todo el apoyo que me brindó.

A Erick por su ayuda incondicional, comprensión y cariño.

A todos los Doctores del Departamento de Computación por todo lo que aprendí de ellos.

A Sofi porque sin ella no habría podido hacer ningún trámite administrativo pero, además, por su preocupación y cariño hacia mí. A Feli y Flor por hacer tan bien su labor.

Al CINVESTAV y a CONACYT por apoyarme con la beca de doctorado.

Resumen

Los sistemas basados en reglas activas (sistemas activos) ejecutan acciones de manera automática ante la ocurrencia de ciertos eventos, los cuales pueden ser generados dentro o fuera del sistema. La parte medular de tales sistemas la constituye la base de reglas activas, la cual representa el conocimiento que se tiene acerca de un dominio particular. Una regla activa tiene la forma: **ON** *evento* **IF** *condición* **THEN** *acción*, la cual indica la tarea que se debe llevar a cabo cuando ha ocurrido un evento y ciertas condiciones se han cumplido. Este tipo de reglas también se denominan reglas ECA (Evento - Condición - Acción).

Desarrollar sistemas activos no es una tarea sencilla debido a factores como los siguientes: primero, no es fácil decidir las partes del sistema que pueden implementarse mediante reglas activas; segundo, se tiene que analizar el costo que se debe pagar por la introducción de las reglas en la aplicación; tercero, la falta de herramientas de verificación y análisis que ayuden a identificar errores en la base de reglas y examinar si esta realiza adecuadamente las tareas para las que fue diseñada.

En la literatura se ha reportado un vasto trabajo relacionado con la verificación de errores en sistemas basados en reglas de producción, las cuales son un caso particular de las reglas activas ya que sólo contienen la parte de la condición y la acción, no así para sistemas activos. Asimismo, se han desarrollado métodos de análisis para las propiedades de terminación y confluencia, algunos de los cuales no determinan con exactitud su cumplimiento. La propiedad de terminación garantiza que el proceso de ejecución de una base de reglas se realiza en un número finito de pasos. La propiedad de confluencia determina si el orden de ejecución de reglas disparadas simultáneamente en un mismo estado del sistema influye en el resultado final que se obtenga.

En este trabajo de tesis se aborda la verificación de errores y el análisis de las propiedades de terminación y confluencia de reglas activas. Las principales aportaciones obtenidas se describen a

continuación:

1. Definiciones de errores de redundancia, inconsistencia, incompletitud y circularidad en el contexto de reglas activas.

   Hasta el momento, no se han reportado en la literatura relacionada con reglas activas definiciones para los errores mencionados, los conceptos originales sólo consideran reglas de producción. Dado que las reglas activas están formadas por un evento, una condición y una acción, es importante contar con definiciones de errores que consideren todas estas partes ya que el evento hace que estos puedan aparecer en formas más complicadas.

   Tomando en cuenta esta situación, se analizaron los tipos de errores que aparecen en bases de reglas de producción y, posteriormente, se redefinieron para adaptarlos al contexto de reglas activas. Como resultado se propusieron definiciones para los errores de redundancia, inconsistencia, incompletitud y circularidad que consideran todos las partes componentes de las reglas activas.

2. Definiciones de errores potenciales.

   En algunas ocasiones, los errores ocurren sólo hasta que se satisfacen ciertas condiciones. En ese momento, pueden llegar a causar un mal funcionamiento de la base de reglas y, por lo tanto, deben prevenirse. Este tipo de errores los denominamos errores potenciales. Entre ellos están las reglas en conflicto potencial, las cuales pueden generar inconsistencia sólo cuando el resultado de la evaluación de sus condiciones es verdadero.

   Al conocer los errores potenciales que existen en la base de reglas se pueden tomar medidas para manejarlos cuando sucedan o, incluso, rediseñar el conjunto de reglas para eliminarlos.

3. Método de verificación de errores basado en la CCPN.

   A partir del conjunto de definiciones se desarrolló un método de detección de errores basado en una extensión de las redes de Petri llamada Red de Petri Coloreada Condicional (CCPN, por sus siglas en inglés *Conditional Colored Petri Net*). El modelo de CCPN representa con claridad cada elemento de la regla así como su interacción. El método consta de dos etapas: primero, usando la CCPN se identificaron patrones que caracterizan a las reglas que pueden contener errores y se aprovechó la matriz de conexión de la CCPN para detectarlos

automáticamente. Segundo, se extrajeron las reglas identificadas por los patrones y se analizó su interacción para determinar con exactitud la ocurrencia de algún tipo de error.

4. Método de análisis de las propiedades de confluencia y terminación basado en la CCPN.

   El método de análisis de las propiedades de confluencia y terminación desarrollado está basado en la CCPN y consta de dos partes. En la primera de ellas, usando la CCPN, se identificaron los patrones que caracterizan reglas que pueden no terminar y reglas que pueden no ser confluentes. En la segunda parte, se desarrollaron algoritmos para analizar la interacción de las reglas identificadas a través de los patrones y así concluir acerca del cumplimiento de las propiedades.

   La ventaja que tienen estos enfoques de verificación de errores y análisis de propiedades de terminación y confluencia es que no necesitan examinar todas las reglas para formular conclusiones. El identificar patrones en la CCPN permite centrarse sólo en aquellas reglas que puedan ser incorrectas.

5. Desarollo de la herramienta ECAPNVerifier.

   ECAPNVerifier es un sistema que permite verificar una base de reglas de forma automática. Esta herramienta se integró a ECAPNSim, el cual es un sistema que genera la CCPN de una base de reglas así como su correspondiente matriz de conexión.

Abstract

Active rule-based systems (active systems) respond automatically to events that are taking place inside or outside the system itself. The most important element of active systems is the rule base which represents knowledge about a particular area. An active rule of the rule base is written as follows: **ON** *event* **IF** *condition* **THEN** *action*. It specifies the task which has to be performed when an important event happens and some conditions are met. This kind of rules are also called ECA (Event - Condition - Action) rules.

Developing a rule base is commonly perceived as a difficult task for a variety of reasons, first, it is not obvious which parts of an application should be supported using active mechanisms, second, what performance penalty is likely to result from the use of rules, and third, the verification and analysis tools which detect errors and determine if the rule base properly fulfills its goal may be minimal.

Many works on error verification using production rules, which are a special case of active rules since they only have condition and action parts, have been reported in specialized literature. However, to the best of our knowledge, there is no enough work about error verification usign active rules. Also, many works about confluence and termination properties analysis have been published. But, some of them cannot draw conclusion accurately. Confluence property guarantees that the final result of rules triggered at the same time will be the same independently of their execution order. Termination property guarantees that rule processing will be done in a finite number of steps.

This thesis tackles error verification and confluence and termination analysis properties in active rule base. In the following we describe our main contributions:

1. Definitions for redundancy, inconsistency, incompleteness and circularity errors using active rules.

Up to date, there isn't any report in literature about active rules related with definitions of mentioned errors, original conceptions only take into account production rules. Since active rules consist of an event, a condition and an action, it is important to have error definitions which consider all these parts because of event makes errors to appear in more complicated ways.

Considering this situation, errors which appear in production rule base were analized and, later, they were redefined to adapt them to active rule context.

As result we obtained definitions for redundancy, inconsistency, incompleteness and circularity errors which take into account all the rule's parts.

2. Definitions of potential errors.

   Potential errors are those which can cause a bad performance of the rule base at a specific moment and, therefore, they have to be prevented. Among potential errors are potential conflicting rules which can cause inconsistency only when their conditions may be true at the same time.

   If potential errors are discovered it is possible to formulate ways to deal with them when they happen, or even the rule base can be designed again to take away potential errors.

3. CCPN-based method to verify errors in an active rule base.

   From above definitions we developed an error verification method, which is based on a Petri net extension called Conditional Colored Petri Net (CCPN). CCPN model represents each rule element as well as their interaction. Our method consists of two parts: first, using CCPN we identified patterns which characterize rules with errors and took advantage of its incidence matrix representation to automatically detect them. Second, we extract the rules represented identified by patterns and we analyzed their interaction to draw conclusion about errors.

4. CCPN-based method to analyze confluence and termination properties.

   We developed a method for analyzing above properties which is based on CCPN. Our method is performed in two stages: first, we identify those CCPN structures which depict non-confluence and non-termination problems. Second, we analyzed rule interaction of rules represented by above structures and we conclude about confluence and termination.

The great advantage of our verification and analysis method is that we can draw conclusion about errors and fulfillment of confluence and termination properties with no need to test all the rules pairs in the rule base. Using CCPN we can focus our test on the rules which may be wrong.

5. ECAPNVerifier development.

   ECAPNVerifier is a software system which verifies an active rule base automatically. It was integrated into a more robust tool called ECAPNSim which can generate a CCPN and its corresponding incidence matrix form an active rule base.

Capítulo 1

Introducción

Los sistemas activos reaccionan ante eventos, que pueden suceder dentro o fuera del sistema, ejecutando acciones automáticamente. Por ejemplo, consideremos una aplicación que está supervisando el cambio en la intensidad de la luz natural para entonces ajustar la intensidad de la luz dentro de una habitación. Este procedimiento necesita ser ejecutado automáticamente por las siguientes razones: 1) brindar confort al usuario, y 2) ahorrar energía eléctrica. Aplicaciones como la descrita pueden desarrollarse expresando el conocimiento que se tiene de la aplicación en forma de reglas, es decir, la situación previa puede escribirse como "*cuando* cambie la intensidad de la luz natural, *si* la intensidad de la luz está por debajo de un cierto umbral, *entonces* incrementar la intensidad de la luz de la habitación", e incorporando tales reglas al desarrollo del sistema completo. Una regla de la forma anterior (*cuando* evento - *si* condición - *entonces* acción) se conoce como regla activa o regla Evento - Condición - Acción (ECA).

A primera vista, parece sencillo hacer que un sistema tenga un comportamiento reactivo a través de la introducción de reglas activas. Sin embargo, esta tarea se complica si consideramos aspectos como los siguientes: primero, qué partes de la aplicación pueden desarrollarse mediante reglas activas y cuál es el costo que se debe pagar por el empleo de las mismas; segundo, la expresividad del lenguaje de eventos, lo cual permite definir el número y tipo de eventos que pueden ser supervisados, y tercero, la evaluación que se hará de la condición y la acción con respecto al evento, es decir, cuánto tiempo después de que el evento ha sucedido se debe evaluar la condición y ejecutar la acción. Los aspectos anteriores reflejan la necesidad de metodologías de

análisis y diseño para desarrollar sistemas activos, y la necesidad de mecanismos que soporten la implementación de tales sistemas. Sin embargo, dado que el conjunto de reglas activas es la parte medular de un sistema activo y considerando la naturaleza, a veces crítica, de esos sistemas, es aún más importante contar con herramientas de verificación que determinen si una base de reglas contiene errores, y con herramientas de análisis que indiquen si el procesamiento de las reglas termina y es confluente. La propiedad de *terminación* garantiza que no existe un subconjunto de reglas que se está disparando entre sí continuamente formando un ciclo. La propiedad de *confluencia* determina si el orden de ejecución de reglas disparadas simultáneamente en un mismo estado del sistema influye en el resultado final que se obtenga [1].

Mucho se ha escrito sobre la verificación de errores estructurales (redundancia, inconsistencia, incompletitud y circularidad) de sistemas que emplean reglas de producción, las cuales son un caso particular de las reglas activas ya que sólo consisten de los elementos condición y acción, como medio de expresión de conocimiento [12], [13], [6], [14], [15], [16],[7], [17], [18], [19], [20], [11], sin embargo, casi nada se ha reportado sobre la verificación de reglas activas [5].

También se han desarrollado y reportado trabajos en la literatura acerca de métodos para determinar si una base de reglas termina y es confluente. Algunos de los métodos reportados están basados en el análisis de grafos [30], [31], [32], [33], [34], [35], [36]; algunos otros se basan en el álgebra relacional [37], algunos más tratan a las reglas activas como reglas deductivas [38] y otros abordan el problema por medio de las redes de Petri [41], [39], [40].

1.1. Motivación

El tema de aseguramiento de calidad en el software ha tomado importancia dado el crecimiento en tamaño y complejidad de los sistemas que actualmente se desarrollan. Los sistemas basados en conocimiento no están exentos de esta condición. De hecho, en aplicaciones como control de tráfico aéreo, en las que los sistemas basados en conocimiento participan, la calidad es vital para su uso.

Los errores debilitan la calidad de cualquier producto de software. En un sistema basado en conocimiento los errores se pueden introducir durante el diseño del sistema o durante la adquisición del conocimiento. Entre las principales causas de errores están las siguientes: 1) la falta de especificaciones o adherencia a ellas, 2) errores sintácticos y semánticos, y 3) la incorrecta representación del conocimiento. Para asegurar que un sistema basado en conocimiento operará durante un largo

tiempo, es necesario realizar una verificación formal de su base de reglas en donde se identifiquen los errores que pudieron (inadvertidamente) ser introducidos.

Además de la verificación de errores, existen algunas propiedades de las reglas que se deben examinar para comprobar su adecuado funcionamiento. Tales propiedades son: terminación y confluencia. Analizar la propiedad de terminación es importante dado que previene la ejecución infinita de un conjunto de reglas. El análisis de la propiedad de confluencia permite determinar si se obtendrá el mismo resultado final en el sistema sin importar el orden de ejecución de reglas disparadas simultáneamente.

Por estas razones es necesario desarrollar un conjunto de métodos de verificación de errores y análisis de las propiedades de terminación y confluencia de reglas que permitan obtener conclusiones acerca del diseño de la base de reglas.

1.2. Objetivos

El objetivo principal de este trabajo de tesis se enuncia a continuación:

"Desarrollar un conjunto de métodos para el análisis de reglas activas que permitan identificar los errores estructurales introducidos durante su etapa de diseño y analizar las propiedades de terminación y confluencia"

Entre los objetivos específicos tenemos los siguientes:

- Redefinir las nociones de errores de redundancia, inconsistencia, incompletitud y circularidad en el contexto de reglas activas.
- Desarrollar un método de verificación de errores en reglas activas.
- Desarrollar un método de análisis de propiedades de confluencia y terminación de reglas activas.
- Desarrollar un sistema de software que implemente los métodos de verificación de errores y análisis de propiedades de confluencia y terminación.

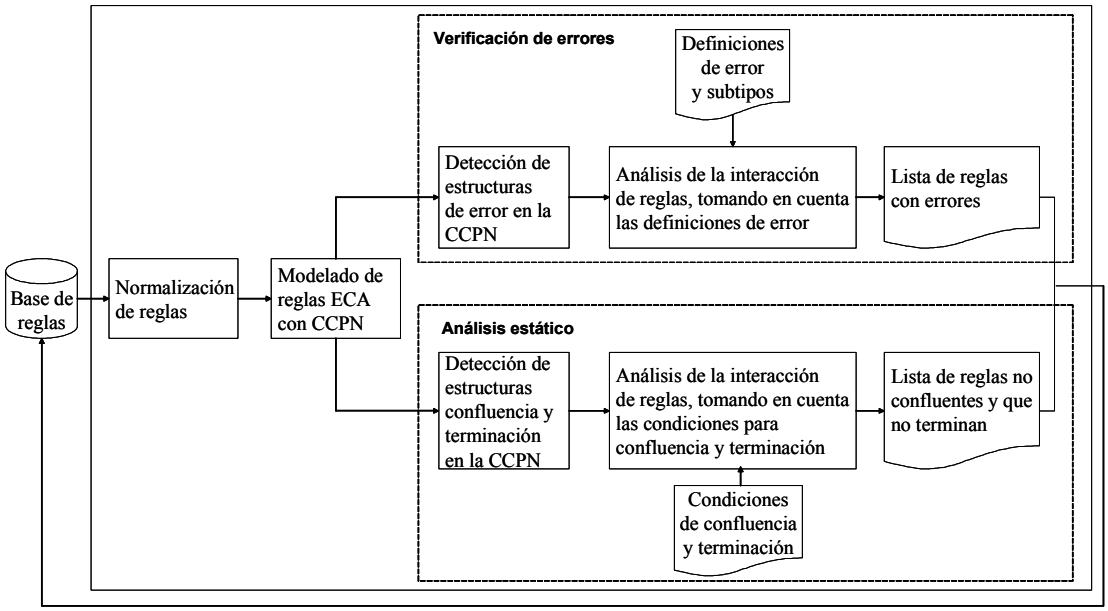


Figura 1.1: Planteamiento del problema

1.3. Planteamiento del problema

La figura 1.1 muestra el planteamiento de nuestro problema. En la primera fase, se detectan los errores que puedan estar presentes en la base de reglas. Para lograrlo, lo primero que se tiene que hacer es redefinir las nociones de redundancia, inconsistencia, incompletitud y circularidad. En estas definiciones se describen las condiciones para que sucedan los errores. Posteriormente, la base de reglas se normaliza y modela como una estructura de Red de Petri Coloreada Condicional (CCPN) para facilitar la detección de errores e identificar las reglas que pueden tener problemas de no-confluencia y no-terminación. La etapa de normalización se introduce para simplificar el análisis. La CCPN es una extensión de las redes de Petri que representa cada elemento de la regla independientemente y permite visualizar de forma gráfica la interacción entre las reglas. Además, cuenta con herramientas como la matriz de conexión que nos permiten analizar características de los sistemas modelados. Sobre el modelo de CCPN obtenido, se identifican las estructuras que representan errores y las que representan problemas de no-terminación y confluencia. Finalmente, se analiza la interacción de las reglas involucradas en estas estructuras y se obtienen conclusiones

acerca de los errores contenidos y las propiedades de confluencia y terminación.

La metodología que proponemos seguir consta de los siguientes pasos:

1. Redefinir los conceptos de error de redundancia, inconsistencia, incompletitud y circularidad.
2. Desarrollar métodos de análisis para identificar los errores en la base de reglas.
   - Modelar la base de reglas con la CCPN.
   - Detectar estructuras de error. Este paso regresará las reglas que pueden tener errores.
   - Analizar la interacción de las reglas obtenidas previamente para determinar con exactitud si contienen errores.
3. Desarrollar métodos de análisis de las propiedades de confluencia y terminación.
   - Modelar la base de reglas por medio de la CCPN.
   - Identificar las estructuras en la CCPN que indiquen posibles problemas de no-confluencia y no-terminación.
   - Desarrollar métodos que determinen si un conjunto de reglas es confluente. Estos métodos deben examinar con detalle cada elemento de la condición y la acción de una regla. Entre más exacta sea la información que se tome, con mayor precisión se puede determinar si un conjunto de reglas es confluente.
   - Desarrollar métodos para determinar si el procesamiento de una base de reglas termina. Estos métodos verifican si las reglas involucradas en un ciclo se pueden activar entre sí. Si esto sucede, entonces el procesamiento no termina. De lo contrario, se puede asegurar que, no importando que exista un ciclo, el procesamiento de estas reglas terminará.
4. Desarrollar un sistema de software que implemente los métodos de verificación de errores y análisis de propiedades de confluencia y terminación.
   - Implementar los métodos de verificación de errores.
   - Implementar los métodos de análisis de propiedades de confluencia y terminación.

- Integrar la implementación realizada con el sistema ECAPNSim ya que este genera la CCPN y su matriz de conexión de reglas activas.
- Realizar pruebas al sistema completo.

1.4. Descripción del trabajo

Las principales contribuciones de este trabajo son las siguientes:

1. Definiciones de errores de redundancia, inconsistencia, incompletitud y circularidad en el contexto de reglas ECA.
2. Definiciones de subtipos de error útiles para identificar situaciones problemáticas potenciales.
3. Método de verificación de errores basado en la CCPN.
4. Método de análisis de las propiedades de confluencia y terminación basado en la CCPN.
5. Sistema de software que verifica los errores y analiza las propiedades de confluencia y terminación de reglas activas.

El presente documento está estructurado de la siguiente manera: en el Capítulo 2 se presentan los conceptos básicos que intervienen en la construcción de sistemas basados en conocimiento, así como algunos trabajos reportados en la literatura para la verificación de errores en estos sistemas.

En el Capítulo 3 se presenta una introducción a las reglas activas. Se describe su modelo de ejecución así como los tipos de interacción que se establecen entre las reglas. También se detallan los trabajos reportados para verificar errores en reglas activas y determinar si su proceso de ejecución es confluente y termina. Finalmente, se describen algunas aplicaciones activas tales como la tecnología de bases de datos activas, casas inteligentes, aplicaciones financieras.

El modelo de CCPN usado para representar reglas ECA y su interacción se describe en el Capítulo 4. Para ilustrar el proceso de modelado se muestra un ejemplo.

Las definiciones de errores de redundancia, inconsistencia, incompletitud y circularidad desarrolladas en el contexto de reglas activas se presentan en el Capítulo 5. De igual manera se muestran los subtipos de error identificados.

El Capítulo 6 describe nuestro método de detección de errores. Las tres etapas de las que consiste este método: normalización, modelado y verificación de reglas se describen detalladamente. También se muestran algunas sugerencias para corrección de errores.

El análisis de confluencia y terminación que desarrollamos se presenta en el Capítulo 7. Una base de datos activa con enfoque en aplicaciones bancarias se usa para determinar si es confluente y termina.

El Capítulo 8 presenta una descripción de la implementación que realizamos, la cual llamamos ECAPNVerifier.

Finalmente, el Capítulo 9 muestra las conclusiones obtenidas y el trabajo futuro.

Capítulo 2

Sistemas Basados en Conocimiento

Los sistemas basados en conocimiento (SBC) son programas de computadora que intentan imitar la manera de resolver problemas que exhiben los expertos dentro de un área. Cuando se plantea un problema al SBC, este formula preguntas al usuario de manera efectiva y organizada para llegar a una solución. Además, es capaz de explicar el razonamiento mediante el cual llegó a esa conclusión y las soluciones potenciales que eliminó o descartó.

El ámbito de aplicación de los SBC se encuentra en dos grandes áreas: la investigación y la práctica. En el ámbito de la investigación, los SBC proveen la oportunidad de explorar una variedad de aspectos fundamentales, por ejemplo, el aprendizaje, el razonamiento inteligente y la representación y adquisición del conocimiento. Dentro del dominio de la práctica, hay ciertos propósitos genéricos en los cuales los SBC se pueden aplicar, entre ellos están: proveer consejos a los no-expertos, brindar asistencia inteligente a expertos, mantenimiento de estándares organizacionales y entrenamiento.

Los SBC tienen diferentes ventajas, entre las cuales podemos destacar las siguientes:

- Fácil modificación, ya que separa el conocimiento del mecanismo de razonamiento.
- Consistencia en las respuestas, porque siempre produce la misma respuesta al mismo problema, mientras que en la vida real diferentes expertos pueden tener respuestas distintas a la misma pregunta.
- Accesibilidad permanente, debido a que pueden trabajar las 24 horas del día, todos los días.
- Preservación del conocimiento, especialmente cuando los expertos tienen una mala salud.

- Explicación de la solución, ya que pueden presentar la línea de razonamiento seguida para llegar a una conclusión.

De igual manera los SBC tienen algunas desventajas, por ejemplo:

- Las respuestas no siempre son correctas, debido a que los expertos pueden cometer errores;
- El conocimiento está limitado al dominio del experto, ya que este es quien provee los hechos que ayudan a la solución del problema;
- La falta de sentido común.

A continuación describiremos algunos conceptos importantes de los SBC.

2.1. Conceptos básicos de los SBC

2.1.1. Definición de SBC

En términos generales, un SBC se puede definir como: "un sistema computarizado que usa el conocimiento que posee acerca de un dominio para resolver un problema perteneciente a ese dominio. Esta solución es esencialmente la misma a la que llegaría un experto en el área del problema cuando se le presenta el mismo obstáculo" [10]. Si la definición anterior se sigue estrictamente, un programa escrito en un lenguaje de programación tradicional que calcule corriente y voltaje en un circuito eléctrico y que para ello efectúe el mismo análisis que un experto en el área y use las mismas fórmulas podría ser erróneamente catalogado como SBC. Por lo tanto, hay que precisar la definición anterior con las siguientes características fundamentales que distinguen a los SBC de programas tradicionales:

1. La separación del conocimiento de la manera en la que este se usa.
2. El uso de un dominio altamente específico.
3. La heurística más que la naturaleza algorítmica del conocimiento empleado.

Las características anteriores se mezclaron durante el desarrollo del sistema MYCIN en la Universidad de Stanford en la década de los 70's [10]. El propósito de este sistema era diagnosticar y especificar tratamientos para alteraciones de la sangre a través de la "conversación" con un médico, quien realizaba preguntas acerca de los signos y síntomas del paciente y solicitaba los resultados de ciertos exámenes de laboratorio. Una vez que las condiciones del paciente se determinaban, el sistema recomendaba un tratamiento para corregir el desorden sanguíneo. Los desarrolladores de MYCIN trataron de cumplir las dos primeras características descritas, lo que les llevó a darse cuenta de que la estructura básica usada para manipular el conocimiento y llegar a un diagnóstico y tratamiento (llamada *mecanismo de inferencia*), podía ser usado con conocimiento de otros dominios para aplicar el mismo estilo de análisis diagnóstico.

Previamente, cada una de esas características se habían aplicado por separado, por ejemplo, el primer punto proviene de los Solucionadores Generales de Problemas (GPS, por sus siglas en inglés, *General Problem Solver*) desarrollados durante el final de la década de 1950 y principio de la década de 1960. En un GPS el usuario definía objetos y operaciones que se podían efectuar sobre esos objetos, entonces se generaban secuencias de operadores que de manera progresiva iban generando submetas hasta conectar un estado inicial del problema con un estado final (u objetivo).

La segunda característica fue explotada durante el desarrollo del sistema DENDRAL, el cual infiere la estructura molecular de compuestos desconocidos [10], al final de la década de 1960 y principio de la década de 1970.

La tercer característica se origina de la habilidad de los humanos para resolver problemas difíciles sin el uso constante de modelos o algoritmos.

2.1.2. Principales componentes de un SBC

Básicamente, un SBC consta de tres componentes: la base de conocimiento, la máquina de inferencia y la interfaz de usuario, los cuales interactúan de formas predeterminadas. Este sistema usualmente se representa como se muestra en la figura 2.1.

La principal característica que muestra la figura 2.1 es la separación del conocimiento tanto de la máquina de inferencia como de la interfaz de usuario, dando una estructura modular.

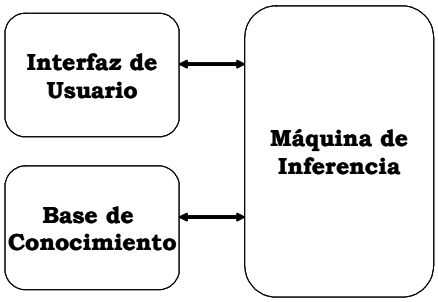


Figura 2.1: Principales componentes de un SBC

La base de conocimiento La base de conocimiento representa el componente más importante del SBC. Contiene todo el conocimiento relevante de un dominio específico necesario para resolver un problema, el cual ha sido recolectado por el ingeniero de conocimiento entrevistando a diferentes expertos. El conocimiento puede verse desde dos perspectivas: su naturaleza y su formato. La naturaleza se refiere a si el conocimiento es algorítmico o heurístico. La mayoría de las aplicaciones basadas en conocimiento ocurren en dominios donde el enfoque de resolución de problemas por medio de algoritmos no funciona bien o no existe. Entonces el conocimiento consiste de reglas empíricas que han sido aprendidas y desarrolladas a través del tiempo y de la práctica resolviendo problemas. El formato del conocimiento se refiere a la forma en la que el conocimiento se representa internamente dentro del SBC de manera que pueda ser usado para resolver problemas.

La máquina de inferencia La máquina de inferencia es el intérprete del conocimiento almacenado en la base de conocimiento. Examina el contexto de la base de conocimiento así como los datos acumulados acerca del problema actual y deriva datos y conclusiones adicionales. El conjunto de problemas que un SBC particular cubre puede ser visto como un gran grafo. En un lado del grafo están una serie de nodos que representan todos los signos, síntomas y características de interés que se usan como entradas cuando se intenta resolver un problema. Del otro lado, los nodos representan todas las posibles soluciones a un problema. Los arcos que conectan los nodos en el grafo describen el conocimiento usado por el experto durante la resolución del problema. El proceso de derivar una solución para resolver un problema puede verse como la forma de encontrar una conexión entre las entradas y las conclusiones. La creación de esas conexiones puede ocurrir de diferentes formas,

la máquina de inferencia puede intentar trabajar: 1) de las características hacia las conclusiones (llamado razonamiento hacia adelante); 2) de las soluciones a las características (llamado razonamiento hacia atrás); y 3) trabajar en ambas direcciones simultáneamente (llamado razonamiento bidireccional). El método usado depende de las características del problema particular así como del razonamiento del experto.

Interfaz de usuario La interfaz de usuario es la porción del SBC que maneja la conexión entre el sistema y el usuario.

2.1.3. Representación del conocimiento

Existen diferentes formas de representar el conocimiento dentro de un SBC: redes semánticas, tablas de decisión, reglas, marcos y demonios.

Redes semánticas Las redes semánticas utilizan nodos y enlaces para representar el conocimiento. Los nodos representan objetos, eventos o conceptos. Los enlaces muestran las relaciones entre los nodos. Aunque no hay reglas formales sobre la estructura de las redes semánticas, una aproximación clásica es representar los nodos como círculos o cajas y los enlaces por arcos dirigidos que conectan los nodos. Una red semántica debe ser examinada o validada analizando todas las direcciones de los arcos dirigidos. Entre las ventajas de las redes semánticas se encuentran la facilidad con que pueden representar las relaciones de herencia y su flexibilidad.

Tablas de decisión Las tablas de decisión son útiles para documentar procedimientos. Tienen diferentes ventajas, pero la más importante es que pueden ser entendidas y usadas tanto por los administradores como por el personal técnico.

Reglas Las reglas son un mecanismo popular para representar conocimiento heurístico. Las reglas más comúnmente usadas son las reglas de producción. Una regla de este tipo utiliza la sintaxis **IF** *condición* **THEN** *acción*. Dada una cierta situación, la máquina de inferencia busca satisfacer la condición declarada en la parte **IF** de la regla. Si la condición es verdadera, entonces se ejecuta la acción especificada en la parte **THEN**. Una modificación de este tipo de reglas usa la estructura **IF** *condición* **THEN** *acción 1* **ELSE** *acción 2* en donde se especifica una acción alternativa a realizar si

no se satisface la condición. Recientemente se han adoptado otro tipo de reglas, denominadas reglas evento - condición - acción (reglas ECA), para expresar conocimiento. La sintaxis de estas reglas consiste de tres elementos: **ON** *evento* **IF** *condición* **THEN** *acción*. Este tipo de reglas reaccionan ante ciertos eventos ejecutando acciones automáticamente si, previamente, se cumplieron algunas condiciones.

Marcos Las herramientas basadas en marcos usan un enfoque relacional de tablas para representar el conocimiento. Un marco describe un objeto que contiene toda la información acerca del mismo en forma de pares atributo - valor. La colección de atributos puede contener valores actuales, apuntadores a otros marcos, reglas o procedimientos para obtener un valor. Una de las grandes ventajas de los marcos es que los valores de atributos pueden consistir de procedimientos en lugar de ser valores estáticos.

Demonios Los demonios son funciones que no son invocadas explícitamente sino que hibernan hasta que ocurre una condición predefinida. Esta técnica permite representar el conocimiento en un nivel global sin seguir el protocolo de reglas dentro de un SBC. Esta es una gran ventaja pero también resulta peligroso debido a que el proceso no está relacionado directamente con ningún programa y puede ser computacionalmente costoso.

2.1.4. Incertidumbre dentro de los SBC

La mayoría de los SBC tienen algún grado de incertidumbre asociado a ellos. Los tipos de incertidumbre que pueden ocurrir en los SBC pueden ser causados por problemas con los datos. Por ejemplo: 1) pueden faltar datos o pueden no estar disponibles; 2) los datos pueden estar presentes pero no son confiables o son ambiguos; y 3) la representación de los datos puede ser imprecisa o inconsistente. De forma alternativa, la incertidumbre puede ser causada por la representación del conocimiento debido a que puede no ser apropiada en todas las situaciones.

Dadas esas fuentes de error, la mayoría de los SBC requieren incorporar alguna forma de manejo de incertidumbre. Cuando se implementa un esquema de manejo de incertidumbre se deben considerar los siguientes aspectos: 1) cómo representar los datos con incertidumbre; 2) cómo combinar dos o más piezas de datos con incertidumbre; y 3) cómo manejar la inferencia usando datos con incertidumbre.

Existen diferentes métodos que intentan manejar la incertidumbre de los SBC, entre ellos encontramos los siguientes: el enfoque Bayesiano, los factores de certeza, la teoría de la evidencia de Dempster-Shafer y la lógica y los conjuntos difusos [10]. El manejo de incertidumbre está más allá del ámbito de este trabajo de tesis, por lo que no abordaremos con más detalle este tema.

2.2. Desarrollo de SBC

Existen diferentes fuentes de error durante el proceso de construcción de un SBC: primero, los expertos piensan intuitivamente y, en consecuencia, omiten algunos pasos en el proceso de razonamiento; segundo, los sistemas se desarrollan de manera incremental, lo cual lleva a especificaciones inconsistentes, y tercero, se pueden introducir errores semánticos y sintácticos durante la implementación del sistema. Esto es especialmente cierto cuando múltiples expertos proveen información al sistema. La tarea de verificación requiere asegurar que las reglas en el sistema sean completas y consistentes.

2.2.1. Errores estructurales

Los *errores estructurales* típicos que se pueden producir durante el desarrollo de bases de reglas son los siguientes: *redundancia*, *inconsistencia*, *incompletitud* y *circularidad*. Se denominan estructurales por las relaciones sistemáticas que mantienen las reglas cuando aparece un error. Así, por ejemplo, si pudiéramos esquematizar el comportamiento del error de circularidad veríamos una secuencia que inicia y termina en el mismo punto.

Redundancia. La redundancia ocurre cuando existen reglas duplicadas o incluidas en la base de reglas. La redundancia incrementa el número de reglas y también puede causar inferencias adicionales inútiles. Un ejemplo de reglas incluidas es el siguiente:

$R1 : p \rightarrow q$

$R2 : p \rightarrow qr$

ya que $R2$ incluye a $R1$.

Inconsistencia. Si una base de reglas es inconsistente se producen hechos en conflicto, es decir, tanto el hecho q como su negación pueden ser inferidos de la base de reglas. El ejemplo más simple de una base de reglas que contiene reglas inconsistentes es el siguiente:

$R1 : p \rightarrow q$

$R2 : p \rightarrow \neg q$

Incompletitud. Las reglas están incompletas cuando algunas metas u objetivos son inalcanzables o una conclusión se vuelve un punto muerto esto es, no es ni una meta ni es usada como condición de otra regla. Si una base de reglas es incompleta (faltan reglas), no se puede derivar información útil de la base de reglas. Por ejemplo, si p es un hecho importante pero la base de reglas no tiene una regla como $R :\rightarrow p$ entonces p no puede ser inferida.

Circularidad. Este error ocurre cuando algunas reglas tienen una dependencia circular. La circularidad puede causar un razonamiento infinito que debe ser terminado. Un ejemplo de reglas circulares es el siguiente:

$R1 : p \rightarrow q$

$R2 : q \rightarrow p$

Un caso especial de circularidad es $R : p \rightarrow p$.

2.2.2. Verificación de errores

La naturaleza (a veces crítica) de las aplicaciones en donde se emplean los SBC requiere que estos estén libres de errores, por lo tanto, es necesario realizar una verificación formal para eliminar los errores introducidos en la fase de desarrollo.

Existen diversas definiciones de *verificación*, entre las que podemos destacar las siguientes:

- Construir el sistema correctamente [10].
- Demostrar la consistencia y completitud de un sistema [10].
- Demostrar la correctitud y propiedades de la estructura de la base de reglas [11].

Estas definiciones apuntan a que la verificación debe permitir examinar la correspondencia entre las especificaciones del sistema y su funcionalidad real y asegurar que el sistema está libre de errores introducidos por los desarrolladores durante la etapa de implementación.

Existen diversas propuestas para verificar que una base de reglas esté libre de errores. Los métodos tabulares [12], [13] fueron los primeros en ser diseñados para verificar anomalías en un sistema. Su estrategia es comparar, en pares, las reglas de la base de reglas con el fin de detectar

ciertas relaciones entre sus premisas y sus conclusiones. Por ejemplo, EMYCIN verifica la validez sintáctica de cada regla para detectar errores comunes, por ejemplo, que los términos estén escritos correctamente o que los valores asociados a un determinado parámetro estén dentro del dominio de ese parámetro. Ejecutar la verificación de las reglas en tiempo de adquisición reduce la probabilidad de que el sistema falle debido a errores "obvios". La verificación sintáctica de EMYCIN se hace simplemente comparando cada cláusula con una plantilla predefinida e identificando los casos en los que no coincide. Cuando se encuentra un parámetro que no está definido, el verificador intenta corregirlo con el corrector Interlisp, usando una lista de parámetros ya definidos en el sistema. Si no es posible corregirlo, pregunta al usuario si se trata de un nuevo parámetro y de ser así presenta una pantalla para describirlo. Se procede de manera similar cuando se encuentran valores ilegales en algún parámetro. También se puede realizar una verificación semántica limitada, cada vez que se agrega o se modifica una regla se compara con las ya existentes para asegurarse que no se contradice directamente con alguna de ellas o está incluida en alguna de ellas.

Los métodos basados en redes de Petri [6], [7], [11], [14], [15], [16], [17], [18], [19], [20] emplean esta herramienta para modelar la base de reglas y posteriormente aplican técnicas de análisis específicas a la red de Petri con el fin de detectar inconsistencias. Generalmente estos métodos necesitan checar el modelo bajo todos los posibles estados iniciales para garantizar la ausencia de errores, lo cual puede ser muy complicado.

Los métodos basados en grafos [21], [22] permiten representar dependencias conceptuales. Además, proveen métodos de análisis para estudiar propiedades tales como conectividad o alcanzabilidad, las cuales también se pueden verificar mediante los métodos basados en redes de Petri. La verificación basada en grafos, al igual que la verificación basada en redes de Petri, tiene la desventaja de que su ejecución necesita ser simulada usando todas las posibles combinaciones de hechos iniciales.

Los métodos basados en la generación de etiquetas [23], [24] se basan sobre el algoritmo ATMS. Estos métodos toman el concepto de contexto para especificar un conjunto de estados inciales, los cuales pueden causar la deducción de alguna inconsistencia. Por lo tanto, el objetivo de estos métodos es construir tales contextos.

Algunos métodos de verificación descritos previamente se han implementado y han dado origen a herramientas de verificación entre las que podemos citar las siguientes: EVA (Expert system Validation Associate), KB-REDUCER, COVER (COmpleteness VERifier), IMVER (Incidence Matrix

VERification), VERITAS, VALENS (VALid ENgineering Support) y OAV-VVT.

El proyecto EVA intentó construir un conjunto integrado de herramientas para verificar redundancia, consistencia, completitud y correctitud de cualquier SBC escrito en cualquier lenguaje para SBC. Se eligió la lógica de primer orden como forma de representar el conocimiento y Prolog como plataforma de implementación. Como era de esperarse, el proyecto EVA no resolvió el problema de forma general, sólo cumplió sus objetivos de manera limitada [25].

La primera versión de KB-REDUCER fue desarrollada por Allen Ginsbert en los Laboratorios AT&T en 1987. Esta herramienta se diseñó para verificar automáticamente la consistencia y redundancia de bases de conocimiento durante su mantenimiento. KBR1 se desarrolló para verificar sistemas basados en reglas escritas en lógica proposicional. El algoritmo de KBR está basado en ATMS y requiere que la base de conocimiento pueda ser estratificada en niveles [25].

La primera versión de COVER (COmpleteness VERifier) fue desarrollada en U.K. y se usó para verificar una aplicación médica de un SBC. COVER empleó Prolog como plataforma de implementación y utilizó como base de su verificación las definiciones formales de un conjunto de anomalías. Las operaciones de verificación de COVER están divididas en tres grupos: operaciones de verificación de integridad, operaciones de verificación de reglas y operaciones de verificación de cadenas de reglas [26].

La técnica de matriz de incidencia representa al conjunto de reglas mediante una matriz en la cual los elementos representan proposiciones presentes en una regla. Esta técnica fue usada por el sistema IMVER-1 el cual fue desarrollado en la Universidad de Liverpool como un sistema experimental de verificación de bases de reglas. Las principales críticas hechas a IMVER-1 estuvieron relacionadas con la cantidad de espacio de almacenamiento necesario para alojar las matrices que representaban bases de reglas de tamaño real y el poder de procesamiento necesario para llevar a cabo las operaciones de matrices requeridas. Para superar estas limitaciones, se desarrolló el sistema IMVER-2 (una actualización de IMVER-1) el cual utilizaba una codificación binaria para representar la matriz de la base de reglas [27].

VERITAS fue desarrollado para la verificación automática de la base de conocimiento SPARSE, aunque la herramienta fue desarrollada con propósitos de verificación genéricos. VERITAS es independiente del dominio y de la gramática de las reglas, por lo tanto, teóricamente, cualquier sistema basado en reglas puede ser analizado con él [28].

VALENS (VALid ENgineering Support) puede ser usado durante o después de la construcción de una base de conocimiento o puede ser integrado en una herramienta que permita a los usuarios escribir sus propias reglas de negocio. La entrada del componente de verificación es un conjunto de reglas y la salida de este componente es un conjunto de reglas inválidas. La entrada/salida es especificada en XML. VALENS puede ser integrado en el lenguaje de programación Aion el cual es un entorno de desarrollo para SBC [29].

2.3. Comentarios finales

Los SBC reflejan las habilidades que tiene una persona para resolver problemas de un cierto tipo. Para ello necesitan de tres partes principales: la base de conocimiento, la máquina de inferencia y la interfaz de usuario. La base de conocimiento es una de las partes más importantes de estos sistemas ya que contiene la información relevante acerca de una cierta área o dominio. Si el conocimiento no está correctamente expresado o es incompleto, la máquina de inferencia no podrá generar nuevos datos. Existen diferentes formas de representar el conocimiento, entre las que destacan las reglas de producción por la facilidad con que pueden expresar conocimiento de naturaleza heurística. En tiempos recientes, las reglas ECA también se han usado como medio de representación de erudición. La ventaja de este tipo de reglas es que pueden reaccionar automáticamente ante ciertos eventos producidos dentro o fuera del sistema.

Para asegurar que un SBC funcione de acuerdo a lo esperado, es necesario someterlo a un proceso de verificación, en el cual se buscan errores tales como: redundancia, inconsistencia, incompletitud y circularidad.

Capítulo 3

Reglas Activas

Los sistemas basados en reglas activas (también llamadas reglas Evento - Condición - Acción, ECA) son capaces de responder automáticamente a eventos que toman lugar dentro o fuera del sistema, ejecutando acciones establecidas con antelación. Esto libera al usuario de la obligación de realizar ciertas labores manualmente y, en consecuencia, los errores debidos a la omisión de tareas dentro del sistema disminuyen. El mecanismo mediante el cual estos sistemas pueden exhibir este comportamiento reactivo son las reglas activas.

3.1. Descripción de reglas activas

Generalmente, una regla activa tiene la forma [1], [2]:

`ON` *evento*

`IF` *condición*

`THEN` *acción*

Esta sintaxis le permite responder de forma automática, mediante la ejecución de acciones predefinidas, a eventos relevantes para el sistema siempre que se cumplan algunas condiciones. Por ejemplo, es posible definir una regla activa que supervise la temperatura ambiente y, en caso de que esta se encuentre por arriba de un cierto umbral, encienda el aire acondicionado. La regla ECA que describe tal comportamiento tiene la forma siguiente:

`ON` <*aumenta la temperatura*>

`IF` <*la temperatura está por arriba de un cierto nivel*>

`THEN` <*enciende aire acondicionado*>

En este documento, una regla activa se denota como $r(e, c, a)$ donde e, c, y a son el evento, la condición y la acción de la regla r. La base de reglas, R, está formada por varias reglas activas, es decir, $R = \{r_1, r_2, ..., r_n\}$ donde cada r_i tiene la forma descrita previamente.

Cada elemento de una regla ECA posee características que enriquecen su poder de expresión de conocimiento. A continuación se detalla cada elemento que conforma a una regla ECA.

3.1.1. Evento

Un evento es "*algo*" que sucede en un instante de tiempo dado. Su detección depende, en gran medida, de su *fuente* o *generador*. Un evento puede originarse por cualquiera de las siguientes fuentes [1]:

- *Operación de estructura.* El evento sucede por una operación sobre alguna parte de la estructura de la BD, por ejemplo, insertar una tupla o actualizar un atributo.
- *Invocación de comportamiento.* El evento sucede por la ejecución de operaciones definidas por el usuario.
- *Transacción.* El evento sucede por comandos de transacciones, por ejemplo, el comando abort.
- *Definidos por el usuario.* Se utiliza un mecanismo de programación para permitir que un programa de aplicación señale la ocurrencia de un evento, por ejemplo, como respuesta a alguna información introducida por el usuario.
- *Excepción.* El evento sucede como resultado de una excepción, por ejemplo, intentar acceder a los datos sin tener autorización.
- *Reloj.* El evento sucede en un instante de tiempo que puede ser absoluto, relativo a otro acontecimiento o periódico.
- *Externo.* El evento sucede por una situación fuera de la BD, por ejemplo, la temperatura subió por encima de 30 grados.

La *granularidad* de un evento indica si un evento está definido para todos los objetos en un conjunto, para un cierto subconjunto o para miembros específicos del conjunto.

Un evento puede ser de dos tipos: *primitivo* o *compuesto* [2]. El evento es primitivo cuando sucede por una ocurrencia de cualquiera de las fuentes descritas. El evento es compuesto cuando sucede por una combinación de eventos primitivos o compuestos usando una gama de operadores que constituyen el *álgebra de eventos*.

En la literatura se han propuesto álgebras de eventos para determinados sistemas [3], [5]; sin embargo, los operadores más comunes son los siguientes: *disyunción,* <OR*(E_1, E_2)*> sucede cuando cualquiera de los eventos E_1 o E_2 ocurre; *conjunción,* <AND*(E_1, E_2)*> sucede cuando ambos eventos E_1 y E_2 ocurren en cualquier orden; *secuencia,* <SEQ*(E_1, E_2)*> sucede cuando E_1 ocurre antes que E_2; *cerradura,* <CLOSURE*(E) en Int*> sucede la primera vez que E ocurre en el intervalo de tiempo *Int*, sin importar las posteriores ocurrencias de E; *historia,* <TIMES*(n, E) en Int*> se señala cuando el evento E ocurre n veces durante el intervalo de tiempo *Int*; *negación,* <NEG*(E) en Int*> detecta la no ocurrencia del evento E en el intervalo de tiempo *Int*; *último,* <LAST*(E) en Int*>, toma la última ocurrencia del evento E en el intervalo de tiempo *Int*; *simultáneo,* <SIM*(E1, E2)*> ocurre cuando suceden al mismo tiempo los eventos *E1* y *E2*; *alguno,* <ANY*(E1,E2,...,E_n,E_m)*> ocurre cuando han sucedido m eventos de n posibles.

Cuando se detecta un evento compuesto, diferentes instancias de eventos, tal vez del mismo tipo, pudieron haber ocurrido para formarlo. Por ejemplo, consideremos el evento compuesto SEQ(E_1, E_2). Si dos instancias e_1 y e_1' del evento E_1 ocurren en ese orden, y entonces una ocurrencia e_2 del evento E_2 sucede, la pregunta que surge es: ¿Qué instancias deben ser consideradas para formar el evento compuesto? Entre las posibilidades que existen se encuentran SEQ(e_1, e_2), SEQ(e_1 ',e_2), SEQ(e_1, e_2)∪SEQ(e_1 ', e_2). Para detectar la forma en que esos eventos se combinaron para formar el evento compuesto se usan las *políticas de consumo de eventos*. Las cuatro políticas de consumo más comunes son: contexto reciente, contexto cronológico, contexto continuo y contexto acumulativo. En la política de *contexto reciente*, como su nombre lo indica, se considera el conjunto de eventos más reciente que puede ser usado para formar la composición. Usando las instancias de eventos anteriores, el evento compuesto detectado sería SEQ(e_1 ', e_2) dado que e_1 ' y e_2 son las últimas instancias generadas de los eventos E_1 y E_2, respectivamente. La política de *contexto cronológico*, consume un evento en orden cronológico. De esta manera, en el ejemplo anterior el evento que se

detecta es SEQ(e_1, e_2) puesto que e_1 ocurrió antes que e_1' y e_2 es la unica instancia generada del evento E_2. La política de *contexto continuo*, define una ventana deslizante e inicia una nueva composición cada vez que se detecta un evento primitivo. En este caso se inicia la construcción de dos eventos secuencia, uno con la instancia e_1 y otro con la instancia e_1'. Ambos eventos se detectan cuando se genera la instancia e_2. Finalmente, la política de *contexto acumulativo*, almacena todos los eventos primitivos hasta formar el evento compuesto. El evento compuesto tomando en cuenta las instancias de eventos anteriores sería SEQ((e_1, e_1'), e_2). A diferencia de la política de contexto acumulativo, usando esta política de consumo sólo se genera un evento secuencia cuyo primer parámetro incluye tanto a e_1 como e_1'.

El *rol* de un evento indica si el evento siempre se debe especificar para las reglas activas o si puede omitirse. Si el rol es *opcional* y no se especifica el evento, se obtienen reglas de la forma condición - acción. Cuando el rol es *ninguno*, entonces no se pueden especificar eventos y todas las reglas son condición - acción. Si el rol es *obligatorio*, todas las reglas serán ECA porque el evento siempre debe ser especificado.

3.1.2. Condición

La condición de una regla evalúa el contexto en el cual el evento tomó lugar.

El *rol* de una condición indica si esta siempre debe especificarse [1]. En las reglas ECA la condición puede ser *opcional*. Si la condición no se especifica o el rol es *ninguno*, entonces se obtienen reglas de la forma evento - acción. Sin embargo, si el evento y la condición son opcionales, al menos uno de ellos debe ser especificado.

El *contexto* indica el marco en el cual será evaluada la condición. Cada uno de los componentes de las reglas ECA no se evalúa por separado ni de forma independiente de la BD. Por lo tanto, el procesamiento de una regla ECA puede estar asociado con cuatro estados de la BD: al inicio de la transacción actual; cuando ocurre un evento; cuando la condición es evaluada; y cuando la acción es ejecutada.

3.1.3. Acción

La gama de tareas que pueden ser llevadas a cabo por una acción se especifica como sus *opciones*. Las acciones pueden ser [4]: actualizar la estructura de la BD o de un conjunto de reglas, realizar

la invocación de comportamiento dentro de la BD o una llamada externa, informar al usuario o al administrador del sistema de alguna situación anómala, interrumpir una transacción, o tomar un curso de acción alternativo usando la cláusula *hacer – en_ lugar_ de*.

El *contexto* de una acción indica la información que está disponible para la acción.

En el ejemplo 3.1 se muestran algunas reglas activas tomadas de la referencia [3]. Estas reglas son parte de un sistema de BD activa, el cual consiste una una BD tradicional y una base de reglas (este tipo de sistemas se detallan con mayor exactitud en la Sección 3.3.1). Las reglas están definidas sobre el esquema de relación `account(`*num, name, balance, rate*`)` el cual almacena información de las cuentas bancarias de los clientes. Es importante notar que el atributo *rate* se encuentra en el intervalo [0, 10]. Las reglas verifican automáticamente el cumplimiento de algunas políticas establecidas por el banco.

Ejemplo 3.1 *Políticas para manejar las cuentas de los clientes en un banco.*

Política 1: Cuando se registra una nueva cuenta, si su balance es menor que \$500 y su tasa de interés es mayor a 0 %, entonces la tasa de interés de esa cuenta se fija en 0 %.

Política 2: Cuando se modifica la tasa de interés de una cuenta, si actualmente esta tiene un balance menor a \$500 y su tasa de interés es mayor a 0 %, entonces la tasa de interés de la cuenta se actualiza a 0 %.

Política 3: Cuando se modifica el valor del balance de una cuenta, si la tasa de interés de esa cuenta es mayor a 1 % y menor a 3 %, entonces la tasa de interés de la cuenta se actualiza a 2 %.

Política 4: Cuando se modifica el valor del balance de una cuenta, si la tasa de interés de esa cuenta es mayor a 1 % y menor a 3 %, entonces la cuenta es eliminada de la relación `account`.

Las políticas anteriores se pueden representar como reglas activas de la forma siguiente:

Regla 1

`ON` insert `account`

`IF` insert.*balance* < 500 and insert.*rate* > 0.0

`THEN` update `account` set *rate* $= 0.0$

where *balance* < 500 and *rate* > 0.0

Regla 2

`ON` update `account`.*rate*

`IF` update.*balance* $<$ 500 and update.*rate* $>$ 0.0

`THEN` update `account` set *rate* = 0.0

where *balance* $<$ 500 and *rate* $>$ 0.0

Regla 3

`ON` update `account`.*balance*

`IF` update.*rate* >1 and update.*rate* $<$ 3.0

`THEN` update `account` set *rate* = 2.0

where *rate* $>$ 1 and *rate* $<$ 3.0

Regla 4

`ON` update `account`.*balance*

`IF` update.*rate* >1 and update.*rate* $<$ 3.0

`THEN` delete from `account`

where *rate* $>$ 1 and *rate* $<$ 3.0

Definir reglas activas no es suficiente para lograr el comportamiento reactivo de los sistemas donde se incorporan. La definición de las reglas debe estar apoyada por una estrategia para ejecutarlas. Esta estrategia está descrita por su modelo de ejecución.

3.1.4. Modelo de ejecución

El modelo de ejecución indica la forma en la que un conjunto de reglas se trata en tiempo de ejecución.

Aunque este modelo está estrechamente relacionado con aspectos esenciales del SGBD, existen fases en la evaluación de una regla que merecen atención [1]:

- *Fase de señalización*, se refiere a la ocurrencia de un evento debido a una fuente de evento.
- *Fase de disparo*, toma los eventos producidos hasta el momento y activa las reglas correspondientes. Cuando se asocia una regla con la ocurrencia de un evento se instancia una regla.
- *Fase de evaluación*, evalúa la condición de la regla que ha sido activada. Si existen varias instancias de reglas y de todas ellas se satisface su condición entonces se tiene un conjunto de reglas en conflicto ya que todas ellas pueden ejecutar su acción al mismo tiempo.

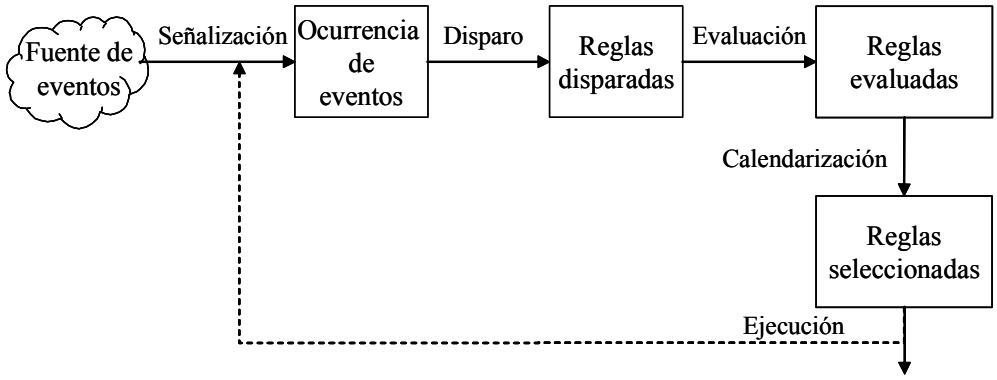


Figura 3.1: Principales pasos que toman lugar durante la ejecución de una regla

- *Fase de calendarización*, indica la forma de resolver los conflictos en el conjunto anterior.
- *Fase de ejecución*, realiza las acciones de la instancia de regla escogida. Durante la ejecución de la acción otros eventos pueden ser señalados lo que puede producir el disparo de reglas en cascada.

La figura 3.1 ilustra las fases descritas.

Todas esas etapas pueden no ser ejecutadas de forma secuencial, todo depende del modo de acoplamiento de los pares evento - condición y condición - acción. Los modos de acoplamiento más comunes son [1]:

- *Inmediato*, en este caso la condición (acción) se evalúa (ejecuta) inmediatamente después del evento (condición).
- *Diferido*, en este caso la condición (acción) se evalúa (ejecuta) dentro de la misma transacción que el evento (condición) de la regla, pero no necesariamente en la primera oportunidad.
- *Separado*, en este caso la condición (acción) se evalúa (ejecuta) en una transacción diferente de la del evento (condición).

La relación entre los eventos y las reglas que disparan se captura por la *granularidad de la transición*. Cuando la granularidad de la transición es *tupla*, la ocurrencia de un único evento

dispara una única regla. Cuando la granularidad es de tipo *conjunto*, se agregan los eventos a una colección que se usa para disparar una regla. Otra característica de las relaciones entre eventos y reglas disparadas, es la *política de efecto total*. Esta política indica si debe considerarse el efecto total de una cierta cantidad de eventos o deben considerarse cada uno por separado. Por ejemplo, si una instancia es actualizada y luego borrada, el efecto total es borrar la instancia original; si una instancia se inserta y luego se actualiza, el efecto total es insertar la instancia actualizada.

Cuando los eventos son señalados por la evaluación de la condición o acción de una regla se emplea una *política de ciclo*. En general, hay dos opciones. Si la política de ciclo es *iterativa*, la evaluación de la condición y la ejecución de la acción no se suspenden para responder a los eventos señalados. Si la política de ciclo es *recursiva*, los eventos señalados por la evaluación de la condición o acción provocan que la condición o acción se suspenda.

La fase de calendarización de evaluación de una regla determina qué sucede cuando múltiples o varias reglas se disparan al mismo tiempo. Los dos puntos principales de esta etapa son:

- *La selección de la siguiente regla a ser disparada.* El orden de las reglas puede influenciar fuertemente el resultado final y reflejar el tipo de razonamiento empleado por el sistema. Las reglas se pueden seleccionar por *aproximaciones dinámicas*, que son aquellas que dan prioridad a las reglas basándose, ya sea, en actualizaciones recientes o en la complejidad de la condición. Las *aproximaciones estáticas,* a menudo, son determinadas por el sistema o por el usuario como un atributo de la regla. En este último caso, una regla se selecciona de un conjunto de reglas disparadas simultáneamente usando un mecanismo de asignación de prioridades. Las reglas pueden ser colocadas en orden usando un esquema numérico, en el cual cada regla tiene un valor absoluto que es su prioridad.

- *El número de reglas a ser disparadas.* Entre las opciones se tiene: (1) disparar todas las reglas de forma secuencial; (2) disparar todas las reglas en paralelo; (3) disparar una regla por saturación y (4) disparar una o sólo algunas reglas.

El ejemplo 3.2 ilustra el proceso de ejecución, con el modo de acoplamiento inmediato, de las reglas activas usando la base de reglas descrita en el ejemplo 3.1

Ejemplo 3.2 *Proceso de ejecución de la base de reglas del ejemplo 3.1.*

account			
num	*name*	*balance*	*rate*
1	Juan Pérez	1000	1.5
2	Luis Martínez	1800	2.5
3	María Sánchez	2000	2.7
4	Ivonne López	10000	3.0
5	Mario Ibarra	300	0.0
6	Karla Hernández	100	0.0

Figura 3.2: Estado inicial de la relación `account`

Consideremos la relación `account` con la información mostrada en la figura 3.2.

Supongamos que se ejecuta la operación "update `account` set *balance* = 400 where update.*rate* > 1 and update.*rate* < 3". Este procedimiento asigna el valor 400 al atributo *balance* en la relación `account` de todas las tuplas cuyo valor en el atributo *rate* sea mayor a 1 y menor a 3. De forma que las tuplas con *num* 1, 2 y 3 ahora tienen el valor 400 en su atributo *balance*.

La operación anterior genera la detección el evento update `account`.*balance* (*fase de señalización*), el cual dispara la **Regla 3** y la **Regla 4** (*fase de disparo*). Posteriormente, se evalúa la condición de ambas reglas (*fase de evaluación*) para determinar el conjunto de acciones a ejecutar. El resultado de la evaluación es verdadero debido a que existen tuplas que poseen un valor en el atributo *rate* mayor a 1 y menor a 3, por lo tanto, se debe llevar a cabo la acción de ambas reglas. Por simplicidad, consideraremos el orden en que aparecen las reglas en la lista para ejecutar sus acciones (*fase de calendarización*). De esta manera, primero se efectúa la acción de la **Regla 3**, es decir, a todas las tuplas de la relación `account` que tengan un valor entre 1 y 3 en el atributo *rate*, se les asigna el valor 2 en ese atributo. El nuevo estado de la relación se muestra en la figura 3.3.

Dado que el modo de acomplamiento de las reglas es inmediato, la acción anterior trae como consecuencia la ocurrencia y detección del evento update `account`.*rate* el cual dispara la **Regla 2**. La evaluación de su condición es verdadera ya que existen tuplas cuyo valor de los atributos *balance* y *rate* es menor a 500 y mayor a 0, respectivamente. Por lo tanto, se ejecuta la acción de la **Regla 2** modificando el contenido de la relación `account` como se muestra en la figura 3.4.

La **Regla 2** se dispara nuevamente ya que su acción coincide con su evento. Sin embargo, en

account			
num	*name*	*balance*	*rate*
1	Juan Pérez	400	**2.0**
2	Luis Martínez	400	**2.0**
3	María Sánchez	400	**2.0**
4	Ivonne López	10000	3.0
5	Mario Ibarra	300	0.0
6	Karla Hernández	100	0.0

Figura 3.3: Estado de la relación `account` después de ejecutar la acción de la regla 3

account			
num	*name*	*balance*	*rate*
1	Juan Pérez	400	**0.0**
2	Luis Martínez	400	**0.0**
3	María Sánchez	400	**0.0**
4	Ivonne López	10000	3.0
5	Mario Ibarra	300	0.0
6	Karla Hernández	100	0.0

Figura 3.4: Estado de la relación `account` después de ejecutar la acción de la regla 2

esta ocasión la evaluación de su condición es falsa ya que no existen tuplas en la relación `account` que tengan un valor mayor a 0 en su atributo *rate*. De esta manera, termina el procesamiento de este conjunto de reglas. Finalmente, se lleva a cabo la acción de la **Regla 4**, la cual no tiene efecto sobre la información de la BD ya que no hay tuplas cuyo valor del atributo *rate* esté entre 1 y 3.

El procesamiento de la regla termina dado que no hay una regla que sea disparada por la última acción ejecutada.

3.1.5. Interacción de reglas

Durante la ejecución de las reglas, estas pueden interactuar de distintas formas. El *disparo*, la *activación/desactivación* y *conmutatividad* de reglas son las formas más comunes de interacción [3]. Para describir e ilustrar estos conceptos consideraremos la base del reglas del ejemplo 3.1.

La regla r_i *dispara* a r_j cuando la ejecución de la acción de la primera genera el evento que dispara a la segunda. La **Regla 3** dispara la **Regla 2** ya que la ejecución de la acción de la primera origina que se detecte el evento que la segunda regla está supervisando. De igual manera, la **Regla 2** se dispara a sí misma ya que su acción coincide con su evento.

La regla r_i *activa* a r_j siempre que la ejecución de la acción de la primera produzca datos que puedan satisfacer la condición de la segunda. En otras palabras, si la condición de r_j es verdadera después de que r_i ha ejecutado su acción, entonces r_i activa a r_j. La **Regla 3** activa a la **Regla 2** ya que modifica la información de la BD de manera que existen tuplas que pueden cumplir la condición de la segunda. Sin embargo, la **Regla 2** no se activa a sí misma ya que su acción no produce datos que puedan satisfacer su condición.

De manera similar, decimos que r_i *desactiva* a r_j si la ejecución de la acción de la primera elimina datos que puedan satisfacer la condición de la segunda. En un cierto momento, la **Regla 4** podría desactivar a la **Regla 3** ya que la acción de la primera elimina todas las tuplas de la relación `account` cuyo valor del atributo *rate* sea mayor a 1 y menor a 3. Como estas tuplas son necesarias para que la condición de la **Regla 3** sea verdadera y ya no existen en la relación, la regla queda desactivada.

Finalmente, r_i y r_j *conmutan* si se obtiene el mismo resultado independientemente del orden en el que se ejecuten sus acciones. La **Regla 3** y la **Regla 4** no son conmutativas porque si esta última se ejecuta primero puede modificar el comportamiento de la **Regla 3**.

La interacción de las reglas es un aspecto importante a considerar durante la ejecución de las mismas, especialmente durante la fase de calendarización ya que un orden inadecuado de ejecución puede alterar significativamente los resultados finales. También es relevante para el análisis de las propiedades de terminación y confluencia, las cuales se describen con detalle en la Sección 3.2.2.

3.2. Desarrollo de reglas activas

El desarrollo de aplicaciones activas no es un proceso sencillo debido a que muchas veces es difícil identificar las partes que deberían ser efectuadas usando mecanismos activos y las que deberían ser implementadas usando otras técnicas, así como cual es el costo que se debe pagar por el empleo de las reglas. Además, durante el proceso de diseño de las mismas se pueden introducir (inadvertidamente) errores difíciles de detectar a simple vista. De igual manera, se tiene que tomar en cuenta el cumplimiento de las propiedades de terminación y confluencia.

Por lo tanto, para asegurar una alta calidad de los sistemas activos es necesario contar con técnicas de verificación y análisis de reglas.

3.2.1. Verificación de reglas

La verificación de reglas de producción ha sido extensamente estudiada (ver Capítulo 2), no así la verificación de las reglas ECA. Sólo la referencia [5] aborda de manera colateral este problema. La referencia [47] examina el problema de *validación* de reglas que es un aspecto diferente de la verificación ya que mientras esta determina si se ha desarrollado un sistema correctamente, la validación examina si se ha obtenido un sistema correcto.

En el trabajo reportado en la referencia [5] se muestra el desarrollo de casas inteligentes para apoyar el cuidado y la supervisión del estado de salud de adultos mayores. Los autores implementaron un sistema que provee un análisis inteligente de los datos provenientes de la casa combinando la tecnología de BD activa y el razonamiento temporal. El sistema resultante es un Gestor de BD activa (SGBDA), el cual, a través de reglas ECA, puede detectar eventos generados por una persona. De acuerdo al contexto en el que tales eventos ocurran, el sistema produce alertas para notificar de situaciones inusuales o peligrosas. Un aspecto clave en este sistema es el tiempo en el cual ocurran los eventos.

Para definir las reglas ECA emplearon los operadores de Galton [5], los cuales están enfocados principalmente en eventos, estados y su interacción.

La motivación principal para realizar la verificación de reglas es el tipo de aplicación con el que están tratando ya que intervienen aspectos de seguridad y cuidado de la salud. Debido a esto su proceso de verificación está estrechamente relacionado con las características del sistema.

Las propiedades de las reglas que verifican son las siguientes: a) consistencia, se determina si las acciones de las reglas disparadas son consistentes. Además, de manera separada en tiempo de ejecución, también se verifica la consistencia de los eventos. b) Disparo: se examina si la especificación de cada cláusula `ON` puede satisfacerse alguna vez. c) Reunión: determina si cualquier subconjunto de reglas se puede disparar. Debido a la complejidad computacional de esta propiedad sólo se consideran combinaciones de n reglas y n es un número pequeño. d) Cobertura: se verifica si existen pares de reglas (r_1, r_2) tales que las condiciones de disparo de r_1 son un subconjunto de las de r_2. e) Cascada: analiza si existe una secuencia de activaciones entre reglas establecidas desde el diseño. f) Cumplimiento de la poscondición: examina si se puede ejecutar una determinada acción después de la activación de una regla.

Como se puede observar, estas propiedades están relacionadas con la semántica de las reglas más que con la ocurrencia de errores estructurales como los que se describieron en el Capítulo 2.

Estas características se verifican en el marco de la lógica métrica temporal ya que ofrece un lenguaje lo suficientemente expresivo para mapear las reglas ECA definidas para este sistema en él y expresar consultas que permitan examinar las propiedades anteriores.

En la referencia [47] se presenta un enfoque para validar reglas ECA, el cual se basa en la técnica de *planeación*. En esta técnica se establece un objetivo inicial y se determina una secuencia de acciones que asegura el cumplimiento del objetivo. Una sesión típica de validación con este enfoque no sólo permite al diseñador de reglas examinar una cierta propiedad de las reglas, sino también mostrar la salida del plan, el cual actúa como una herramienta de explicación mostrando una de las posibles secuencias de acciones que llevaron a un estado en el que se cumplió el objetivo. Las características de las reglas que se evalúan con este enfoque son las que se describen en la referencia [5], es decir, consistencia, disparo, reunión, cobertura, cascada y cumplimiento de la poscondición. Sin embargo, para que esta técnica pueda proporcionar resultados completos es necesario analizar el sistema en cuestión para todos los posibles objetivos iniciales, lo cual puede

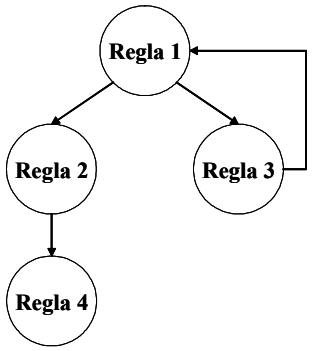


Figura 3.5: Grafo que muestra un posible disparo infinito de reglas

ser una tarea complicada.

3.2.2. Análisis de reglas

Durante el proceso de ejecución de las reglas, las características más importantes que se deben analizar son la terminación y confluencia.

La propiedad de *terminación* garantiza que el proceso de ejecución de una base de reglas se realiza en un número finito de pasos, es decir, que no existe un subconjunto de reglas que se está disparando entre sí continuamente formando un ciclo [1]. La figura 3.5, en la cual los nodos representan reglas y las aristas la relación PUEDE-DISPARAR, muestra que el proceso de ejecución de la **Regla 2** yla **Regla 4** termina pero cualquier disparo de la **Regla 1** ó la **Regla 3** inicia una serie de disparos que puede no terminar.

Las relaciones de disparo entre las reglas del ejemplo ejemplo 3.1 se muestran en la figura 3.6. La regla 2 forma un autociclo porque la ejecución de su acción genera el evento que la dispara. Sin embargo, como se ha descrito en el ejemplo 3.2, cuando la acción de la regla 2 se efectúa la evaluación de su condición es falsa, por lo que su proceso de ejecución termina sin que se genere el ciclo.

El análisis de *confluencia* determina si el orden de ejecución de reglas disparadas simultáneamente en un mismo estado del sistema influye en el resultado final que se obtenga. Para explicar el concepto de confluencia consideremos la figura 3.7 en donde cada nodo representa un estado

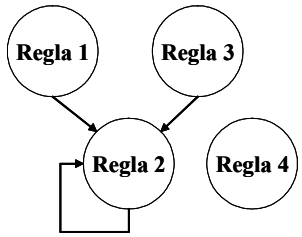


Figura 3.6: Relaciones de disparo entre las reglas del ejemplo 3.1

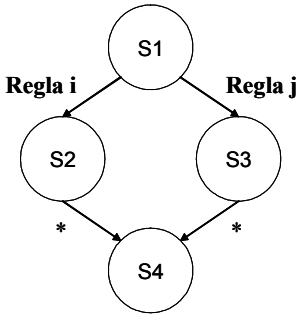


Figura 3.7: Grafo que muestra el comportamiento confluente de una base de reglas

del sistema. El estado inicial `S1` tiene como sucesores a los estados `S2` y `S3`, los cuales se obtienen después de que se disparan las reglas i y j, respectivamente. Cada una de estas reglas pudo haber originado disparos subsecuentes de otras reglas (denotados con "*" en el grafo). Si al finalizar cada secuencia de disparo se obtiene un único estado final `S4` independientemente del orden en que se hayan ejecutado las secuencias, entonces la base de reglas es confluente.

En la fase de calendarización del ejemplo 3.2 se debe asignar un orden de ejecución para las acciones de la **Regla 3** y la **Regla 4** considerando el estado de la BD mostrado en la figura 3.2. El resultado al ejecutar primero la acción de la **Regla 3** (y las acciones generadas como consecuencia) y después la acción de la **Regla 4** se muestra en la figura 3.4. Si el orden de ejecución se invierte, entonces la acción de la **Regla 4** remueve las tuplas de la relación `account` cuyo valor del atributo *rate* está entre 1 y 3. El resultado de esta acción deja el estado mostrado en la figura 3.8.

Posteriormente se ejecuta la acción de la **Regla 3** la cual no modifica ninguna tupla, ya que

account			
num	*name*	*balance*	*rate*
4	Ivonne López	10000	3.0
5	Mario Ibarra	300	0.0
6	Karla Hernández	100	0.0

Figura 3.8: El estado de la relación `account` después de ejecutar la acción de la **Regla 4**

todas las que podía modificar fueron eliminadas por la acción previa, pero genera el evento que dispara la **Regla 2**. Dado que en este momento no hay tuplas en la relación `account` que satisfagan la condición de esta relación, el proceso de ejecución de las reglas termina.

Como se puede observar, el estado de la BD al ejecutar la secuencia **Regla 3**, **Regla2** y **Regla 4** (figura 3.4) es diferente del que se obtiene al ejecutar la secuencia **Regla 4** y **Regla 3**, **Regla 2** (figura 3.8). Por lo tanto, la base de reglas del ejemplo 3.1 no es confluente.

Diversos estudios acerca de terminación y confluencia se han propuesto en la literatura. Algunos de ellos están basados en el análisis de grafos [30], [31], [32], [33], [34], [35], [36]; otros se basan en el álgebra relacional [37]; algunos más tratan a las reglas activas como reglas deductivas [38] y otros abordan el problema por medio de las redes de Petri [41], [39], [40].

En el trabajo reportado en [30] se analiza la propiedad de terminación usando la gráfica de disparo la cual es un grafo dirigido cuyos nodos representan reglas y los arcos indican que una regla produce un evento que puede disparar otra regla. La ejecución de las reglas termina si el grafo es acíclico y puede no terminar de otra forma. Esta aproximación es conservativa en el sentido que los problemas potenciales de no-terminación se detectan incluso cuando en la práctica el proceso de ejecución de las reglas finaliza. El análisis de confluencia se realiza sobre el grafo de ejecución, cuyos nodos representan estados del sistema y los arcos dirigidos se etiquetan con el nombre de las reglas cuya ejecución hace que se pase de un estado a otro. Si el grafo es acíclico y cada par de reglas conmuta, es decir, la ejecución de la primer regla seguida de la segunda y viceversa produce el mismo estado, el proceso de ejecución de las reglas es confluente.

En la referencia [31] el análisis de la gráfica de disparo se complementa con el análisis de la gráfica de activación, en la cual los nodos representan reglas y los arcos indican que la acción de

una regla puede modificar la evaluación de la condición de otra regla. El proceso de ejecución puede no terminar si ambos grafos tienen un ciclo en común.

El análisis de terminación también puede abordarse empleando los grafos de evolución [34], [35], [36]. Este tipo de grafos simulan estáticamente el proceso de ejecución de las reglas considerando las relaciones de activación y desactivación entre las mismas. El grafo de evolución provee un análisis más detallado que las gráficas de activación y disparo. Sin embargo, el problema de confluencia no se aborda con este enfoque.

En la referencia [37] se propone un algoritmo para determinar si un conjunto de reglas termina y es confluente. Este algoritmo propaga el efecto de la acción de una regla condición - acción en la condición de otra regla de este tipo. Tanto la condición como la acción de la regla se expresan en un álgebra relacional extendida. Este algoritmo se integra al análisis basado en grafos para determinar cuándo un arco tiene que se incluido en la gráfica de disparo y para verificar si dos reglas conmutan. Sin embargo, hay casos especiales en los cuales para dos operaciones de actualización `M1` y `M2` el algoritmo de propagación produce la respuesta "puede no conmutar" cuando de hecho ambas conmutan. Debido a esto, los autores proponen también una serie de mejoras para su algoritmo.

En [38] se abordan los problemas de terminación y confluencia traduciendo las reglas activas en reglas deductivas. Durante esta traducción se toma en cuenta la semántica de ejecución de las reglas (inmediata o diferida). Una vez que las reglas han sido traducidas, se aplican algunos resultados conocidos en reglas deductivas, especialmente los obtenidos con Datalog, para decidir si las reglas activas terminan y son confluentes. Cabe mencionar que las autoras proponen un algoritmo para asignar prioridades a las reglas y de esta forma asegurar que la ejecución de las reglas siempre será determinista.

Los trabajos reportados en [39] y [41] modelan las reglas activas con redes de Petri. En general los lugares de entrada representan el evento de la regla, las transiciones representan la condición y los lugares de salida la acción. Posteriormente se aplican técnicas de análisis tales como el árbol de alcanzabilidad para decidir si una base de reglas termina y es confluente. Particularmente, en [41] se aplica esta técnica para abordar el problema de terminación. Es necesario que se proporcione una marca inicial y, dado que los lugares de la red de Petri no están limitados, se debe definir un número de tokens que cada lugar puede almacenar. La construcción del árbol de alcanzabilidad terminará siempre que una de las siguientes situaciones ocurra: 1) se llegue al número especificado,

2) durante una secuencia de disparo de transiciones una marca se alcance más de una vez, y 3) todas las posibles secuencias de ejecución lleguen a un estado final. Las primeras dos situaciones muestran que la ejecución de las reglas puede no terminar, mientras que la tercera situación muestra que la ejecución de las reglas termina. Hay que notar que los resultados dependen de la marca inicial, si esta no se elige correctamente entonces no es posible detectar los problemas. En [40] se explora el problema de terminación por medio de la construcción de una red de Petri Coloreada Condicional, la cual describe con mayor detalle la interacción de las reglas. Si la red no contiene ciclos se puede asegurar que el procesamiento de las reglas terminará, de otra forma no se puede asegurar este resultado. Para descubrir automáticamente la existencia de ciclos se usa la matriz de conexión de la red. En este enfoque no se toma en cuenta la interacción de las reglas por lo que se detectan problemas de no-terminación aún cuando estos no ocurran. Además, en este trabajo no se aborda el análisis de la propiedad de confluencia.

3.3. Aplicaciones

Existen diversas sistemas en los cuales las reglas activas juegan un papel preponderante, entre ellas podemos mencionar las BD activas, la tecnología de casas inteligentes, las aplicaciones financieras y el mantenimiento de restricciones de integridad.

3.3.1. Bases de datos activas

Las BD activas (BDA) combinan la tecnología de BD tradicional con la programación basada en reglas para lograr que la BD reaccione ante ciertos estímulos, posiblemente externos, llamados eventos. La capacidad reactiva de las BDA's es útil para un amplio espectro de aplicaciones, las cuales incluyen seguridad, materialización de vistas, verificación de integridad o integración de BD heterogéneas. Un sistema de BDA consiste de una BD (pasiva) y un conjunto de reglas activas, tal como se muestra en la figura 3.9. Cuando ocurre un evento relevante dentro o fuera del sistema, el Sistema Gestor de BD activa (SGBDA) lo ejecuta en la BD y también verifica si en la base de reglas existe alguna que responda a tal evento. De ser así, se efectúan los pasos del modelo de ejecución.

Diversos sistemas y prototipos de BDA's se han diseñado e implementado parcial o completamente. Desafortunadamente, estos sistemas han sido diseñados de manera independiente, sin el

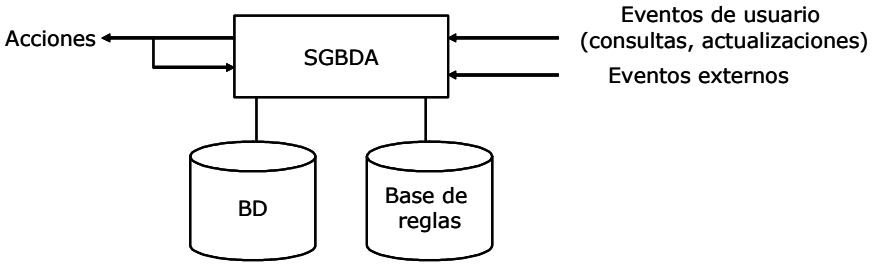


Figura 3.9: Sistema de base de datos activa

apoyo de una teoría común que guíe la semántica de las reglas y, por lo tanto, se obtienen comportamientos diferentes para reglas similares.

Un ejemplo de BDA, tomado de la referencia [7] se muestra a continuación.

Ejemplo 3.3 *Sistema de BDA para manejar las ventas y bonos de empleados en una compañía.*

La BD consta de las relaciones `EMPLEADO`, `PRIMA` y `VENTAS` las cuales almacenan la información del empleado, la cantidad de ventas que este realizó en el mes, y el bono recibido si es que se superó el límite de venta, respectivamente.

`EMPLEADO(`*emp_id, nombre, rango, salario*`)`

`VENTAS(`*emp_id, mes, número*`)`

`PRIMA(`*emp_id, cantidad*`)`

Las políticas que se aplican a los empleados son las siguientes:

Política 1. Cuando la cantidad de la prima de un empleado se modifica, si la cantidad es mayor que $100.00, entonces el rango del empleado se incrementa en uno.

Política 2. Cuando el rango de un empleado se modifica, si el rango ahora es mayor que el nivel 5, entonces la prima del empleado se incrementa en diez veces el nivel del rango.

Política 3. Cuando se obtienen las ventas del mes y el número de estas es superior a 50, entonces el rango del empleado se incrementa en un nivel.

Política 4. Cuando el nivel del rango de un empleado se modifica y el rango alcanzó el nivel 15, entonces el salario del empleado se incrementa en un 10 %

Las políticas anteriores se traducen en las siguientes reglas ECA:

Regla 1

ON update PRIMA.*cantidad*

IF update.*cantidad* > 100.0

THEN update EMPLEADO set *rango* = EMPLEADO.*rango* + 1

where EMPLEADO.*emp_id* = update.*emp_id*

Regla 2

ON update EMPLEADO.*rango*

IF update.*rango* > 5

THEN update PRIMA set *cantidad* = PRIMA.*cantidad* + update.*rango* * 10

where PRIMA.*emp_id* = update.*emp_id*

Regla 3

ON insert VENTAS

IF insert.*número* > 50

THEN update EMPLEADO set *rango* = old.*rango* + 1

where EMPLEADO.*emp_id* = insert.*emp_id*

Regla 4

ON update EMPLEADO.*rango*

IF update.*rango* = 15

THEN update EMPLEADO set *salario* = old.*salario* * 1.1

where EMPLEADO.*emp_id* = update.*emp_id*

3.3.2. Casas inteligentes

Las casas inteligentes usan pequeñas computadoras distribuidas en la casa las cuales se usan para encender/apagar electrodomésticos o para enviar/recibir información. El uso de las computadoras elimina la necesidad de encender un switch o girar una perilla para hacer que las cosas funcionen y permite que los elementos de la casa sean controlados remotamente por, o para responder automáticamente a, las personas que habitan en ella [5]. Esta tecnología es particularmente útil para que los adultos mayores puedan seguir viviendo de manera independiente ya que ofrece una solución viable para ayudarlos en el cuidado de su salud.

La información generada de los sensores dentro de la casa y las conexiones embebidas en aplicaciones domésticas comunes se procesan por un SGBDA para proveer asistencia a las personas de diferentes maneras: prevención de situaciones peligrosas, confort, seguridad y cuidado de la salud.

Prevención de situaciones peligrosas. En estas situaciones la información recolectada del entorno de las personas puede ser verificada y también se pueden tomar en consideración las acciones realizadas previamente. El propósito es proveer una alarma cuando las actividades del habitante lo puedan llevar a situaciones potencialmente peligrosas. Por ejemplo, una persona puede dirigirse de una habitación a otra dentro de la casa y dejar algún electrodoméstico encendido en la habitación de la cual salió. Si la televisión se deja funcionando no es peligroso, pero si se deja la olla de presión y se va a tomar un baño esto sí puede resultar riesgoso y, por lo tanto, se requiere de alguna forma la intervención del sistema. Esto puede representarse con la siguiente regla ECA:

`ON` <*salir de la habitación*>

`IF` <*algún aparato está encendido*>

`THEN` <*activar alarma*>

Confort. Este aspecto se enfoca en el medio ambiente y lo altera de acuerdo a las necesidades de la persona y sus acciones. Por ejemplo, si la temperatura ambiente sube o baja de un cierto límite, entonces hay que reajustarla dentro de la casa.

`ON` <*temperatura de la habitación por debajo del valor normal*>

`IF` <*hay alguien en la casa*>

`THEN` <*encender el calentador*>

Seguridad. La gente mayor es vulnerable a ataques de intrusos en su casa. En tales casos, se deben proveer alarmas cuando sea necesario.

`ON` <*alarma activada por un intruso*>

`IF` <*no hay personal esperando en la posición referida*>

`THEN` <*aplica el procedimiento de seguridad*>

Cuidado de la salud. Aquí la información puede ser accedida desde dispositivos de cuidado de la salud auto-operados, por ejemplo, monitores de presión sanguínea o glucosa. Algunas veces es necesario monitorear signos vitales periódicamente. Por ejemplo, es posible representar con reglas ECA el monitoreo de la presión sanguínea. Si se presenta un incremento sustancial en dos mediciones sucesivas, entonces el personal médico debe ser alertado de esa situación.

`ON` *<presión sanguínea mayor a 200/175 durante dos muestras sucesivas>*

`IF` *<el régimen médico para el control de la presión sanguínea ha sido alterado recientemente>*

`THEN` *<notificar al personal médico>*

3.3.3. Aplicaciones financieras

Un portafolio bursátil se define como una "combinación lineal de instrumentos financieros" [8] tales como bonos. Los administradores de los portafolios se evalúan sobre qué tan bien se comporta su portafolio con respecto a un punto de referencia, el cual está relacionado con medidas numéricas tales como intereses, los cuales se calculan sobre pagos periódicos, y el valor actual del mercado. Además de esas medidas de desempeño, se establecen diferentes restricciones sobre el tipo de inversión permitida para un portafolio dado. La presencia de esas restricciones y medidas de desempeño junto con la naturaleza dinámica del mercado resulta en un portafolio que frecuentemente cambia su composición, tanto en términos de los instrumentos y la cantidad de esos intrumentos presentes en el portafolio. Debido a esto, una BD que apoye eficientemente a las aplicaciones financieras debe proveer apoyo para desarrollar y administrar objetos temporales así como administración del conocimiento, la capacidad de almacenar e imponer reglas que reflejen la semántica de la aplicación a través de reglas temporales y no temporales. Las reglas son el aspecto más importante de la tecnología de BDA para aplicaciones financieras. Facilidades tales como apoyo temporal, tipos de datos abstractos y funciones definidas por el usuario se usan en las reglas para expresar el conocimiento y la semántica de la aplicación. Un ejemplo de una regla aplicada al contexto financiero es la siguiente:

`ON` update `stock.`*price*

`IF` *true*

`THEN` update-monitor( )

El objetivo de esta regla es actualizar el precio sobre la pantalla de un comerciante tan pronto como un nuevo precio haya sido registrado en la BD. Claramente, no se quiere que el evento y la acción estén en la misma transacción porque si la acción falla, aún se desea que el precio de las acciones sea actualizado. Por otro lado, si la transacción en que se ejecuta el evento es abortada, no hay necesidad de actualizar el monitor. De esta forma, el evento y la acción deben ejecutarse en transacciones separadas.

3.3.4. Mantenimiento de restricciones de integridad

Los sistemas de BD comerciales inicialmente motivaron la introducción de reglas activas (*triggers*) como un mecanismo para programar la integridad referencial entre tablas. Por lo tanto, el mantenimiento de las restricciones de integridad constituye una de las áreas principales de aplicación de las reglas activas.

Las restricciones de integridad pueden clasificarse en estáticas y dinámicas. Las primeras son predicados sobre estados de la BD. Las últimas son predicados sobre transiciones de un estado de la BD a otro. El proceso de verificación de restricciones consiste en evaluar los predicados en orden para determinar su valor de verdad. La verificación de la restricción puede ser inmediato o diferido. La verificación inmediata toma lugar inmediatamente después de la ejecución de una operación que puede violar la restricción; el modo diferido se pospone para el final de la transacción que contiene la operación que puede violar la restricción. Cuando la evaluación de la restricción produce un valor de verdad erróneo, la restricción es violada. El sistema de mantenimiento de integridad reacciona a la violación ya sea "reparando" la restricción, es decir, llevando a cabo cambios a la BD para restaurar el valor de verdad de la restricción, o deshaciendo ya sea el enunciado que causó la violación o la transacción completa. Como ejemplo consideremos la siguiente definición hecha en Structured Query Language (SQL) de la relación Empleado(NombreE, NSS, NSSSuper, ND, Salario) [9].

```
Create_Table EMPLEADO(
    NombreE Varchar(15) Not Null,
    NSS Char(9) Not Null,
    NSSSUPER Char(9) Not Null,
    ND Integer(1) Not Null Default 1,
    Salario Decimal(10, 2),
    Constraint CVEEMP
        Primary Key (NSS),
    Constraint CVESUPEMP
        Foreign Key(NSSSUPER) References EMPLEADO(NSS)
            ON Delete SET Null ON Update Cascade,
    Constraint CVEDEPTOEMP
        Foreign Key(ND) References DEPARTAMENTO(NUMERO)
```

ON Delete SET Default ON Update Cascade);

La instrucción CREATE_TABLE define un esquema relación de BD, sus atributos y restricciones de integridad. En este caso se define la relación `EMPLEADO` con los atributos *NombreE*, *NSS*, *NSSSuper*, *ND* y *Salario* los cuales almacenan la información del nombre del empleado, su número de seguro social, el número de seguro social de su supervisor, el número de departamento al que está adscrito y su salario, respectivamente. Cada atributo se define junto con su tipo de dato y una indicación de si puede tomar un valor nulo o no.

La palabra reservada CONSTRAINT define las restricciones que se deben verificar automáticamente para este esquema. La primera restricción, llamada **CVEEMP**, verifica que el atributo que identifica de manera única al empleado, *NSS*, no pueda tomar un valor nulo. La segunda restricción, **CVESUPEMP**, supervisa los cambios que se hagan al número de seguro social de los empleados que son supervisores. Si se elimina un empleado que es supervisor de otro(s), de forma automática se pone un valor NULL en el atributo *NSSSuper* de todos aquellos empleados que tenían como supervisor al empleado eliminado. En caso de que sólo haya cambiado su número de seguro social, este cambio se propaga al atributo *NSSSuper* de todos los empleados que supervisa. Estas acciones se hacen de forma automática lo que permite mantener fácilmente la consistencia de los datos. La restricción **CVEDEPTOEMP** actúa de manera similar vigilando cuando cambia algún número de departamento para propagar este cambio.

3.4. Comentarios finales

Las reglas activas son un mecanismo poderoso para realizar tareas de manera automática como respuesta a sucesos importantes para una aplicación. Sin embargo, su desarrollo no es una tarea sencilla. Se deben tomar en cuenta los errores introducidos (inadvertidamente) y que pueden causar un mal funcionamiento. Los errores que se deben identificar en la base de reglas son: redundancia, inconsistencia, incompletitud y circularidad. Además, se tiene que considerar el cumplimiento de las propiedades de terminación y confluencia, las cuales garantizan que el proceso de ejecución se realiza en un número finito de pasos y que se obtiene el mismo estado final en el sistema no importando el orden en el que se ejecuten reglas disparadas simultáneamente, respectivamente. Por lo tanto, es necesario desarrollar métodos de verificación de errores y análisis de reglas.

Capítulo 4

Modelado y Simulación de Reglas Activas con CCPN

Existen diferentes trabajos en la literatura para modelar reglas ECA y su interacción, por ejemplo, la gráfica de disparo, la gráfica de activación [45], la gráfica de disparo modificada [42], los grafos de evolución [35], [36] y la red de Petri Coloreada Condicional [46]. La gráfica de disparo es un grafo cuyos nodos representan reglas y los arcos que unen los nodos significan que una regla puede disparar a otra. La gráfica de disparo modificada enriquece la gráfica de disparo agregando el concepto de ancla. Un ancla representa un evento externo. La gráfica de activación es similar a la gráfica de disparo en el sentido de que los nodos representan a las reglas, sin embargo, los arcos que unen los nodos significan que la acción de una regla puede ocasionar que la condición de otra regla sea verdadera. Los grafos de evolución simulan el proceso de ejecución de las reglas considerando la interacción entre ellas, para lograrlo se auxilian de la gráfica de disparo. Los modelos anteriores representan reglas de manera muy general, de forma que detalles importantes como la generación de eventos compuestos se omiten. La red de Petri Coloreada Condicional es una extensión de las redes de Petri diseñada especialmente para modelar reglas ECA y su interacción.

4.1. Conceptos preliminares

Una red de Petri (PN, por sus siglas en inglés *Petri Net*) es un grafo dirigido que consiste de dos tipos de elementos llamados *lugares* y *transiciones.* En el grafo, los lugares se representan como círculos y las transiciones como barras o rectángulos. Un arco dirigido conecta un lugar con una transición y viceversa. Los arcos se etiquetan con un entero positivo que representa su *peso*, de manera que un arco con peso k representa un conjunto de k arcos paralelos y si un arco no tiene etiqueta, su peso es uno. Para denotar el estado del sistema modelado, la PN puede estar *marcada.* Una *marca* asigna a cada lugar un entero positivo k, que representa el número de *tokens* colocados en ese lugar. Gráficamente, se colocan k puntos negros (tokens) en el lugar p.

Formalmente, una PN se define de la siguiente manera [44]:

Definición 4.1 *Una PN es una 5-tupla,* $PN = \{P, T, F, W, M_0\}$*, donde:*

$P = \{p_1, p_2, \ldots, p_m\}$ es un conjunto finito de lugares
$T = \{t_1, t_2, \ldots, t_n\}$ es un conjunto finito de transiciones
$F \subseteq (P \times T) \cup (T \times P)$ es un conjunto de arcos
$W : F \to \{1, 2, 3, \ldots\}$ es una función de peso
$M_0 : P \to \{0, 1, 2, 3, \ldots\}$ es el marcado inicial
$P \cap T = \emptyset$ y $P \cup T \neq \emptyset$

El peso de un arco se denota por $w(p, t)$ cuando el lugar p se conecta con la transición t, o $w(t, p)$ cuando la transición t se conecta con el lugar p. Una marca, denotada por M, es un vector de m componentes, donde m es el número total de lugares. El n-ésimo componente de M, denotado por $M(p_n)$, es el número de tokens en el lugar p.

El conjunto de los *lugares de entrada* de una transición $t \in T$ se representa por ${}^\bullet t = \{p \mid p \in P, t \in T, p$ es un lugar de entrada de $t\}$. El conjunto de los *lugares de salida* de una transición $t \in T$ se representa por $t^\bullet = \{p \mid p \in P, t \in T, p$ es un lugar de salida de $t\}$. El conjunto de transiciones que se conectan hacia un lugar $p \in P$ se representa por ${}^\bullet p = \{t \mid p \in P, t \in T, t$ es una transición de entrada de $p\}$. Finalmente, el conjunto de transiciones de salida de un lugar $p \in P$ se representa por $p^\bullet = \{t \mid p \in P, t \in T, t$ es una transición de salida de $p\}$.

La ejecución de la PN está determinada por la cantidad y distribución de los tokens en la red, así como por la aplicación de las reglas de habilitación y disparo.

1. *Regla de habilitación de transiciones.* Una transición t se dice que está habilitada si cada $p \in {}^\bullet t$ contiene, al menos, un número de tokens igual al peso del arco dirigido que conecta a p con t, es decir, $M(p) \geqslant w(p,t), \forall p \in {}^\bullet t$

2. *Regla de disparo de transiciones.*

   - Una transición habilitada t puede dispararse o no dependiendo de la interpretación adicional que tenga la transición, y
   - El disparo de una transición habilitada t elimina de cada lugar de entrada p el mismo número de tokens que el valor del peso del arco dirigido que conecta a p con t. Además, agrega a cada lugar de salida p un número de tokens igual al peso del arco dirigido que conecta a la transición t con el lugar p.

Matemáticamente, el disparo de t en M alcanza una nueva marca:
$M'(p) = M(p) - w(p,t) + w(t,p), \forall p \in P$

Ejemplo 4.1 *Una red de Petri sencilla.*

La figura 4.1 muestra una PN sencilla. Esta estructura contiene los siguientes datos:
$P = \{p_1, p_2, \ldots, p_7\}$
$T = \{t_1, t_2, \ldots, t_5\}$
$F = \{(p_1,t_1),\ (p_2,t_2),\ (p_3,t_3),\ (p_4,t_4),\ (p_5,t_5),\ (p_7,t_4),\ (t_1,p_2),\ (t_1,p_3),\ (t_2,p_4),\ (t_2,p_7),\ (t_3,p_5),\ (t_4,p_6),\ (t_5,p_6)\}$
$w(p_1,t_1) = w(t_1,p_3) = 2$. El peso del resto de los arcos es 1.
$M_0 = (2\ 0\ 0\ 0\ 0\ 0\ 1)^T$

Bajo la marca inicial $M_0 = (2\ 0\ 0\ 0\ 0\ 0\ 1)^T$, sólo la transición t_1 está habilitada. Cuando esta transición se dispara, de acuerdo a la regla de disparo, se remueven los dos tokens del lugar p_1 y se depositan dos tokens en el lugar p_3 y un token en el lugar p_2. La nueva marca obtenida es $M_1 = (0\ 1\ 2\ 0\ 0\ 0\ 1)^T$. La figura 4.2 muestra la PN con la nueva marca.

A partir de la última marca obtenida, las transiciones t_2 y t_3 se han habilitado. Si se dispara t_2 origina la marca $M_2 = (0\ 0\ 2\ 1\ 0\ 0\ 0)^T$ mientras que si se dispara t3 la nueva marca obtenida es $M_3 = (0\ 1\ 1\ 0\ 0\ 0\ 1)^T$.

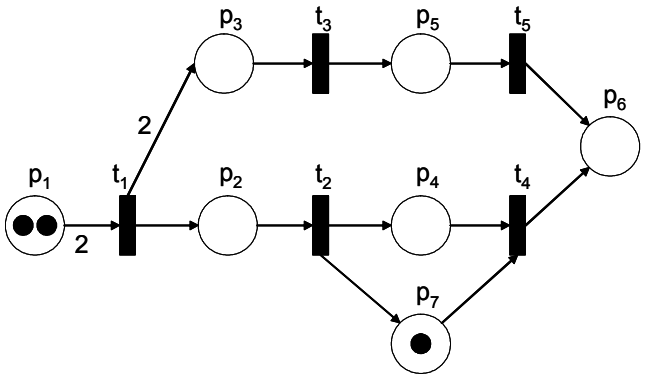


Figura 4.1: Una PN sencilla

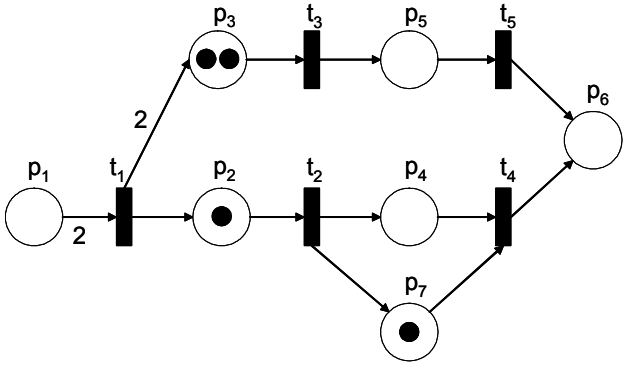


Figura 4.2: La PN después del disparo de la transición t_1

Las PN's no sólo cuentan con el modelo gráfico sino con métodos de análisis que permiten estudiar propiedades importantes de los sistemas que representan. Los métodos para analizar PN pueden clasificarse en los siguientes grupos: 1) el método de árbol de cobertura, 2) la ecuación de estado, que utiliza la matriz de incidencia, 3) la técnica de simplificación y 4) la simulación del comportamiento de la PN. El primer método tiene que ver con todas las marcas alcanzables a partir de la marca inicial. Este método se puede aplicar a todas las clases de redes, pero está limitado a redes "pequeñas" debido a la complejidad en el incremento del espacio de estados. El enfoque de ecuación de estados y la técnica de simplificación de PN son muy poderosos, pero en muchos casos se puede aplicar sólo a subclases especiales de PN o en situaciones especiales. Para modelos de PN complejos, la simulación de eventos discretos es otra forma con la que se puede revisar las propiedades del sistema.

4.2. Red de Petri Coloreada Condicional

La Red de Petri Coloreada Condicional (CCPN, por sus siglas en inglés *Conditional Colored Petri Net*) es una extensión de las PN diseñada específicamente para modelar reglas ECA y su interacción.

4.2.1. Definición de la CCPN

La CCPN hereda los atributos y la regla de disparo de las transiciones de PN, así como la definición de tipos de datos, asignación de colores (valores) a los tokens y la asignación de tipos de datos a los lugares, conceptos presentes en la red de Petri coloreada (CPN). La siguiente notación es útil en la definición de CCPN.

Los elementos de un tipo, T. El conjunto de todos los elementos en T se denota usando la misma letra T.

El tipo de una variable, v, se representa por $Tipo\,(v)$

El símbolo $\mathcal{B}$ se utiliza para denotar un tipo de dato booleano, el cual contiene los elementos $\{falso, verdadero\}$ y las operaciones estándar del álgebra booleana.

Para denotar el i-ésimo token almacenado en el lugar p_j se emplea la notación $M\,(p_j)_i$

Definición 4.2 *Un multiconjunto m, sobre un conjunto no vacío S, es una función* $m : S \rightarrow N$ *la*

cual puede representarse como la sumatoria $\sum_{s \in S} m(s)' s$. *Donde* S_{MS} *es el conjunto de todos los multiconjuntos sobre S. Los enteros no negativos* $\{m(s) \mid s \in S\}$ *son los coeficientes del multiconjunto.* $s \in m$ *si y sólo si* $m(s) \neq 0$.

Formalmente, la CCPN se define de la siguiente manera:

Definición 4.3 *Una Red de Petri Coloreada Condicional (CCPN) es una 11-tupla*

$$CCPN = (\sum, P, T, A, N, C, Con, Acción, D, \tau, I)$$

donde:

1. $\sum$ es un conjunto finito de tipos de datos, también llamados conjuntos de colores.
2. P es un conjunto finito de lugares. Para facilitar la representación gráfica de los lugares, P está dividido en los siguientes subconjuntos:

   $$P = P_{prim} \cup P_{comp} \cup P_{virtual} \cup P_{copia}$$

   donde P_{prim}, P_{comp}, $P_{virtual}$ y P_{copia} representan lugares primitivos, compuestos, virtuales y copia, respectivamente.
3. T es un conjunto finito de transiciones. Existen tres tipos de transiciones:

   $$T = T_{regla} \cup T_{comp} \cup T_{copia}$$

   donde T_{regla}, T_{comp} y T_{copia} representan transiciones tipo regla, compuestas y copia, respectivamente.
4. A es un conjunto finito de arcos, tales que

   $$P \cap T = P \cap A = T \cap A = \emptyset$$

   donde A está dividido en dos subconjuntos, A_{inh} y A_{nor}, los cuales representan los conjuntos de arcos inhibidores ā y arcos normales a, respectivamente.
5. $N : A \rightarrow P \times T \cup T \times P$ es una función nodo.

6. $C : P \rightarrow 2^{\Sigma}$ es una función color.

7. Con es una función para evaluar condiciones. Está definida desde una transición $t \in T_{regla} \cup T_{comp}$ hacia expresiones tales que

   $\forall t \in T_{regla} : [Tipo\,(Con\,(t)) = \mathcal{B}]$,
   donde la función Con evalúa la condición de la regla ECA.

   $\forall t \in T_{comp} : [Tipo\,(Con\,(t)) = \mathcal{B}]$,
   donde la función Con evalúa el intervalo de tiempo, almacenado en t, contra la estampa de tiempo de los tokens.

8. *Acción* es una función sobre la *acción* de la regla ECA. Está definida desde el conjunto T_{regla} hacia expresiones tales que:

$$\forall t_n \in T_{regla}, p \in t_n^{\bullet} : [Tipo\,(Acción\,(t_n)) = C\,(p)_{MS}]$$

9. $D : T_{comp} \rightarrow \mathbb{R} \times \mathbb{R}$ es una función de duración de tiempo. Está definida desde el conjunto T_{comp} hacia intervalos de tiempo $[d1\,(t)\,, d2\,(t)]$, donde $t \in T_{comp}$ y $d1\,(t)\,, d2\,(t)$ son los instantes inicial y final, respectivamente, del intervalo de tiempo.

10. $\tau : M\,(p) \rightarrow \mathbb{R}^{+}$ es una función de intervalos de tiempo. Está definida desde la marca $M\,(p)$ de un lugar p, hacia $\{0\} \cup \mathbb{R}^{+}$, la cual asigna a cada token del lugar p una estampa de tiempo correspondiente a un instante del reloj en la forma $año : mes : día - hora : minuto : segundo$, por ejemplo, un token puede tener una estampa de tiempo como 2005 : 10 : 06 - 16 : 30 : 00.

11. $I : P \rightarrow C\,(p)_{MS}$ es una función de inicialización de valores en la CCPN. Está definida a partir del conjunto P hacia el multiconjunto de colores $C\,(p)_{MS}$.

El conjunto $\sum$ determina los tipos de datos que se utilizarán dentro de la CCPN para manipular la información relacionada con los eventos, la evaluación de expresiones booleanas en la parte condicional y los datos necesarios para llevar a cabo la acción de las reglas ECA.

El conjunto de lugares $P_{prim} \subseteq P$ representa los eventos primitivos de las reglas ECA. De forma gráfica se representan con un círculo dibujado con una línea continua, tal como se muestra en la

figura 4.3(a). Este tipo de lugares pueden ser lugares de entrada a cualquier tipo de transición, es decir, $\forall p \in P_{prim}, [p \in {}^{\bullet}t, t \in T]$, pero sólo pueden ser lugares de salida de transiciones tipo T_{regla}, es decir, $\forall p \in P_{prim}, [p \in t^{\bullet}, t \in T_{regla}]$, ya que representan operaciones primitivas que describen la acción de la regla. El conjunto $P_{comp} \subseteq P$ se usa para representar eventos compuestos, los cuales manejan estampas de tiempo. Gráficamente se representan con dos círculos concéntricos dibujados con líneas continuas, como se muestra en la figura 4.3(b). Estos lugares pueden ser la entrada de cualquier tipo de transiciones, es decir, $\forall p \in P_{comp}, [p \in {}^{\bullet}t, t \in T]$, y sólo pueden ser lugares de salida de transiciones tipo T_{comp}, esto es, $\forall p \in P_{prim}, [p \in t^{\bullet}, t \in T_{comp}]$. Los lugares del conjunto $P_{virtual} \in P$ se utilizan para almacenar tokens durante la formación de eventos compuestos. La representación gráfica de estos lugares se muestra en la figura 4.3(c).

El conjunto $P_{copia} \subseteq P$ se utiliza para duplicar eventos cuando estos disparan dos o más reglas ya que cada una debe procesarse de manera independiente. Cada lugar del tipo P_{copia} se representa gráficamente por dos círculos concéntricos, el círculo externo de dibuja con una línea continua mientras que el círculo interno se dibuja con una línea punteada, como se muestra en la figura 4.3(d). Los lugares $p \in P_{copia}$ son lugares de entrada para las transiciones tipo T_{regla} y T_{comp}, es decir, $\forall p \in P_{copia} [p \in {}^{\bullet}t, t \in \{T_{regla} \cup T_{comp}\}]$; sin embargo, son únicamente lugares de salida para las transiciones tipo T_{copia}, es decir, $\forall p \in P_{copia}, [p \in t^{\bullet}, t \in T_{copia}]$.

El conjunto de transiciones T está dividido en los conjuntos T_{regla},T_{comp} y T_{copia}.

Una transición $t \in T_{regla}$ almacena la condición de una regla ECA. El lugar de entrada de esta transición representa el evento que activa la regla mientras que su lugar de salida representa la acción de la regla. Cabe mencionar que esta transición puede tener varios lugares de salida (primitivos) dependiendo del número de acciones que ejecute la regla ECA. De manera gráfica, este tipo de transiciones se muestran con un rectángulo, tal como se presenta en la figura 4.3(e).

Una transición $t \in T_{comp}$ utiliza sus lugares de entrada para formar las estructuras de los eventos compuestos. Las transiciones de este tipo se representan gráficamente mediante una doble barra, como lo muestra la figura 4.3(f).

Una transición $t \in T_{copia}$ se usa para duplicar su lugar de entrada. Gráficamente se dibuja como una línea horizontal, tal como se muestra en la figura 4.3(g).

El conjunto de arcos A de la CCPN está dividido en los conjuntos A_{inh} y A_{nor}. El conjunto A_{nor} está formado por los arcos normales de teoría de PN, los cuales conectan lugares y transiciones (y

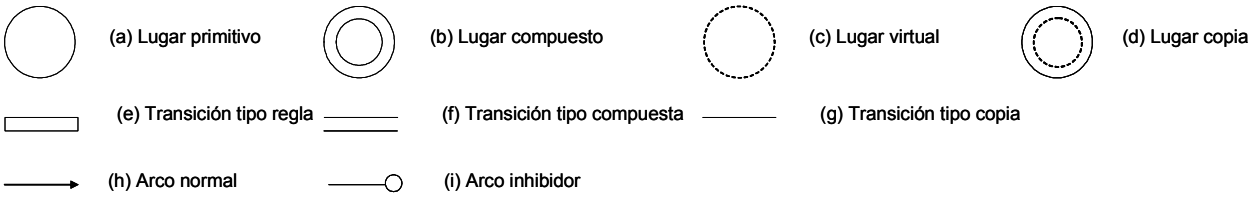


Figura 4.3: Elementos gráficos de la CCPN

viceversa) siempre que no se especifique el evento compuesto negación. Los arcos A_{nor} se representan con una flecha, como se muestra en la figura 4.3(h). El conjunto de arcos A_{inh} está formado por los arcos inhibidores y se utiliza para formar el evento compuesto negación. En la teoría de PN, un arco inhibidor habilita una transición siempre que en su lugar de entrada no se encuentre un token. La figura 4.3(i) muestra la manera en que se dibuja este tipo de arcos.

La función nodo N mapea cada arco de la CCPN en pares ordenados de números enteros, los cuales indican los índices de las transiciones y los lugares que están conectados mediante un arco.

La función *Color* mapea cada lugar $p \in P$ con un tipo de color $C(p)$, esto es, los datos almacenados dentro de un lugar pertenecen a un tipo de color.

Cuando una función de condición Con evalúa la condición de la regla ECA, mapea cada transición $t \in T_{regla}$ a una expresión booleana donde todas las variables tienen un tipo de dato que pertenece a $\sum$. Si la evaluación de Con es verdadera, se produce el disparo de t. Por otro lado, la función Con verifica si la estampa de tiempo de los tokens provenientes de los lugares de entrada de una transición $t \in T_{comp}$ cumplen con el intervalo de tiempo de t.

La función *Acción* mapea cada transición $t \in T_{regla}$ hacia un conjunto de valores de algún tipo de datos $C(p)$, los cuales serán depositados en su correspondiente lugar de salida.

El intervalo de tiempo D es utilizado por la función Con para evaluar transiciones $t \in T_{comp}$.

Los tokens almacenados en un lugar p pueden tener diferentes estampas de tiempo, por lo tanto, se usa la función τ para cada token $M(p_i)$.

La función de inicialización I mapea cada lugar p hacia expresiones que deben de ser de tipo $C(p)_{MS}$.

Además de la representación gráfica, la CCPN cuenta con una representación matemática dada por la matriz de conexión, la cual es similar a la definición de matriz de incidencia de la teoría de

PN tradicional.. La matriz de conexión para la CCPN se define de la siguiente manera:

Definición 4.4 *Matriz de conexión de una CCPN. Para una CCPN N, con n transiciones y m lugares, la matriz de conexión* $A = [\ a_{ij}\ ]$ *es una matriz de números enteros de* $n \times m$. *El valor para cada elemento de la matriz está dado por:*

1. $-w(j,i)$ *Cuando el lugar* $p_j \in P$ *es un lugar de entrada a la transición* $t_i \in T$. $w(j,i)$ *es el peso del arco que conecta a la transición* t_i *con su lugar de entrada* $p_j \in P$.
2. 0 *Cuando no existe un arco que conecta al lugar* $p_j \in P$ *con la transición* $t_i \in T$ *y viceversa.*
3. $w(i,j)$ *Cuando el lugar* $p_j \in P$ *es un lugar de salida a la transición* $t_i \in T$. $w(i,j)$ *es el peso del arco que conecta a la transición* t_i *con su lugar de salida* $p_j \in P$.

En la matriz de conexión de la CCPN todos los arcos se tratan de la misma forma ya que sólo se representa la unión entre un lugar y una transición (y viceversa) pero no el tipo de esta, es decir, la información del tipo de arco (normal o inhibidor) no se considera.

4.3. Modelado de reglas activas con CCPN

De manera general, una regla activa r_i se modela como una transición $t \in T_{regla}$ la cual almacena la condición, c_i, de r_i y cuyos lugares de entrada y salida representan el evento, e_i, y la acción, a_i, de la regla r_i, respectivamente. De acuerdo a discusiones previas, $\forall t \in T_{regla}, {}^\bullet t \in P$ y $t^\bullet \in P_{prim}$, por lo tanto, en su forma más básica, el evento de la condición es primitivo y la estructura generada en la CCPN para esta regla se muestra en la figura 4.4.

Cuando un mismo evento dispara a dos o más reglas, este tiene que ser duplicado por medio de la estructura presentada en la figura 4.5(a). Los lugares de salida de la transición tipo copia contienen la misma información que el lugar de entrada de la misma, es decir, $C(t^\bullet) = C({}^\bullet t), \forall t \in T_{copia}$. La formación de eventos compuestos también está considerada en la CCPN usando la estructura mostrada en la figura 4.5(b). Los lugares de entrada de una transición compuesta representan los eventos necesarios para formar un evento compuesto, mientras que su lugar de salida representa el evento compuesto completo. Finalmente, usamos la estructura mostrada en la figura 4.5(c) para modelar el evento compuesto OR. Los lugares virtuales actúan como un "almacén de eventos",

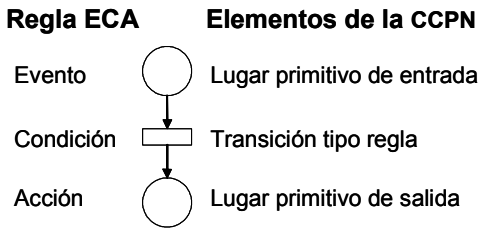


Figura 4.4: Estructura de CCPN básica

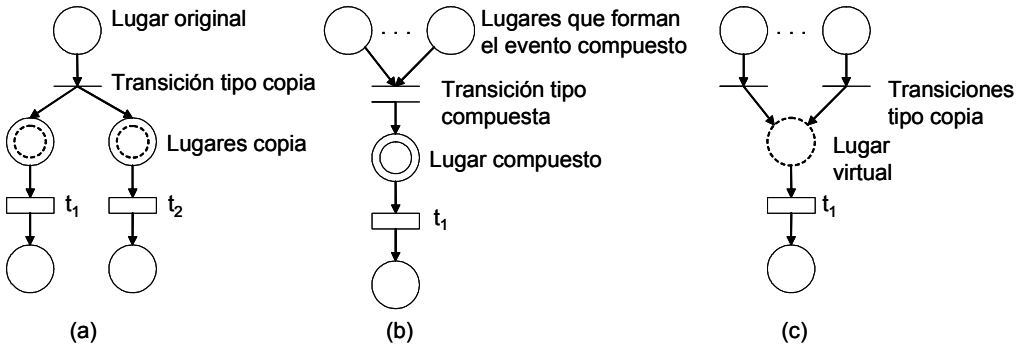


Figura 4.5: Diferentes estructuras de la CCPN para modelar reglas ECA. (a) Estructura copia. (b) Estructura compuesta. (c) Estructura virtual.

es decir, cuando una regla es disparada por diferentes eventos, el lugar virtual los acumula y posteriormente los usa para disparar la regla.

Las transiciones T_{regla} pueden almacenar condiciones sin necesidad de estructuras especiales.

La base de reglas completa se forma conectando los lugares que son tanto de salida como de entrada, es decir, que representan la acción de una regla que, a su vez, es el evento que dispara a otra regla.

4.3.1. Algoritmo de conversión de reglas activas a CCPN

Las reglas ECA se modelan en la CCPN considerando las siguientes características:

1. Los eventos y acciones se modelan como lugares de entrada y salida, respectivamente.

2. Las reglas ECA en sí mismas se mapean en transiciones tipo regla.
3. El disparo de la regla corresponde al disparo de la transición.
4. La detección de eventos se modela como tokens iniciales.
5. Las condiciones se modelan como condiciones agregadas a las transiciones.

El algoritmo original de conversión de reglas ECA a CCPN se prensentó en la referencia [40]. Sin embargo, en ese algoritmo se permitía la generación de autociclos, es decir, que un lugar de entrada a una transición fuera también su lugar de salida. Esta situación dificulta el análisis de la CCPN, por esta razón modificamos el algoritmo para modelar de una forma diferente los autociclos. El algoritmo modificado para representar reglas ECA con CCPN se presenta en la figura 4.6. El segmento con línea punteada muestra la parte que en este trabajo se adicionó al algoritmo original.

De manera general el algoritmo funciona de la siguiente forma: primero se llama al módulo "Obtener esquema conceptual BD", el cual lee el archivo de texto `reglas.eca` para obtener la definición de las relaciones que conforman la BD. Posteriormente se crean los lugares para cada uno de los eventos mediante el módulo "Crear CCPN para eventos" y se almacenan en el conjunto P. Para cada regla de la base de reglas se crea un transición $t \in T_{regla}$ y se le asigna la condición de la regla. Se buscan en el conjunto P los lugares p_1 y p_2 que representan al evento y la acción de la regla, respectivamente. Si p_2 no existe, se crea un lugar $p \in P_{prim}$, se le asigna la información de la acción y se añade al conjunto P. Si ya existe, se verifica que no forme un autociclo con p_1, es decir, ambos lugares tienen que ser diferentes. Si esto es así, se crean los arcos correspondientes para unir p_1 con t y t con p_2. Se agregan todos los elementos creados a sus respectivos conjuntos y se continúa procesando la siguiente regla. Cuando el evento y la acción de la regla están representados por el mismo lugar, en la CCPN se forman un autociclo, es decir, el lugar de entrada a una transición es, a la vez, su lugar de salida. Esta situación dificulta el análisis de la CCPN, de manera que para "romper" un autociclo se introduce una transición $t_c \in T_{copia}$ y un lugar $p_c \in P_{copia}$. El lugar p_1 se conecta con t_c, t_c con p_c, p_c con t (la transición que contiene (y representa) la condición de la regla) y t con p_1. Como se puede observar, el ciclo que aparece en las reglas activas se conserva en la CCPN pero ya no es un autociclo. El comportamiento de la regla no se altera al introducir t_c y p_c ya que este último, al ser un lugar copia, contiene exactamente la misma información que p_1. Cuando se construye la CCPN también se tiene que considerar cuando más de una regla se disparada por

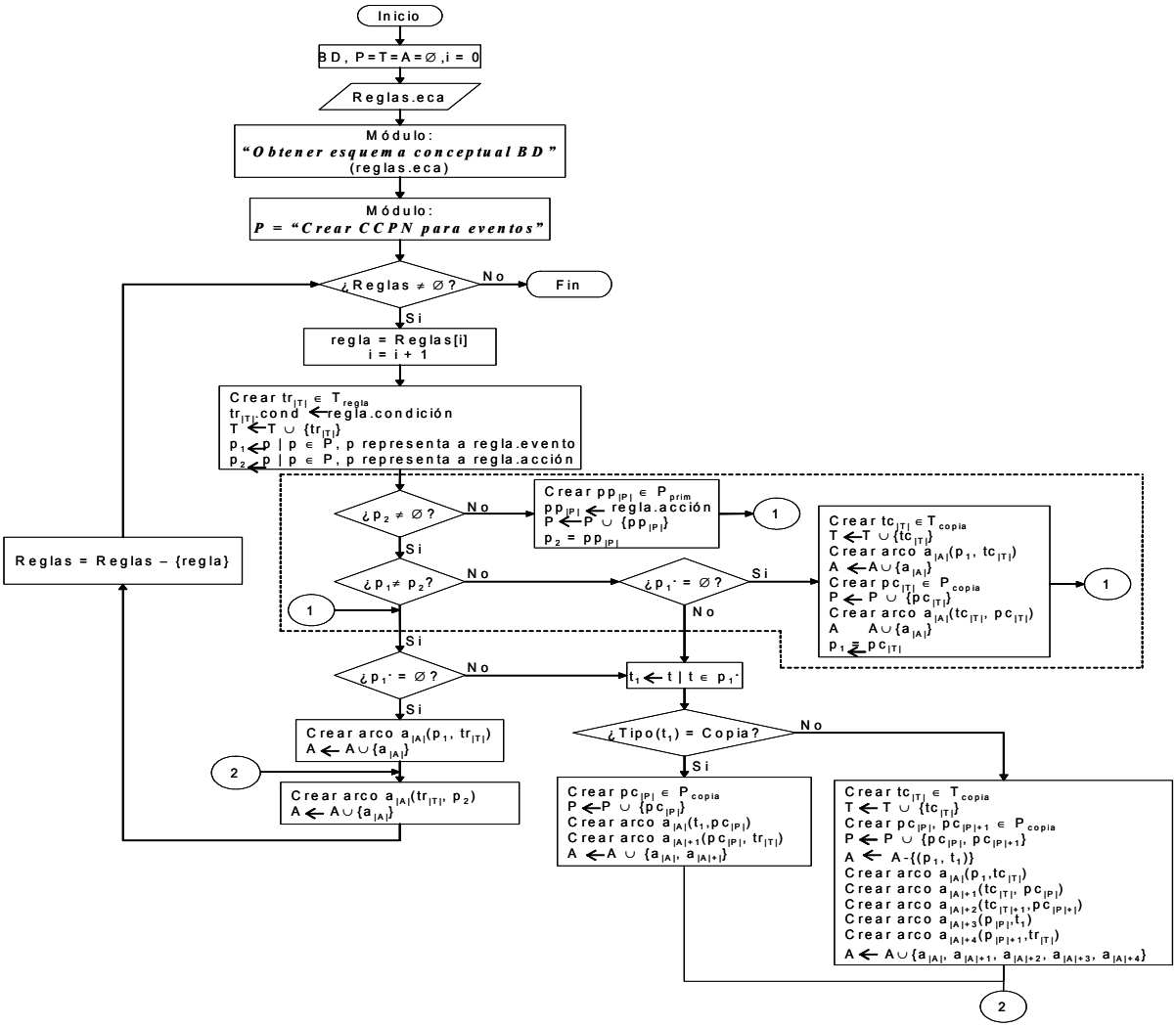


Figura 4.6: Diagrama de flujo para la conversión de reglas ECA en una estructura de CCPN

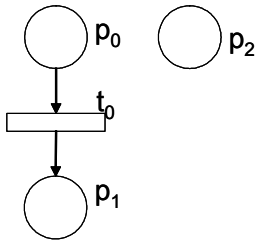


Figura 4.7: CCPN que representa la **Regla 1** del ejemplo 3.1. El lugar p_2 representa el evento update `account`.*balance*

el mismo evento. Si esto sucede, se procede de manera similar introduciendo transiciones y lugares tipo copia.

Para clarificar aún más las características del algoritmo utilizaremos el siguiente ejemplo.

Ejemplo 4.2 *CCPN para la base de reglas descrita en el ejemplo 3.1.*

El primer paso del algoritmo consiste en leer el archivo de texto que contiene la base de reglas para obtener el esquema conceptual de la BD. Este procedimiento lee la relación `account` y sus respectivos atributos.

En el siguiente paso se crean los lugares primitivos p_0, p_1 y p_2 para representar los eventos insert `account`, update `account`.*rate* y update `account`.*balance*, respectivamente. Posteriormente, se crea la transición $t_0 \in T_{regla}$ para representar a la **Regla 1**. Se busca en el conjunto P los lugares p_1 y p_2 que representen el evento y la acción de la regla, respectivamente. Como ambos lugares existen, $p_1 = p_0$ y $p_2 = p_1$, y son diferentes, sólo se crean los arcos $(p_1(p_0), t_0)$ y $(t_0, p_2(p_1))$. La figura 4.7 muestra la estructura generada.

Cuando se procesa la siguiente regla, se crea la transición $t_1 \in T_{regla}$ para representar la **Regla 2**. En este caso, como el evento y la acción de la regla coinciden, los lugares p_1 y p_2 son el mismo (p_1) y todavía no tiene transiciones de salida, como se muestra en la figura 4.7. Por lo tanto, se crea la transición $t_2 \in T_{copia}$, el lugar $p_3 \in P_{copia}$ y los arcos $(p_1(p_1), t_2)$ y (t_2, p_3) para romper el autociclo. Se iguala p_1 con p_3 para indicar que ese es el nuevo lugar de entrada. Se verifica si p_1 tiene transiciones de salida, como no es así, se crean los arcos $(p_1(p_3), t_1)$ y $(t_1, p_2(p_1))$. De esta manera el autociclo ya no aparece.

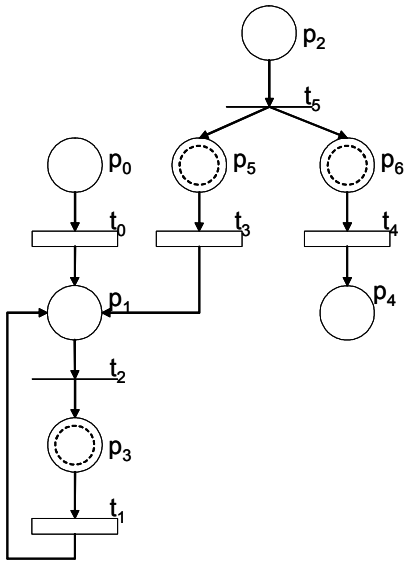


Figura 4.8: CCPN que representa la base de reglas del ejemplo 3.1

El procesamiento de la **Regla 3** es similar al de la **Regla 1** puesto que los lugares que representan el evento y la acción de la regla ya existen y son diferentes, sólo se crea la transición $t_3 \in T_{regla}$ y los arcos (p_2, t_3) y (t_3, p_1).

Finalmente, para representar la **Regla 4** se crea la transición $t_4 \in T_{regla}$. El lugar p_1 que representa el evento de la regla ya existe (p_2), no así el lugar p_2 que representa la acción de la misma. Por lo tanto, se crea este el lugar $p_4 \in T_{prim}$. Se verifican las transiciones de salida del lugar p_1, en este caso, ya tiene una transición de salida que es t_3. Entonces se ejecutan los siguientes pasos: se crea la transición $t_5 \in T_{copia}$, los lugares $p_5, p_6 \in P_{copia}$, los arcos (t_5, p_5), (t_5, p_6), (p_2, t_5), (p_5, t_3), (p_6, t_4) y se elimina el arco (p_2, t_3). Finalmente, se crea el arco (t_4, p_4). Como ya no hay más reglas, el algoritmo termina. La CCPN completa se muestra en la figura 4.8.

Una descripción de todos los lugares de la CCPN de la figura 4.8 se muestra en la tabla 2.1.

Tabla 2.1. Correspondencia entre eventos/acciones y lugares

Lugar	**Tipo**	**Evento/Acción**
p_0	Primitivo	insert `account`
p_1	Primitivo	update `account`.*rate*
p_2	Primitivo	update `account`.*balance*
p_3	Copia	copia de p_1
p_4	Primitivo	delete `account`
p_5	Copia	copia de p_2
p_6	Copia	copia de p_2

La descripción de las transiciones de la CCPN de la figura 4.8 se muestra en la tabla 2.2.

Tabla 2.2. Correspondencia entre reglas y transiciones

Transición	**Tipo**	**Regla**	**Condición**
t_0	Regla	**Regla 1**	insert.*balance* $<$ 500 and insert.*rate* $>$ 0.0
t_1	Regla	**Regla 2**	update.*balance* $<$ 500 and update.*rate* $>$ 0.0
t_2	Copia	**-**	verdadero
t_3	Regla	**Regla 3**	update.*rate* $>$ 1 and update.*rate* $<$ 3
t_4	Regla	**Regla 4**	update.*rate* $>$ 1 and update.*rate* $<$ 3
t_5	Copia	-	verdadero

La matriz de conexión de la CCPN de la figura 4.8 se muestra a continuación.

		L	**U**	**G**	**A**	**R**	**E**	**S**
		0	1	2	3	4	5	6
T	0	-1	1	0	0	0	0	0
R	1	0	1	0	-1	0	0	0
A	2	0	-1	0	1	0	0	0
N	3	0	1	0	0	0	-1	0
S	4	0	0	0	0	1	0	-1
.	5	0	0	-1	0	0	1	1

Como se puede observar, existe una correspondencia entre los elementos de la CCPN y las entradas de la matriz de conexión. Por ejemplo, para analizar la transición t_0, se debe tomar la fila

0 de la matriz de conexión. La entrada negativa en la columna 0 indica que esa columna representa el lugar de entrada, p_0, de la transición t_0. Por otro lado, la entrada positiva en la columna 1 representa el lugar de salida, p_1, de la transición t_0. Esta información corresponde con la CCPN de la figura 4.8.

4.4. Simulación de reglas activas con CCPN

La simulación de reglas ECA con CCPN se lleva a cabo con los tokens ya que estos rigen el comportamiento de la PN. En la CCPN los datos se almacenan en los tokens. Un elemento token de la CCPN se define de la siguiente manera:

Definición 4.5 *En la CCPN, un elemento token es una 3-tupla* $(p, c, estampa)$ *donde* $p \in P$, $c \in C(p)$ *y estampa indica el punto en el tiempo en que ocurre el evento correspondiente y el token es depositado en p. El conjunto de todos los elementos token está denotado por TE. Una marca M es un multi-conjunto sobre TE. La marca inicial* M_0 *se obtiene al evaluar las expresiones de inicialización:*

$$\forall (p, c, estampa) \in TE : M_0(p, c, estampa) = I(p)$$

El conjunto de todas las marcas es expresado por $R(M)$.

Para expresar los diferentes tipos de colores que se encuentran en p, usamos la función $N_{color}(p)$. Esta función tendrá valores mayores a 1 en los lugares $p \in P_{virtual}$ debido a que los tokens son suministrados por los lugares que les preceden. Si $p \in P_{prim} \cup P_{comp} \cup P_{copia}$ entonces los tokens almacenados en p tendrán el mismo tipo de color.

Una transición $t \in T_{copia}$ se activa en una marca M si y sólo si:

$$\forall p \in {}^\bullet t : M(p) > 0$$

Una transición $t \in T_{regla}$ se activa en una marca M si y sólo si se satisface:

$$\forall p \in {}^\bullet t : M(p) > 0 \; y \; Tipo(Cond(t)) = verdadero$$

Definición 4.6 *Una transición* $t \in T$ *se habilita en una marca M si y sólo si:*

1)$\forall p \in {}^{\bullet}t :\mid M(p) \mid < w(p,t), t \in T_{comp}, Tipo(t) = Negación$
2)$\forall p \in {}^{\bullet}t :\mid M(p) \mid \geq w(p,t)$, en cualquier otro caso
donde $w(p,t)$ indica el peso del arco que conecta p con t.

Definición 4.7 *Cuando una transición $t \in T$ se habilita, la función de habilitación $C_{habilitada}$ es definida desde $P \times T$ hacia expresiones tales que:*

$$\forall t \in T, p \in {}^{\bullet}t : [Tipo(C_{habilitada}(p,t)) = C(p)_{MS}]$$

Cuando una transición t es habilitada, la función de habilitación $C_{habilitada}$ es definida para especificar los tokens que provocaron la habilitación de t. En la CCPN, una transición $t \in T_{copia}$ se dispara siempre que esté habilitada; sin embargo, una transición $t \in T_{comp} \cup T_{regla}$ se dispara de forma condicionada. Una transición $t_i \in T_{comp}$ se dispara si está habilitada y si la condición del intervalo de tiempo se cumple. Una transición $t_j \in T_{regla}$ se dispara si está habilitada y si la evaluación de la condición de la regla ECA almacenada en t_j contra el estado que guarde la BD resulta verdadera.

Definición 4.8 *Cuando una transición $t \in T_{comp}$ se habilita, se define una función de composición de eventos $C_{composición}$, que va de $T \times P$ hacia expresiones tales que:*

$$Tipo(C_{composición}(t,p_0)) = Tipo(t)(C(p_k)_{MS}) \text{ donde } p_k \in {}^{\bullet}t \text{ y } p_0 \in t^{\bullet}$$

Dada la existencia de tres tipos de transiciones en la CCPN, existen tres casos para la ejecución de las reglas ECA.

Definición 4.9 *Una transición $t \in T$ se dispara si y sólo si:*

i) $\forall t \in T_{regla}$, t está habilitada y $Tipo(Con(t)) = verdadero$,
ii) $\forall t \in T_{copia}$, t está habilitada y
iii) $\forall t \in T_{comp}$, t está habilitada y $\forall p \in {}^{\bullet}t$:$D(t) = [d_1(t), d_2(t)] : [d_1(t) \leq \tau(M(p)) \leq d_2(t)]$, y cumple la condición de cada evento compuesto.

Los tokens en la CCPN también se desplazan de acuerdo a ciertas reglas descritas a continuación.

Definición 4.10 *(Desplazamiento de tokens 1) Cuando una transición $t \in T$ está habilitada en una marca M_1 y es disparada, el estado de la marca M_1 cambia a una marca M_2, definida por:*

$(i) Sit \in T_{regla}, \forall p \in P$

$M_2(p) = M_1(p) - C_{habilitada}(p,t) + Acción(t,p)$

(ii) Si $t \in T_{copia}, \forall p_1 \in {}^\bullet t, p_2 \in t^\bullet$:

$M_2(p_1) = M_1(p_1) - C_{habilitada}(p_1,t)$

$M_2(p_2) = M_1(p_2) + C_{habilitada}(p_1,t)$

(iii) Si $t \in T_{copia}, \forall p_1 \in {}^\bullet t, p_2 \in t^\bullet$:

a) Si $Tipo(t) = Negación$,

$M_2(p_1) = M_1(p_1)$

$M_2(p_2) = M_1(p_2) + C_{composición}(t,p_2)$

b) en caso contrario

$M_2(p_1) = M_1(p_1) - C_{enabled}(p_1,t)$

$M_2(p_2) = M_1(p_2) + C_{composition}(t,p_2)$

Si una transición t se habilita pero no se dispara, el token que la habilitó se desecha de la CCPN. Las transiciones que pueden habilitarse y no dispararse son las $t \in T_{regla} \cup T_{comp}$ ya que la evaluación de la condición de la regla almacenada en una $t \in T_{regla}$ puede resultar falsa. Por otro lado, una transición $t \in T_{comp}$ se habilita ante la presencia de los eventos que forman al evento compuesto, pero si no cumplen con el intervalo de tiempo D y la condición adicional que estipula el propio evento compuesto, entonces t no se dispara. Una excepción de las transiciones $t \in T_{comp}$ es cuando $Tipo(t) = Negación$ ya que cuando t se habilita significa que el intervalo D ha terminado y no ocurrió el evento representado por $p, \{p\} = {}^\bullet t$, entonces se dispara t. Por lo tanto, el desplazamiento de tokens en caso que una transición $t \in T_{regla} \cup (T_{comp} - \{t \mid Tipo(t) = Negación\})$ no se dispare es el siguiente:

Definición 4.11 *(Desplazamiento de tokens 2) Cuando una transición* $t \in T_{regla} \cup (T_{comp} - \{t \mid Tipo(t) = Negación)\}$ *está habilitada en una marca* M_1 *pero NO es disparada,* $Tipo(Con(t)) = falso$*, y además, para el caso de que* $t \in T_{comp}$*, la estampa de tiempo del token cae fuera del intervalo* D*, el estado de la marca* M_1 *cambia a una marca* M_2*, definida por:*

$$\forall p \in P : M_2(p) = M_1(p) - C_{habilitada}(p,t)$$

La expresión $M_1[t \succ M_2$ denota que M_2 puede alcanzarse directamente a partir de M_1, disparando $t \in T$.

Ejemplo 4.3 *Simulación de la ejecución de la CCPN de la figura 4.8*

En la CCPN el modo de acoplamiento de reglas que se considera por omisión es el inmediato, así que la simulación de la ejecución se realizará bajo este modo. Además, consideraremos el estado inicial de la BD mostrado en la figura 3.2.

Consideremos que en la CCPN existe un token en el lugar p_2 con la información "update `account` set *balance* = 400 where update.*rate* > 1 and update.*rate* < 3" (figura 4.9(a)). La transición t_5 está habilitada y, dado que es una transición tipo copia, esto es suficiente para que se dispare. Entonces dos tokens exactamente iguales al originalmente depositado en el lugar p_2 se colocan en los lugares p_5 y p_6, respectivamente (figura 4.9(b)). Las transiciones t_3 y t_4 evalúan la misma condición y supongamos que el resultado es verdadero. Por comodidad, asumiremos que primero se ejecuta la "acción" de la transición t_3 y posteriormente la de t_4. Bajo estas suposiciones, se coloca un token con la información "update `account` set *rate* = 2.0 where *rate* > 1 and *rate* < 3.0" en el lugar p_1 (figura 4.9(c)). Nuevamente, la transición t_2 está habilitada y esto es suficiente para que se dispare, originando la eliminación del token del lugar p_2 y la colocación del token correspondiente en el lugar p_3. La evaluación de la condición almacenada en la transición $t_1$es verdadera y se coloca de nuevo un token en p_2 con la información "update `account` set *rate* = 0.0 where *balance* < 500 and *rate* > 0.0". El proceso se repite y en esta ocasión la evaluación de la condición almacenada en t_1 es falsa, entonces el token se desecha. Finalmente, se ejecuta la "acción" de la transición t_4 lo que causa que se elimine el token del lugar p_6 y se deposite un token en el lugar p_4. Como este lugar no tiene ninguna transición de salida, el proceso de ejecución termina (figura 4.9(d)). El movimiento de tokens puede verse en la figura 4.9.

4.5. Comentarios finales

Existen diferentes modelos para representar reglas ECA y su interacción, por ejemplo, los grafos de disparo, activación, y evolución. Sin embargo, todos ellos consideran cada regla como una unidad indivisible, lo cual para ciertos propósitos puede ser práctico pero para otros puede esconder detalles importantes.

La CCPN es una extensión de las PN's que modela cada elemento de la regla por separado, proporciona estructuras para considerar la formación de diversos eventos compuestos, muestra

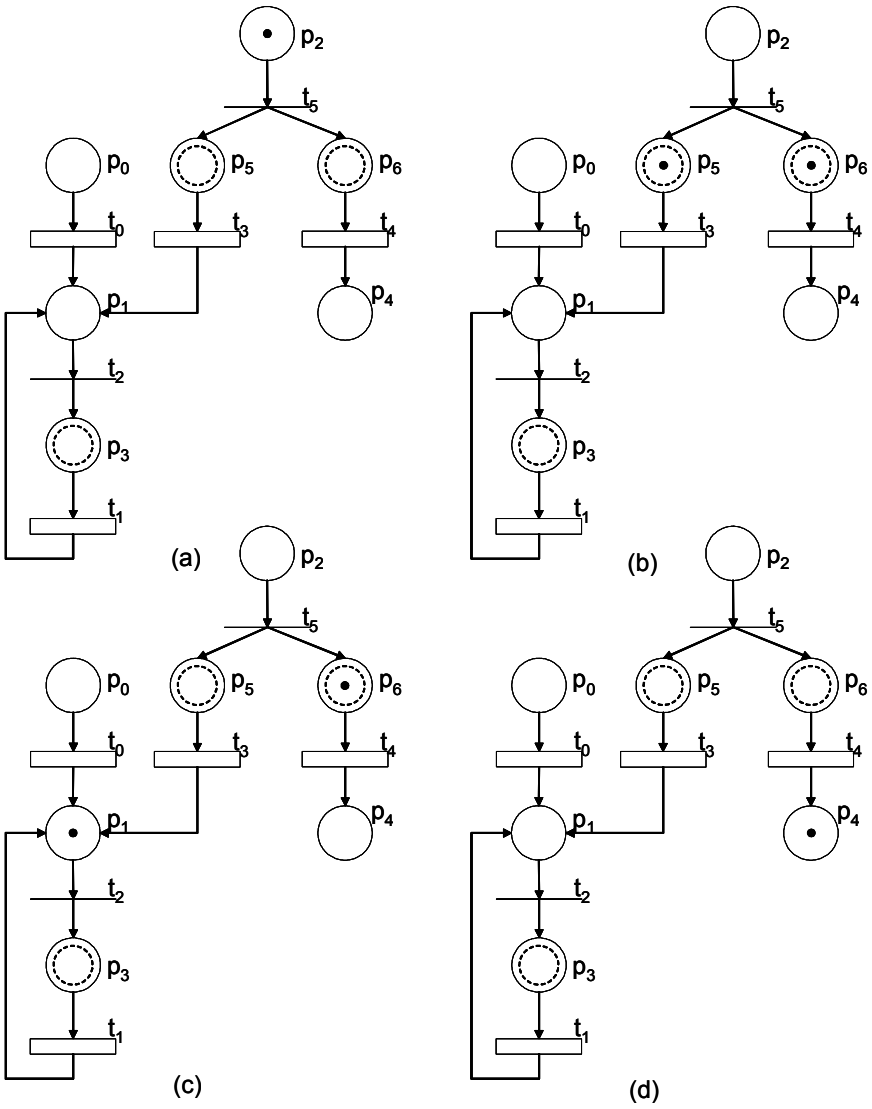


Figura 4.9: Movimiento de tokens durante la simulación de la CCPN. (a) El estado inicial de la CCPN. (b) El estado de la CCPN después de disparar t_5. (c) El estado de la CCPN después de disparar t_3. (d) El estado final de la CCPN después de disparar dos veces t_1 y una vez t_4.

gráficamente la interacción de las reglas y permite simular el proceso de ejecución de las mismas. Asimismo, provee herramientas de análisis formales que permiten revisar propiedades importantes de los sistemas que se están modelando.

Capítulo 5

Errores Estructurales en Base de Reglas Activas

Las reglas activas, al igual que las reglas de producción, pueden tener errores de redundancia, inconsistencia, incompletitud y circularidad. Sin embargo, al considerar el elemento extra en las reglas activas (evento) los errores pueden aparecer en formas más complicadas. Por lo tanto, es necesario redefinir las nociones de errores consideradas en las reglas de producción en el contexto de reglas activas, así como desarrollar métodos de verificación.

En este capítulo se describen las definiciones desarrolladas, tomando en cuenta todos los elementos de las reglas ECA, para cada uno de los errores anteriores. Por simplicidad, se considera que el evento (condición) que dispara a (evalúa) cada regla es una conjunción de uno o más eventos (condiciones) y que su acción lleva a cabo sólo una tarea. Esto no limita el análisis ya que sólo se usan reglas disparadas por los eventos `AND`, `OR` y `NOT` y cuya condición es de la forma <*condición 1* `op. lógico` *condición 2* `op. lógico` ... `op. lógico` *condición n*>, donde `op. lógico` = {and, or, not}, y, como se demostrará en capítulos posteriores, este tipo de reglas, al igual que las que ejecutan más de una tarea, pueden transformarse en otras equivalentes que cumplan con la restricción anterior. En este contexto, una regla ECA, r_i, tiene la forma:

`ON AND`(e_1, e_2, ..., e_n)

`IF AND`(c_1, c_2, ..., c_m)

`THEN` a_1

Decimos que $E_i = \{e_1, e_2, ..., e_n\}$, $C_i = \{c_1, c_2, ..., c_m\}$ y $A_i = \{a_i\}$. Nos referiremos como *premisa de la regla* r_i a su evento y condición.

Para ilustrar nuestras definiciones de error usaremos una base de reglas que consiste de todas las reglas del ejemplo 3.1 y una nueva regla, **Regla 5**, idéntica a la **Regla 1**.

Ejemplo 5.1 *Base de reglas activas para ilustrar errores de redundancia, inconsistencia, incompletitud y circularidad.*

Esquema de relación:

`account`(*num, name, balance, rate*), *rate* $\in [0, 10]$

Base de reglas:

Regla 1

`ON` insert `account`

`IF` insert.*balance* < 500 and insert.*rate* > 0.0

`THEN` update `account` set *rate* $= 0.0$

where *balance* < 500 and *rate* > 0.0

Regla 2

`ON` update `account`.*rate*

`IF` update.*balance* < 500 and update.*rate* > 0.0

`THEN` update `account` set *rate* $= 0.0$

where *balance* < 500 and *rate* > 0.0

Regla 3

`ON` update `account`.*balance*

`IF` update.*rate* >1 and update.*rate* < 3.0

`THEN` update `account` set *rate* $= 2.0$

where *rate* > 1 and *rate* < 3.0

Rule 4

`ON` update `account`.*balance*

`IF` update.*rate* >1 and update.*rate* < 3.0

`THEN` delete from `account`

where *rate* > 1 and *rate* < 3.0

Regla 5

`ON` insert `account`

`IF` insert.*balance* < 500 and insert.*rate* > 0.0

`THEN` update `account` set *rate* $= 0.0$

where *balance* < 500 and *rate* > 0.0

5.1. Redundancia

El error de *redundancia* está caracterizado por dos tipos de reglas: *reglas redundantes* y *reglas incluidas*.

Dadas dos reglas r_i y r_j que ejecutan la misma acción, si la premisa de r_i es más restrictiva que la de r_j, entonces r_i está incluida en r_j ya que con menos condiciones se llega a ejecutar la misma acción. Por ejemplo, consideremos las siguientes reglas que actúan sobre el esquema de relación del ejemplo 5.1:

Regla 1

`ON AND`(update `account`.*balance*, update `account`.*rate*)

IF update.*balance* < 500 and update.*rate* < 3

THEN delete from `account` where *balance* < 500 and *rate* < 3

Regla 2

ON update `account`.*balance*

IF update.*balance* < 500

THEN delete from `account` where *balance* < 500 and *rate* < 3

La **Regla 1** está incluida en la **Regla 2**.

Definición 5.1 *Reglas incluidas. Una regla r_i está incluida en la regla r_j si se cumplen las siguientes condiciones:*

1. $E_j \subseteq E_i$
2. $C_j \subseteq C_i$
3. $A_j = A_i$

Cuando en la definición anterior $E_j = E_i$ y $C_j = C_i$ las reglas son *redundantes*.

La **Regla 1** y la **Regla 5** del ejemplo 5.1 son redundantes ya que tienen la misma premisa y ejecutan la misma acción.

Algunas veces las reglas no son completamente incluidas o redundantes, es decir, sólo algunos de sus elementos (evento, condición, acción) están incluidos o son redundantes. Denominamos a este tipo de reglas como *parcialmente incluidas (redundantes)* dado que en algún momento estas pueden causar redundancia.

Definición 5.2 *Reglas parcialmente incluidas. Una regla r_i está parcialmente incluida en la regla r_j si sólo dos de los elementos de r_i están incluidos en los correspondientes elementos de r_j.*

Identificamos tres tipos de reglas parcialmente incluidas los cuales describimos a continuación.

Reglas parcialmente incluidas evento - condición

Las condiciones que deben cumplir este tipo de reglas son las siguientes:

1. $E_j \subseteq E_i$
2. $C_j \subseteq C_i$
3. $A_j \neq A_i$

Reglas parcialmente incluidas evento - acción

Las condiciones que deben cumplir este tipo de reglas son las siguientes:

1. $E_j \subseteq E_i$
2. $C_j \subsetneq C_i$
3. $A_j = A_i$

Reglas parcialmente incluidas condición - acción

Las condiciones que deben cumplir este tipo de reglas son las siguientes:

1. $E_j \subsetneq E_i$
2. $C_j \subseteq C_i$

3. $A_j = A_i$

Cuando en los casos anteriores se cumple la igualdad, se tienen *reglas parcialmente redundantes.*

En el ejemplo 5.1, la **Regla 1** y la **Regla 2** son reglas parcialmente redundantes condición - acción ya que ambas evalúan la misma condición y efectúan la misma acción.

La redundancia no causa un mal desempeño de la base de reglas. Sin embargo, sí dificulta su mantenimiento ya que si se desea remover una regla que está duplicada, no sólo hay que eliminar esta regla sino además todas sus copias.

5.2. Inconsistencia

Para describir el error de inconsistencia necesitamos definir el concepto de *acciones contradictorias.*

Definición 5.3 *Acciones contradictorias. Dos acciones a_i y a_j son contradictorias si se oponen entre sí.*

El diseñador de reglas define las acciones contradictorias. En la base de reglas del ejemplo 5.1 se pudieron haber definido como acciones contradictorias el actualizar el atributo *rate* de la relación `account` (update `account` *rate*) y borrar tuplas de la misma relación (delete from `account`). Incluso, acciones contradictorias pueden ser aquellas que ejecutan la misma acción pero usando valores diferentes ya que pueden originar información inconsistente, por ejemplo, las acciones "update `account` set *rate* = 2 where *rate* > 1 and *rate* < 3" y "update `account` set *rate* = 0 where *rate* > 1 and *rate* < 3".

La inconsistencia se refiere a la existencia de *reglas en conflicto*, es decir, reglas cuyas premisas consisten del mismo evento y condición, pero sus acciones son contradictorias.

Definición 5.4 *Reglas en conflicto Dos reglas r_i y r_j están en conflicto si se cumplen las siguientes condiciones:*

1. $E_j \subseteq E_i$
2. $C_j \subseteq C_i$

3. A_j y A_i son acciones en conflicto.

En la definición anterior cuando $E_j = E_i$ y $C_j = C_i$, entonces las reglas están en conflicto. En otro caso, cuando la regla más restrictiva (r_i) suceda el conflicto también sucederá. En el ejemplo 5.1 definimos como contradictorias las acciones "update `account`.*rate*" y "delete `account`". Entonces, la **Regla 3** y la **Regla 4** están en conflicto ya que bajo la misma premisa, ejecutan acciones contradictorias. Por otro lado, si hubiéramos definido las acciones "update `account`.*rate* = 0.0" y "update `account`.*rate* = 2.0" como contradictorias y la **Regla 1** y la **Regla 3** tuvieran la misma premisa, entonces esas reglas también serían reglas en conflicto.

Algunas veces las reglas pueden estar en conflicto sólo cuando se cumplen ciertas condiciones. Este tipo de reglas las denominamos *reglas en conflicto potencial.*

Definición 5.5 *Reglas en conflicto potencial. Dos reglas r_i y r_j están en conflicto potencial si se cumplen las siguientes condiciones: 1) su premisa difiere en un elemento (evento o condición) y 2) sus acciones son contradictorias.*

Hay dos tipos de reglas en conflicto potencial, los cuales describimos a continuación:

Reglas en conflicto potencial evento - acción

Las condiciones que se tienen que cumplir para tener este tipo de reglas son las siguientes:

1. $E_j \subseteq E_i$
2. $C_j \subsetneq C_i$
3. A_j y A_i son acciones contradictorias

Reglas en conflicto potencial condición - acción

Las condiciones que se tienen que cumplir para tener este tipo de reglas son las siguientes:

1. $E_j \subsetneq E_i$
2. $C_j \subseteq C_i$
3. A_j y A_i son acciones contradictorias

En el primer caso, el conflicto sucederá sólo cuando las condiciones C_i y C_j se evalúen como verdaderas al mismo tiempo. En el último caso, el conflicto ocurrirá cuando la condición más restrictiva se evalúe como verdadera.

Si consideramos que las acciones de la **Regla 1** y la **Regla 3** del ejemplo 5.1 son contradictorias entonces estas reglas estarían en conflicto potencial evento - acción dado que ambas son disparadas por el mismo evento y sólo cuando sus condiciones se evalúan como verdaderas al mismo tiempo sucede el problema.

Las reglas en conflicto potencial causan problemas sólo hasta que ciertas condiciones se cumplen. Por lo tanto, es importante identificar tales situaciones y tomar medidas correctivas a fin de evitar que el conflicto suceda.

5.3. Incompletitud

Una base de reglas está incompleta cuando no tiene la información necesaria para ejecutar acciones importantes para el sistema. Durante el proceso de representación del conocimiento por medio de reglas, tanto el experto como el ingeniero puede dejar, inadvertidamente, huecos en la base de reglas. Asimismo, el ingeniero de conocimiento puede perder la noción del crecimiento de la base de reglas y esta puede llegar a ser inmanejable.

La incompletitud está caracterizada por reglas *aisladas*, de *punto muerto* e *inalcanzables*. Una regla aislada no se dispara por la acción de otra regla, ni su acción es el evento que dispara a ninguna otra.

Definición 5.6 *Regla aislada. Una regla r_i es una regla aislada si se cumplen las siguientes condiciones:*

1. $E_i \notin \bigcup_{j=1}^{n} A_j, i \neq j$
2. $A_i \notin \bigcup_{j=1}^{n} E_j, i \neq j$

Si la base de reglas del ejemplo 5.1 no contara con la **Regla 3** entonces la **Regla 4** sería una regla aislada dado que su evento no coincide con la acción de ninguna regla y su acción no dispara regla alguna.

Definición 5.7 *Regla de punto muerto. Una regla r_i es una regla de punto muerto si se cumple que $A_i \notin \bigcup_{j=1}^{n} E_j, i \neq j$*

La **Regla 4** en el ejemplo 5.1 es de punto muerto ya que su acción no dispara ninguna regla.

Definición 5.8 *Regla inalcanzable. Una regla r_i es una regla inalcanzable si $\exists r_j \in R$ tal que r_j dispara a r_i pero r_j no activa (o desactiva) a r_i.*

La **Regla 2** del ejemplo 5.1 se dispara a sí misma ya que su acción coincide con su evento. Sin embargo, su acción no activa a su condición, por lo tanto, esta regla se vuelve inalcanzable.

5.4. Circularidad

Si en una base de reglas existe un subconjunto de ellas que forma un ciclo, entonces esas reglas son reglas circulares. Sin un manejo adecuado de estas reglas se puede originar un disparo infinito.

Definición 5.9 *Reglas circulares. Si en R existe una secuencia de disparo r_i, r_j, ..., r_p, r_q donde r_i dispara y activa a r_j y así sucesivamente hasta que r_q dispara y activa a r_i, entonces esas reglas son circulares. Un caso especial es cuando r_i se dispara y activa a sí misma.*

En el ejemplo 5.1 la **Regla 2** no es una regla circular porque aunque se dispara a sí misma su acción no activa su condición, de manera que su proceso de ejecución termina.

5.5. Comentarios finales

Los errores introducidos en una base de reglas ECA, al igual que las reglas de producción, pueden ser redundancia, inconsistencia, incompletitud y circularidad, los cuales se originan por diversas causas, por ejemplo, el(los) experto(s) no poseen el conocimiento suficiente acerca del problema o incluso la erudición que tienen es incorrecta y/o el desarrollador de la base de reglas no captura adecuadamente la información dada por el experto. Estas anormalidades pueden llevar a un mal funcionamiento del sistema en donde se introduce el conjunto de reglas si no se identifican a tiempo. Sin embargo, hasta la fecha no existen definiciones precisas, en el contexto de reglas activas, de los errores descritos, sólo existen considerando reglas de producción.

Aunque a primera vista parece simple extender las definiciones de errores desarrolladas en el ámbito de las reglas de producción al dominio de las reglas ECA, el evento de estas propicia que se tenga que tomar en cuenta la interacción entre ellas para la definición de los errores, además de originar la ocurrencia de nuevos subtipos de error.

En este capítulo mostramos la definición de los errores de redundancia, inconsistencia, incompletitud y circularidad considerando cada uno de los elementos de las reglas ECA así como, en los casos que se requiere, la interacción de las mismas. Además, se identificaron subtipos de algunos errores que son útiles para prevenir situaciones anómalas futuras. Por ejemplo, identificando reglas en conflicto potencial podemos determinar las circunstancias bajo las cuales se obtendrá un comportamiento erróneo del sistema. Al reconocer tales situaciones se pueden planear acciones para solucionar el problema si se presenta, es decir, se podrá estar preparado para una eventualidad, o se pueden rediseñar las reglas en conflicto potencial a fin de eliminar esta situación.

Capítulo 6

Verificación de Base de Reglas Activas

Para detectar errores en un sistema de software se utilizan dos técnicas principales: las inspecciones y las pruebas de software. Las inspecciones analizan y comprueban las representaciones del sistema como el documento de requerimientos, los diagramas de diseño y el código fuente del programa. Esta es una técnica estática puesto que no requiere que el sistema se ejecute. Las pruebas necesitan llevar a cabo una implementación del software con los datos de prueba y examinar las salidas de software y su comportamiento operacional para comprobar que se desempeñe conforme a lo establecido. Esta es una técnica dinámica debido a que se llevan a cabo en una representación ejecutable del sistema.

Llevar a cabo pruebas sistemáticas de los programas requiere que se desarrollen, ejecuten y examinen diversas pruebas. Cada ejecución de una prueba tiende a descubrir una sola falla en el programa o, en el mejor de los casos, sólo pocas fallas. Las inspecciones del software no requieren que el programa se ejecute, por lo que se pueden utilizar como técnicas de verificación antes de que este se implemente. Para descubrir errores se utilizan el conocimiento del sistema a desarrollar y la semántica de la representación del sistema. Cada error se considera de forma aislada sin considerar cómo afectará el comportamiento del sistema.

Después de que se descubre un defecto en el software, debe corregirse y reevaluar el sistema. Esto implica volver a hacer la inspección al programa.

En este trabajo de tesis, usamos la técnica de inspección ya que permite detectar errores en bases de reglas activas antes de la puesta en marcha del sistema.

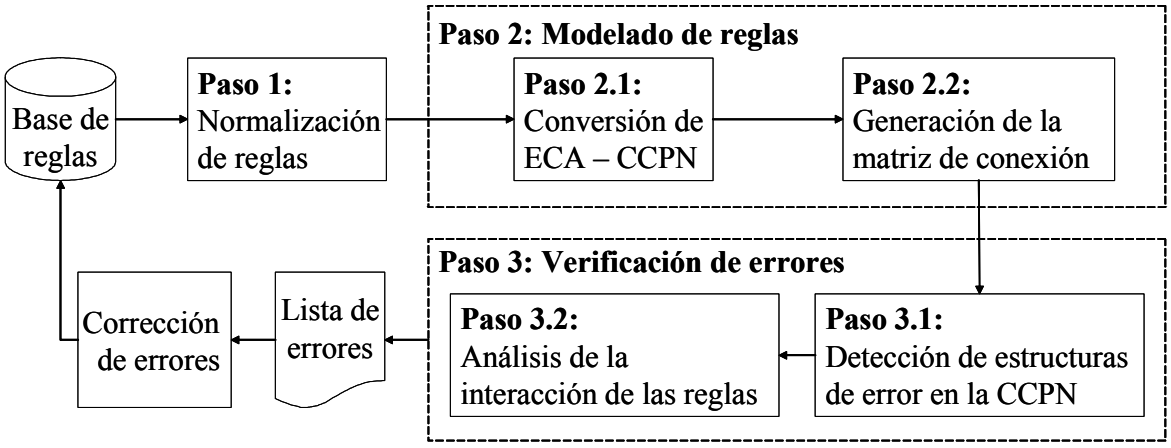


Figura 6.1: Proceso de verificación de reglas basado en la CCPN

Para detectar errores en bases de reglas activas, desarrollamos un método basado en la CCPN que consta de tres partes: *normalización*, *modelado* y *verificación de reglas*. El orden en que se lleva a cabo cada etapa se muestra en la figura 6.1. El resultado al aplicar estos pasos es una lista de los errores identificados en las reglas. Adicionalmente, desarrollamos una etapa de *corrección de errores* que se encarga de proporcionar algunas sugerencias al diseñador de reglas acerca de cómo solucionar los problemas detectados.

Nuestro método toma como entrada una base de reglas activas escritas en la forma general `ON` *evento* `IF` *condición* `THEN` *acción*. El primer paso, *normalización de reglas*, toma la base de reglas original y la transforma en una base de reglas atómicas para simplificar el análisis. En el segundo paso, *modelado de reglas*, la base de reglas atómicas se representa como una estructura de CCPN y se obtiene su matriz de conexión a fin de emplearla en el proceso de detección de errores. El tercer paso, *verificación de errores*, está dividido en dos etapas. La primera de ellas usa la información de la matriz de conexión de la CCPN para identificar las reglas que pueden tener errores. La segunda etapa analiza la interacción de las reglas detectadas y proporciona una conclusión acerca de si existe un error o no. La salida de esta etapa es la lista de los errores encontrados.

Este método nos permite obtener conclusiones acerca de los errores que contiene una base de reglas sin tener que analizarlas todas ya que sólo se enfoca en examinar aquellas que dan indicios de tener algún error.

Al terminar el proceso de detección de errores se sugieren algunas modificaciones usando el procedimiento de *corrección de errores* a fin de obtener una base de reglas libre de estos.

Cada una de las fases que comprenden nuestro método de detección se describe a continuación.

6.1. Normalización de reglas

Las reglas ECA que consideramos tienen las siguientes características:

1. Un evento se puede expresar considerando los operadores descritos en la referencia [1]; sin embargo, limitamos nuestro análisis - sin pérdida de generalidad - a los operadores conjunción (`AND`), disyunción (`OR`) y negación (`NOT`) ya que el resto de los operadores tienen la misma estructura. Por ejemplo, si se respeta el orden de ocurrencia de los eventos en el operador `AND`, entonces se obtiene el operador secuencia.

2. Una condición puede ser simple o compleja. Una *condición simple* tiene la forma: <*variable* **op** *variable*> o <*variable* **op** *constante*>, donde *variable* es un atributo o característica perteneciente al sistema (un atributo de un esquema de relación, por ejemplo), **op** puede ser un elemento del conjunto $\{=, != , >, \geqslant, <, \leqslant\}$ y *constante* es un valor dado. Una *condición compleja* se puede formar conectando varias condiciones simples con los operadores and, or o not, y/o usando expresiones de SQL (Structured Query Language).

3. La acción sólo emplea el operador `AND` ya que cuando una regla se dispara y activa, se deben realizar todas las tareas especificadas en su acción.

Para normalizar las reglas primero las clasificamos de acuerdo a su combinación de operadores en el evento y la condición (premisa). En general, cada regla corresponde a una de las siguientes cuatro categorías:

Reglas tipo 1. *Reglas con el operador de conjunción en ambos elementos de la premisa, i.e.,*

`ON [NOT]AND`$(e_1, e_2, ..., e_n)$

`IF [NOT]AND`$(c_1, c_2, \cdots, c_m)$

`THEN AND`$(a_1, a_2, ..., a_k)$

Reglas tipo 2. *Reglas con el operador de conjunción en el evento y el operador de disyunción en la condición, i.e.,*

`ON [NOT]AND`$(e_1, e_2, ..., e_n)$

`IF [NOT]OR`$(c_1, c_2, \cdots, c_m)$

`THEN AND`$(a_1, a_2, ..., a_k)$

Reglas tipo 3. *Reglas con el operador de disyunción en el evento y el operador de conjunción en la condición, i.e.,*

`ON [NOT]OR`$(e_1, e_2, ..., e_n)$

`IF [NOT]AND`$(c_1, c_2, \cdots, c_m)$

`THEN AND`$(a_1, a_2, ..., a_k)$

Reglas tipo 4. *Reglas con el operador de disyunción en ambas partes de la premisa, i.e.,*

`ON [NOT]OR`$(e_1, e_2, ..., e_n)$

`IF [NOT]OR`$(c_1, c_2, \cdots, c_m)$

`THEN AND`$(a_1, a_2, ..., a_k)$

En las reglas anteriores, eventos de la forma `ON AND`(e_1) u `ON OR`(e_1) son equivalentes a `ON` e_1. El mismo criterio se aplica para las condiciones. El operador `NOT` se encuentra entre corchetes para indicar que se pueden formar nuevas combinaciones de reglas. Por ejemplo, una regla que tiene la estructura:

`ON NOT (AND`(e_1, e_2)`)`

`IF AND`(c_1, c_2)

`THEN AND`(a_1, a_2)

también es una regla tipo 1.

Cada una de las reglas anteriores se tiene que transformar en una regla atómica.

Definición 6.1 *Una regla atómica es aquella cuyo evento y condición son una conjunción de uno o más eventos y cláusulas condicionales, respectivamente, y su acción ejecuta sólo una tarea.*

Una regla r_i se puede dividir en varias reglas atómicas siguiendo los siguientes pasos:

Paso 1. Si r_i es atómica, termina. Si no, continúa el Paso 2.

Paso 2 Transforma cada elemento de r_i en su forma normal disyuntiva usando las reglas del álgebra de Boole de manera que cada elemento de la regla consista de una o más disyunciones cada una de las cuales, a su vez, es una conjunción de una o más instrucciones. Si la regla transformada no tiene disyunciones en sus elementos, entonces es una regla atómica de acuerdo a la Definición 6.1. De otra forma, ve al siguiente paso.

Paso 3 Divide r_i en un conjunto de reglas atómicas cuya premisa y acción son las disyunciones obtenidas en el Paso 2.

La normalización de los cuatro tipos de reglas descritos previamente es la siguiente:

Normalización 1. Dado que las reglas del tipo 1 no tienen disyunciones en su premisa, esta normalización puede realizarse ejecutando directamente el Paso 3.

ON [NOT]AND$(e_1, e_2, ..., e_n)$		ON [NOT]AND$(e_1, e_2, ..., e_n)$
IF [NOT]AND$(c_1, c_2, ..., c_m)$	$\cdots$	IF [NOT]AND$(c_1, c_2, ..., c_m)$
THEN a_1		THEN a_k

Para las reglas de los tipos 2, 3 y 4 consideramos un caso simple en el cual se ejecuta una única acción, a_1. Para la conjunción de acciones sólo necesitamos un paso más para dividir las acciones como en la Normalización 1.

Normalización 2. Reglas del tipo 2 pueden ser normalizadas en un conjunto de reglas atómicas de la siguiente forma:

ON [NOT]AND$(e_1, e_2, ..., e_n)$		ON [NOT]AND$(e_1, e_2, ..., e_n)$
IF [NOT]c_1	$\cdots$	IF [NOT]c_m
THEN a_1		THEN a_1

Normalización 3. Reglas del tipo 3 son normalizadas en reglas atómicas de la siguiente forma:

ON [NOT]e_1		ON [NOT]e_n
IF [NOT]AND$(c_1, c_2, ..., c_m)$	$\cdots$	IF [NOT]AND$(c_1, c_2, ..., c_m)$
THEN a_1		THEN a_1

Normalización 4. Reglas del tipo 4 pueden ser normalizadas como sigue:

ON [NOT]e_1	ON [NOT]e_1		ON [NOT]e_1
IF [NOT]c_1	IF [NOT]c_2	$\cdots$	IF [NOT]c_m
THEN a_1	THEN a_1		THEN a_1
ON [NOT]e_2	ON [NOT]e_2		ON [NOT]e_2
IF [NOT]c_1	IF [NOT]c_2	$\cdots$	IF [NOT]c_m
THEN a_1	THEN a_1		THEN a_1
$\vdots$	$\vdots$		$\vdots$
ON [NOT]e_n	ON [NOT]e_n		ON [NOT]e_n
IF [NOT]c_1	IF [NOT]c_2	$\cdots$	IF [NOT]c_m
THEN a_1	THEN a_1		THEN a_1

La normalización de reglas no afecta el conocimiento que estas representan ni su desempeño y, aunque se pueden generar reglas redundantes en este proceso, es necesario para detectar errores con precisión.

La base de reglas del Ejemplo 5.1 no necesita ser normalizada puesto que todas las reglas son atómicas.

6.2. Modelado de reglas

Para simplificar el proceso de detección de errores, la base de reglas atómicas se modela como una estructura de CCPN. El modelado así como la generación de la matriz de conexión se discutieron en el Capítulo 4, por lo tanto no abundaremos en más detalles. Sólo anotaremos algunas características adicionales.

6.2.1. Conversión de ECA - CCPN

Debido al proceso de normalización, la CCPN adquiere algunas características especiales:

- Las transiciones tipo regla sólo puede tener un arco de salida puesto que ahora cada regla sólo puede ejecutar una acción.
- Los lugares virtuales ya no se emplean en la CCPN dado que después de la normalización todos los eventos son conjunciones de uno o más eventos primitivos.

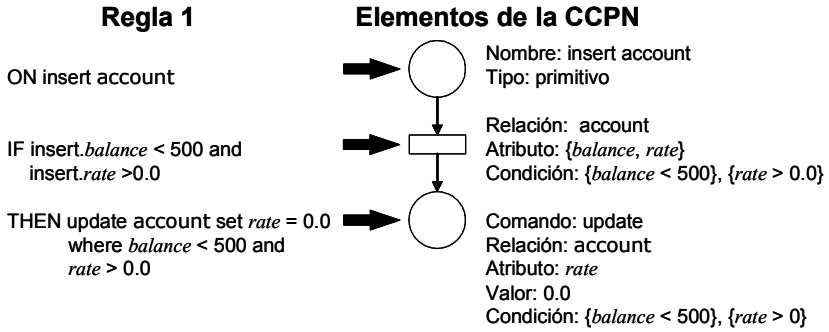


Figura 6.2: Correspondencia entre los elementos de la Regla 1 del ejemplo 5.1 y los elementos de la CCPN, así como la información que almacena cada uno de ellos

La figura 6.2 muestra la correspondencia entre los elementos de la regla 1 del ejemplo 5.1 y los elementos de la CCPN así como la información que cada uno de estos almacena.

La figura 6.3 muestra la CCPN de la base de reglas del ejemplo 5.1. Cada elemento almacena la información del evento/acción y condición en la forma que se describe en la figura 6.2.

La tabla 6.1 muestra una descripción breve de los lugares de la CCPN de la figura 6.3.

Tabla 6.1. Correspondencia entre eventos/acciones y lugares

Lugar	**Tipo**	**Evento/Acción**
p_0	Primitivo	insert `account`
p_1	Primitivo	update `account`.*rate*
p_2	Primitivo	update `account`.*balance*
p_3	Copia	copia de p_1
p_4	Copia	copia de p_2 (izquierdo)
p_5	Copia	copia de p_2 (derecho)
p_6	Primitivo	delete `account`
p_7	Copia	copia de p_0 (izquierdo)
p_8	Copia	copia de p_0 (derecho)

La tabla 6.2 muestra la descripción resumida de las transiciones de la CCPN de la figura 6.3.

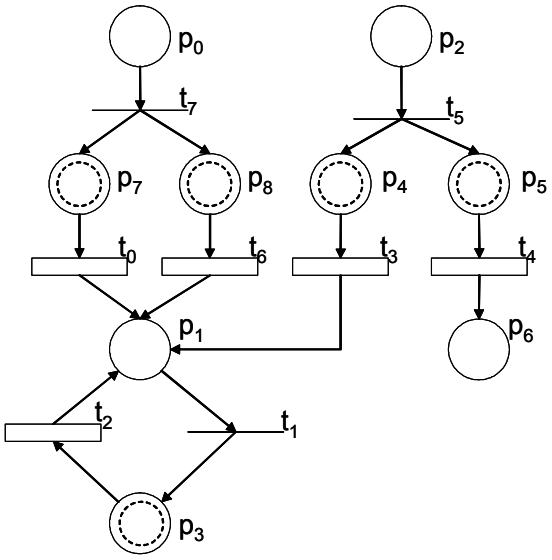


Figura 6.3: CCPN de la base de reglas del ejemplo 5.1

Tabla 6.2. Correspondencia entre reglas y transiciones

Transición	**Tipo**	**Regla**	**Condición**
t_0	Regla	**Regla 1**	insert.*balance* $<$ 500 and insert.*rate* $>$ 0
t_1	Copia	-	verdadero
t_2	Regla	**Regla 2**	update.*balance* $<$ 500 and update.*rate* $>$ 0
t_3	Regla	**Regla 3**	update.*rate* $>$ 1 and update.*rate* $<$ 3
t_4	Regla	**Regla 4**	update.*rate* $>$ 1 and update.*rate* $<$ 3
t_5	Copia	-	verdadero
t_6	Regla	**Regla 5**	insert.*balance* $<$ 500 and insert.*rate* $>$ 0
t_7	Copia	-	verdadero

6.2.2. Generación de la matriz de conexión

La matriz de conexión también refleja los cambios que la estructura de la CCPN sufre a causa del proceso de normalización de reglas. Por ejemplo, la i-ésima fila que representa la i-ésima transición tipo regla, quien a su vez simboliza la i-ésima regla, tendrá un único valor positivo ya que cada

regla atómica sólo ejecuta una acción.

La matriz de conexión de la CCPN de la figura 6.3 es la siguiente:

$$A_{m\times n} = \begin{array}{c|cccccccc|c} & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ \hline 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & -1 & 0 \\ 1 & 0 & -1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 2 & 0 & 1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 \\ 3 & 0 & 1 & 0 & 0 & -1 & 0 & 0 & 0 & 0 \\ 4 & 0 & 0 & 0 & 0 & 0 & -1 & 1 & 0 & 0 \\ 5 & 0 & 0 & -1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 6 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & -1 \\ 7 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \end{array}$$

6.3. Verificación de errores

La verificación de errores permite obtener una lista de los yerros que tiene una base de reglas. Para ello el proceso se lleva a cabo en dos etapas: primero, usando la CCPN se extraen las reglas que pueden tener errores. Segundo, se examina la interacción de las reglas extraídas para determinar si, efectivamente, tienen errores. Realizar el proceso de verificación de esta forma nos permite enfocar nuestro análisis sólo en aquellas reglas que pudieran tener errores, evitando aplicar exámenes innecesarios al resto de ellas.

6.3.1. Detección de estructuras de error en la CCPN

Cuando en la base de reglas existen errores, en la CCPN aparecen algunas estructuras especiales que nos permiten detectarlos. Sin embargo, es una ardua labor para el usuario inspeccionar visualmente la red y descubrir tales estructuras manualmente. Por esta razón, usamos la matriz de conexión de la CCPN para desarrollar algoritmos que permitan la identificación automática de errores.

En esta sección mostramos las estructuras en la CCPN que representan los errores de redundancia, inconsistencia, incompletitud y circularidad así como la forma de detectarlos usando la matriz de conexión. En lo sucesivo nos referiremos a la representación de R con la CCPN como CCPN_R. La

matriz de conexión de la CCPN$_R$ la denotaremos como $A_{m\times n}$ y las entradas positivas y negativas como A^+_{ij} y A^-_{ij}, respectivamente.

Redundancia

La estructura para *reglas redundantes* se muestra en la figura 6.4(a). Puesto que las transiciones t_1 y t_2 tienen los mismos lugares de entrada y de salida, si la condición almacenada en ambas transiciones es la misma entonces esas reglas son redundantes, en otro caso, este patrón es útil para detectar reglas parcialmente redundantes evento - acción.

Las reglas parcialmente redundantes evento - condición se representan por la estructura de la CCPN mostrada en la figura 6.4(b). En este caso, siempre que las condiciones almacenadas en las transiciones t_1 y t_2 sean la misma, este tipo de redundancia sucederá. Hay que observar que de acuerdo a la Definición 5.2 la acción ejecutada por las reglas no es importante, por lo que los lugares de salida son diferentes.

La figura 6.4(c) ilustra el patrón para reglas parcialmente redundantes condición - acción. En este tipo de reglas no importa cómo se formen los eventos, pueden ser compuestos o primitivos o una combinación de ambos; sin embargo, es obligatorio que las condiciones evaluadas por las transiciones t_0 y t_1 sean la misma.

La figura 6.5 muestra la estructura de CCPN para *reglas incluidas* y *parcialmente incluidas*. En este caso los lugares de entrada pueden ser ambos compuestos o uno de ellos puede ser compuesto y el otro primitivo.

En la figura 6.5(a) si ambas transiciones evalúan la misma condición o la condición almacenada en la transición t_3 es un subconjunto de la condición almacenada en la transición t_2, entonces la transición t_2 está incluida en la transición t_3 porque esta última no es tan restrictiva como t_2 (esta última necesita que los eventos p_0 y p_1 sucedan al mismo tiempo mientras que la primera sólo necesita que el evento p_0 suceda) y ambas llegan al mismo resultado.

Hemos mostrado las reglas parcialmente incluidas evento - condición en la figura 6.5(b). En este caso, si las condiciones almacenadas en la transición t_2 y en la transición t_3 son la misma, o la condición evaluada por la transición t_3 es un subconjunto de la condición evaluada por la transición t_2, entonces la transición t_2 será incluida por la transición t_3.

El patrón para detectar reglas parcialmente incluidas condición - acción es similar al que se

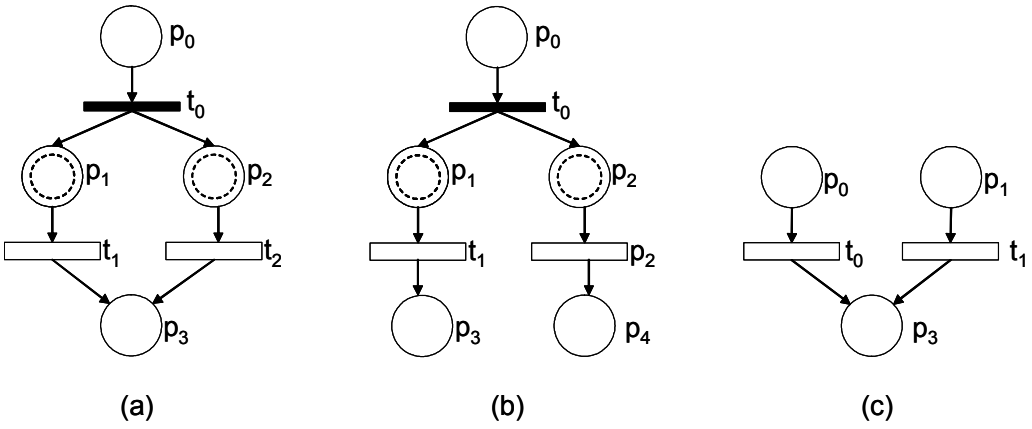


Figura 6.4: (a) El patrón de redundancia y reglas parcialmente redundantes evento - acción en la CCPN. (b) El patrón de las reglas parcialmente redundantes evento - condición en la CCPN. (c) El patrón para reglas parcialmente redundantes condición - acción en la CCPN

muesta en la figura 6.4(c) puesto que la composición de eventos no se toma en cuenta, si la condición evaluada por la transición t_1 es un subconjunto de la condición evaluada por la transición t_0 entonces la transición t_0 será incluida por la transición t_1.

Los siguientes algoritmos nos ayudan a identificar en la CCPN las estructuras descritas:

Usando la matriz de conexión desarrollamos el algoritmo mostrado en la figura 6.6 para encontrar las reglas que ejecutan la misma acción. Denominamos a este algoritmo como REGLAS_MISMA_ACCION( ).

El algoritmo anterior identifica las columnas en la matriz de conexión que tienen más de un valor positivo. Después para cada una de ellas, busca los valores positivos ya que los índices de las filas en donde se encuentran dichos elementos representan sus transiciones de entrada.

La columna 1 de la matriz de conexión anterior cumple las condiciones descritas en la **Propiedad 1**, es decir, tiene más de un valor positivo. De manera que las filas 0, 2, 3 y 6 representan reglas que ejecutan la misma acción, es decir, $t_0^\bullet = t_2^\bullet = t_3^\bullet = t_6^\bullet = p_1$.

Las figuras 6.7 y 6.8 muestran los algoritmos de los métodos OBTENER_LUGARES_INCIDENTES( ), el cual busca en la matriz de conexión las columnas con más de una entrada positiva, y OBTENER_TRANSICION_ENTRADA( ), el cual busca los valores positivos de cada elemento identificado

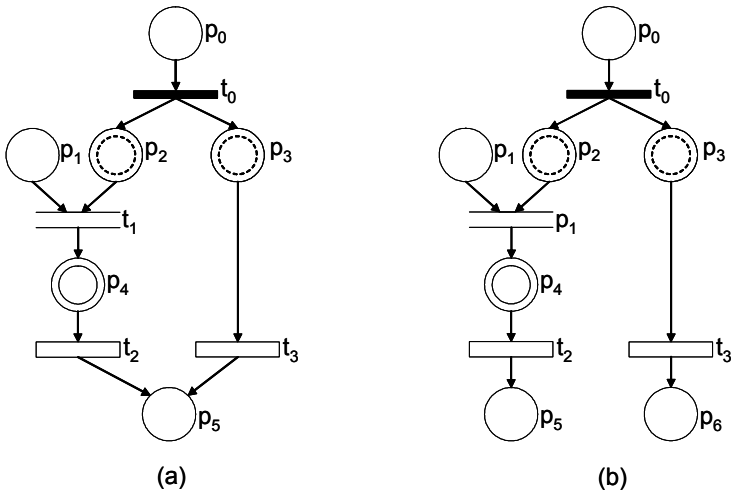


Figura 6.5: (a) La estructura de la CCPN para reglas incluidas y reglas parcialmente incluidas evento - acción. (b) La estructura en la CCPN para reglas parcialmente incluidas evento - condición

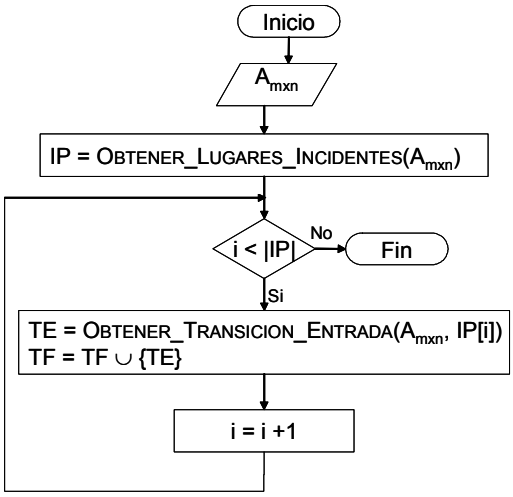


Figura 6.6: Algoritmo REGLAS_MISMA_ACCION( ). Obtiene las reglas que ejecutan la misma acción usando la matriz de incidencia

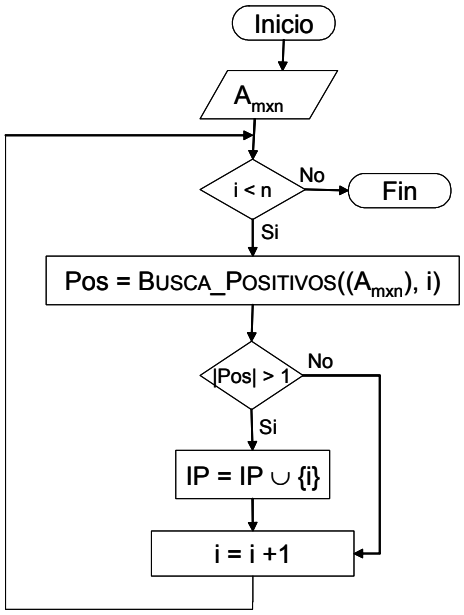


Figura 6.7: Algoritmo del método OBTENER_LUGARES_INCIDENTES( ).

con el método anterior, respectivamente. Ambos métodos se usan para obtener las reglas que ejecutan la misma acción.

La figura 6.9 muestra el algoritmo del método BUSCA_POSITIVOS( ) usado en el método OBTENER_TRANSICION_ENTRADA( ).

El método REGLAS_MISMO_EVENTO( ) permite identificar las reglas que son disparadas por el mismo evento. El algoritmo se muestra en la figura 6.10.

El método OBTENER_LUGARES_DUPLICADOS( ), cuyo algoritmo se muestra en la figura 6.11, usa la matriz de conexión para encontrar de manera automática los lugares que son duplicados por medio de una transición tipo copia.

El algoritmo del método OBTENER_TRANSICION_SALIDA( ) proporciona la primera transición tipo regla que se encuentra a partir de un cierto lugar. Su algoritmo se muestra en la figura 6.12.

El método BUSCA_NEGATIVOS( ) es similar al método BUSCA_POSITIVOS( ) sólo que, como su nombre lo indica, obtiene las entradas negativas.

Las filas 5 y 7 de la matriz de conexión cumplen las características descritas en la **Propiedad**

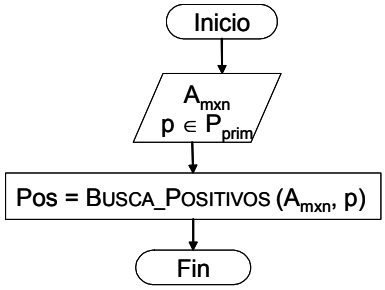


Figura 6.8: Algoritmo del método OBTENER_TRANSICION_ENTRADA( )

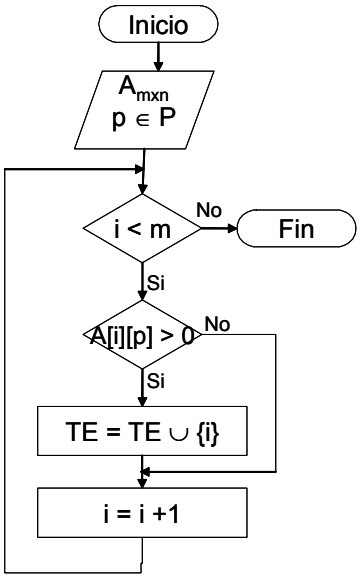


Figura 6.9: Algoritmo del método BUSCA_POSITIVOS( )

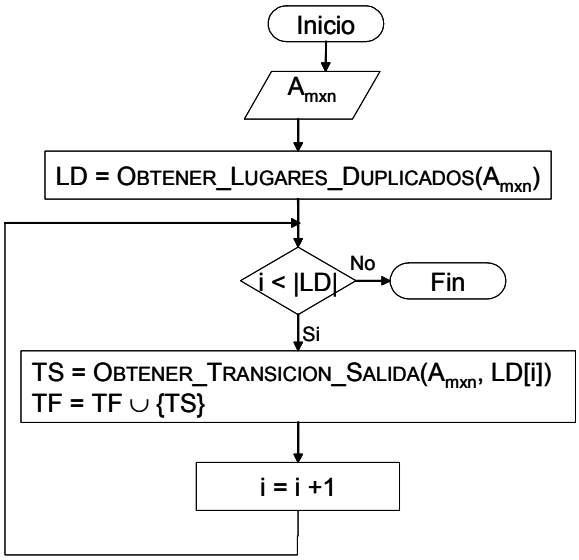


Figura 6.10: Algoritmo REGLAS_MISMO_EVENTO( ). Obtiene las reglas que son disparadas por el mismo evento, usando la matriz de incidencia

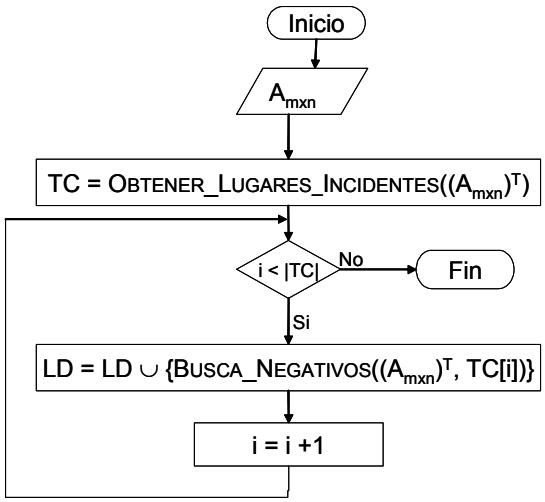


Figura 6.11: Algoritmo del método OBTENER_LUGARES_DUPLICADOS( )

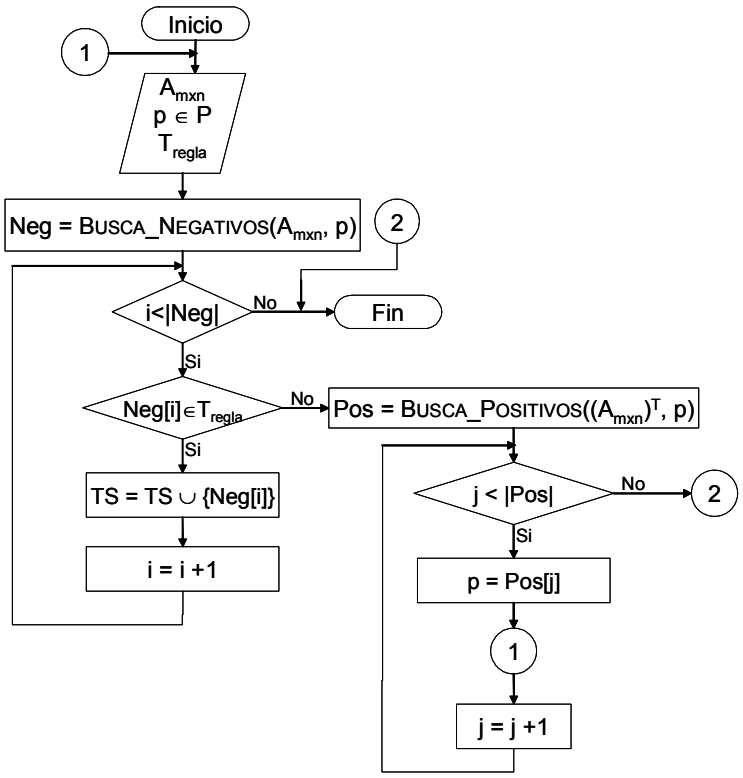


Figura 6.12: Algoritmo del método OBTENER_TRANSICION_SALIDA( )

2. Para la fila 5, la entrada $a^{-}_{5\times 2}$ indica que el lugar duplicado es el representado por la columna 2 y las transiciones que lo comparten son las correspondientes a las filas 3 y 4. Para la fila 7, la entrada $a^{-}_{7\times 0}$ indica que el lugar duplicado es el correspondiente a la columna 0 y las transiciones que lo usan como entrada son las representadas por las filas 0 y 6.

Como resultado del análisis anterior, los pares de transiciones que se deben analizar para confirmar o rechazar la presencia de errores de redundancia son los siguientes: (t_0, t_2), (t_0, t_3), (t_0, t_5), (t_2, t_3), (t_2, t_5), (t_3, t_5) y (t_3, t_4). Los primeros seis pares corresponden a la detección que se realizó de reglas que ejecutan la misma acción. El último par es el resultado del análisis de reglas disparadas por el mismo evento.

Inconsistencia

Para detectar reglas en conflicto, el diseñador de reglas activas debe indicar las acciones que pueden causar un comportamiento anormal en el funcionamiento de la base de reglas. Tal como se describió en la Sección 5.2 existen dos casos de tales acciones: 1) cuando son diferentes y contradictorias, y 2) cuando son la misma acción pero emplean valores que se oponen entre sí.

En general, las estructuras en la CCPN que representan reglas en conflicto son muy similares a las mostradas en la figura 6.4 y la figura 6.5.

Las figuras 6.4(b) y 6.5(b) ilustran las estructuras para reglas en conflicto cuando las acciones que ejecutan son diferentes y se han definido como contradictorias. Las figuras 6.4(a) y 6.5(a) muestran las estructuras para reglas en conflicto que ejecutan la misma acción pero estas se contradicen entre sí. De acuerdo a la definición 5.4, cuando la premisa de las reglas es la misma (figuras 6.4(a) y 6.4(b)) las reglas están en conflicto. En otro caso, cuando ocurra la premisa más restrictiva (figuras 6.5(a) y 6.5(b)) el conflicto también sucederá. Cuando en las figuras 6.4(a), 6.4(b), 6.5(a) y 6.5(b) las transiciones tipo regla almacenan condiciones diferentes, entonces esas estructuras representan reglas en conflicto potencial evento - acción.

Las reglas en conflicto potencial condición - acción, cuando ambas reglas ejecutan la misma acción pero con valores que se oponen entre sí, tienen el patrón mostrado en la figura 6.4(c). La figura 6.13 muestra la estructura en la CCPN para reglas en conflicto potencial condición - acción cuando las acciones son diferentes y contradictorias.

Dado que las estructuras para reglas redundantes y reglas en conflicto son muy similares, usamos

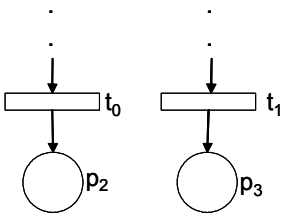


Figura 6.13: Estructura en la CCPN para reglas en conflicto potencial condición - acción

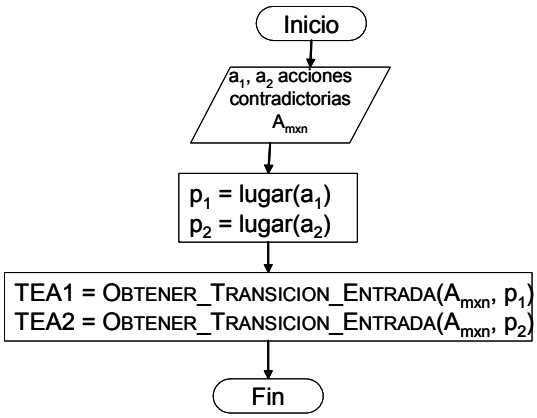


Figura 6.14: Algoritmo del método REGLAS_INCONSISTENCIA( )

los algoritmos de detección de reglas redundantes para identificar reglas en conflicto.

El algoritmo de la figura 6.14 muestra la forma general de obtener las reglas que pueden estar en conflicto a partir de la identificación de los lugares que representan las reglas contradictorias. A este algoritmo nos referiremos como REGLAS_INCONSISTENCIA( ). El método `lugar( )` regresa el índice del lugar que almacena un evento o acción. Esta información no se almacena en la matriz de conexión, sino que se genera cuando se construye la CCPN.

Supongamos que en la base de reglas del ejemplo 5.1 las acciones "delete from `account`" y" update `account`.*rate*" se definieron como contradictorias. De acuerdo a la tabla 6.1, el lugar p_6 almacena la primera acción y el lugar p_1 la segunda. Usando el método OBTENER_TRANSICION_ENTRADA( ) encontramos que sus transiciones de entrada son $\{t_4\}$ y $\{t_0, t_2, t_3, t_6\}$, respectivamente. Por lo

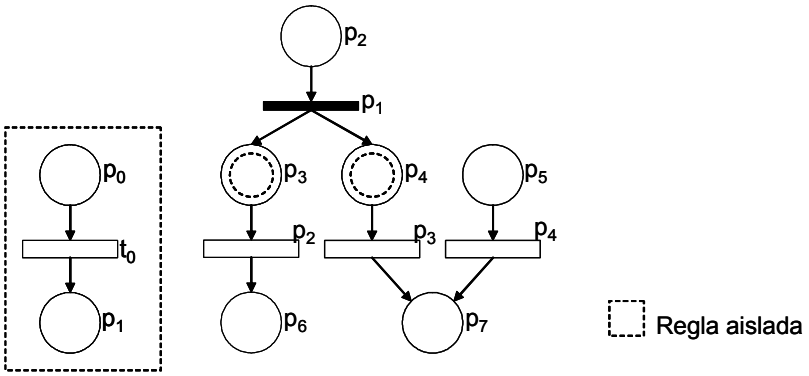


Figura 6.15: Estructura en la CCPN para reglas aisladas

tanto, los pares de transiciones que pueden estar en conflicto son (t_4, t_0), (t_4, t_1), (t_4, t_3), (t_4, t_5).

Incompletitud

Reglas aisladas

De acuerdo a la definición 5.6, una regla aislada no interactúa con ninguna otra. En la CCPN esta regla está caracterizada por una transición tipo regla cuyos lugares de entrada y salida no tienen arcos de entrada y de salida, respectivamente, tal como se muestra en la figura 6.15.

El método REGLA_AISLADA( ), cuyo algoritmo se muestra en la figura 6.16, obtiene las reglas aisladas a partir de la matriz de conexión de la CCPN.

En el ejemplo 5.1 no existe ninguna regla aislada.

Reglas de punto muerto

Una regla de punto muerto está representada por una transición tipo regla cuyo lugar de salida no tiene arcos de salida. Esto significa que este lugar representa a una acción que no es el evento que dispara a otra regla. Por ejemplo, el lugar p_6 en la figura 6.15 es un lugar que no tiene arcos de salida, por lo tanto, su transición de entrada, t_2, representa a una regla de punto muerto.

El método REGLAS_PUNTO_MUERTO( ) encuentra las reglas de punto muerto usando la matriz de conexión. El algoritmo de este método se muestra en la figura 6.17.

En la CCPN de la figura 6.3 el lugar e_6 no tiene arcos de salida, por lo tanto, su transición de entrada, t_4, representa una regla de punto muerto, es decir, la **Regla 4** es de punto muerto.

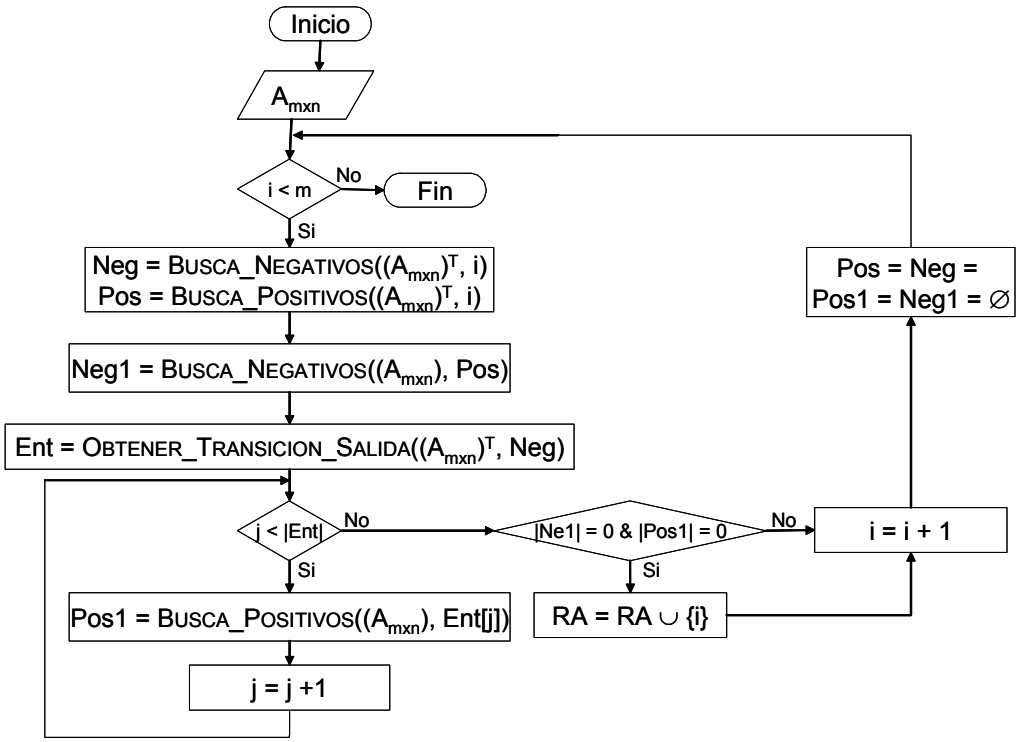


Figura 6.16: Algoritmo del método REGLA_AISLADA( )

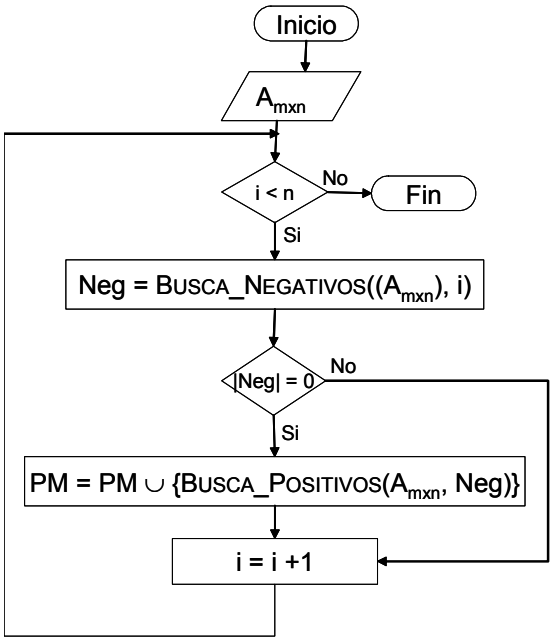


Figura 6.17: Algoritmo para el método REGLAS_PUNTO_MUERTO( )

Reglas inalcanzables

Para detectar reglas inalcanzables primero tenemos que determinar si una regla dispara a otra, para eso definimos el concepto de secuencia de disparo.

Definición 6.2 *Secuencia de disparo. Una secuencia de disparo, S, es una sucesión finita de lugares y transiciones de la forma:* $(p_1, t_1) \rightarrow (t_1, p_2) \rightarrow (p_2, t_3) \rightarrow \ldots \rightarrow (t_{m-1}, p_m)$ *donde:*

$p_i \in P_{prim}, \forall i = 1 \ldots m$
$t_j \in T_{regla} \forall j = 1 \ldots m-1$
$p_{k-1} = t^{\bullet}_{k-1} = {}^{\bullet}t_k$

En la definición anterior, el lugar primitivo p_{k-1} es tanto el lugar de salida de la transición tipo regla t_{k-1} como el lugar de entrada de la transición tipo regla t_k, lo que significa que la acción de la regla $k-1$ es el evento que dispara a la regla k. Una secuencia de disparo termina cuando $p^{\bullet}_m = \emptyset$ o algún par ya se ha agregado. Desde luego las secuencias de disparo que se consideran son aquellas que tienen al menos dos transiciones tipo regla.

El algoritmo que se muestra en la figura 6.18 describe los pasos a seguir para encontrar las secuencias de disparo usando la matriz de conexión. Nos referimos a este algoritmo como el método Secuencia_Disparo( ).

Las secuencias de disparo que se encuentran usando la matriz de conexión de la CCPN de la figura 6.3 son las siguientes: $(p_0, t_0) \rightarrow (t_0, p_1) \rightarrow (p_1, t_2) \rightarrow (t_2, p_1)$; $(p_0, t_6) \rightarrow (t_6, p_1) \rightarrow (p_1, t_2) \rightarrow (t_2, p_1)$ y $(p_2, t_3) \rightarrow (t_3, p_1) \rightarrow (p_1, t_2) \rightarrow (t_2, p_1)$.

A partir de estas secuencias de disparo podemos ver que las reglas que se disparan son las siguientes: (**Regla 1**, **Regla 2**), (**Regla 2**, **Regla 2**), (**Regla 5**, **Regla 2**), (**Regla 3**, **Regla 2**). En los pares anteriores el primer elemento es la regla que dispara y el segundo elemento es la regla disparada. Estos pares de reglas deben analizarse posteriormente.

Cabe mencionar que si en una secuencia de disparo $S = (p_1, t_1) \rightarrow (t_1, p_2) \rightarrow (p_2, t_3) \rightarrow \ldots \rightarrow (t_{m-1}, p_m)$ p_2 no activa o desactiva a t_3 entonces esta transición representa una regla inalcanzable y, a su vez, la transición t_1 se convierte en una regla de punto muerto dado que su proceso de ejecución se interrumpe.

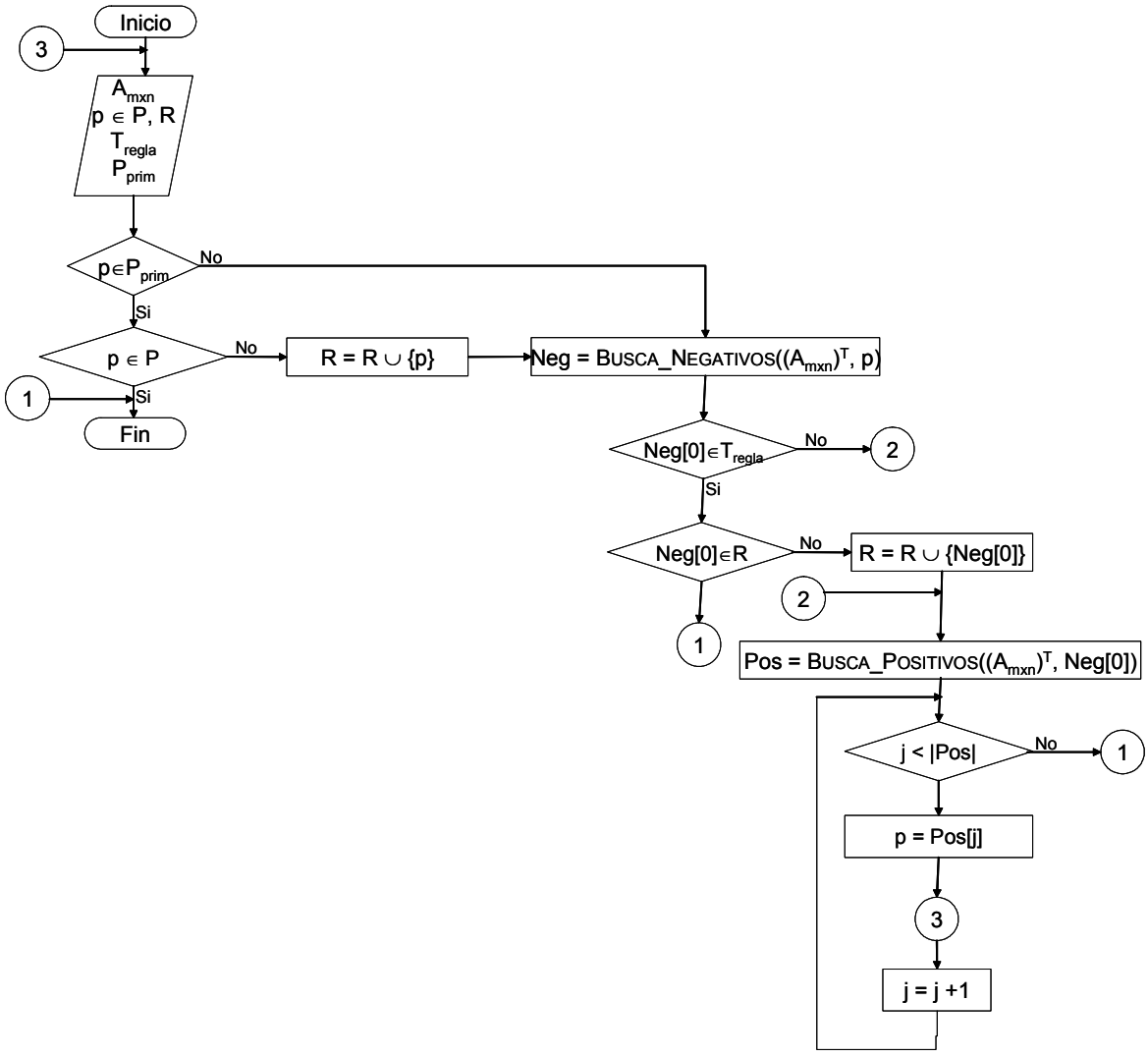


Figura 6.18: Algoritmo SECUENCIA_DISPARO( )

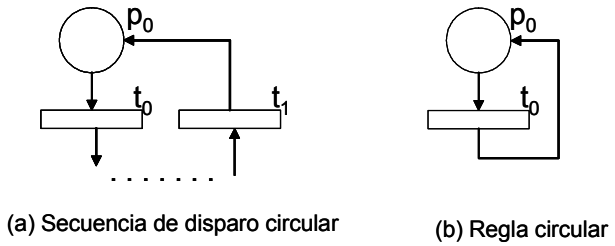


Figura 6.19: Reglas circulares

Circularidad

La figura 6.19(a) muestra la estructura para reglas circulares. Un caso especial de este tipo de reglas es cuando la regla forma un ciclo consigo misma como se muestra en la figura 6.19(b).

Las reglas circulares pueden ser detectadas usando el método SECUENCIA_DISPARO( ) o por medio de la propiedad de los T-invariantes [44].

En la CCPN de la figura 6.3 hay un ciclo descrito por la secuencia $S = (e_1, t_2) \rightarrow (t_2, e_1)$, de manera que la **Regla 2** es un autociclo.

La tabla 6.3 resume los pares de transiciones que pueden tener errores así como el tipo de error que se debe analizar.

Tabla 6.3. Lista de transiciones que representan errores

Tipo de error	**Transiciones a analizar**
Redundancia	(t_0, t_2), (t_0, t_3), (t_0, t_5), (t_2, t_3), (t_2, t_5), (t_3, t_5) y (t_3, t_4)
Inconsistencia	(t_4, t_0), (t_4, t_2), (t_4, t_3) y (t_4, t_5)
Incompletitud	t_4 (regla de punto muerto) (t_0, t_2), (t_2, t_2), (t_5, t_2) y (t_3, t_2) (reglas inalcanzables)
Circularidad	t_2

Las transiciones anteriores se han identificado como que pueden tener errores. Para determinar con exactitud si tienen algún problema es necesario examinar su interacción.

6.3.2. Análisis de interacción de reglas

El análisis de interacción para la detección de errores consiste en determinar si dos reglas cumplen las condiciones descritas en las definiciones del Capítulo 5, es decir, determinar, en general, cuándo el evento/condición de una regla contiene a otro(a), cuándo dos acciones son iguales y cuándo una regla activa/desactiva a otra. Iniciaremos describiendo estos conceptos y posteriormente daremos las condiciones que deben cumplir dos reglas para que se presente un error.

Conceptos preliminares

Después del proceso de normalización de reglas el evento que dispara de cada una de ellas es una conjunción de uno o más eventos.

Definición 6.3 *Conjunto de eventos disparadores. Dada una regla r_i disparada por el evento* `AND`*(e_1, e_2, ..., e_n) decimos que el conjunto de eventos disparadores, E_i, está formado por todos los eventos primitivos e_1, e_2, ..., e_n del evento compuesto* `AND`*, es decir, $E_i = \{e_1, e_2, ..., e_n\}$.*

De esta manera podemos definir el concepto de contención de eventos.

Definición 6.4 *Contención de eventos. Dadas dos reglas r_i y r_j, $e_i \subseteq e_j$ si y sólo si $E_i \subseteq E_j$.*

En la CCPN los lugares primitivos de entrada de una transición tipo regla representan los eventos que disparan a una regla. Encontrando tales lugares es posible determinar la contención de eventos. Cuando se llama al método Obtener_Transicion_Salida( ), descrito en la sección anterior, con los parámetros $A^T_{m\times n}$ - la transpuesta de la matriz de conexión -, i - el índice de la transición - y P_{prim}, - el conjunto de lugares primitivos - se obtienen los lugares primitivos de entrada de la transición t_i. De manera que $E_i \subseteq E_j$ si Obtener_Transicion_Salida($A^T_{m\times n}$, i, P_{prim}) $\subseteq$ Obtener_Transicion_Salida($A^T_{m\times n}$, j, P_{prim}).

Por ejemplo, en la CCPN de la figura 6.3, $P_{prim} = \{0, 1, 2, 6\}$, y las llamadas al método Obtener_Transicion_Salida($A^T_{m\times n}$, 0, P_{prim}) y Obtener_Transicion_Salida($A^T_{m\times n}$, 6, P_{prim}) producen la misma salida, 0, lo que significa que las transiciones t_0 y t_6 tienen a p_0 como lugar primitivo de entrada. Es importante notar que Obtener_Transicion_Salida($A^T_{m\times n}$, t_i, P_{prim}) y ${}^\bullet t_i$ no obtienen los mismos resultados, ya que el primero busca repetidamente en la matriz

de conexión hasta que encuentra el(los) lugar(es) *primitivo(s)* de entrada de t_i, mientras que el segundo regresa el(los) lugar(es) inmediato(s) de entrada sin importar su tipo.

Nuestro análisis para determinar si la condición de la regla r_i está contenida en la correspondiente de r_j está basado en el concepto que denominamos *dominio de la condición.*

Definición 6.5 *Dominio de una condición. El dominio de la condición c_i, denotado por $Dom(c_i)$, es el conjunto de información que satisface c_i.*

El dominio de la condición de la **Regla 3** del ejemplo 5.1 está formado por todas las tuplas en la relación `account` cuyo atributo *rate* tiene un valor que está entre 1 y 3.

Usando el concepto de dominio de la condición podemos definir la contención de condiciones.

Definición 6.6 *Contención de condiciones. Dadas dos condiciones c_i y c_j, ci $\subseteq$ cj si y sólo si $Dom(c_i) \subseteq Dom(c_j)$.*

Consideremos las condiciones de la **Regla 1** y la **Regla 3** del ejemplo 5.1. $c_1 \subseteq c_3$ dado que $Dom(c_1) \subseteq Dom(c_3)$ porque $1 < rate < 3 \subseteq 1 < rate < 10$. El valor 10 se usa a pesar de que explícitamente no está descrito en la condición de la **Regla 1** porque es el máximo valor que el atributo *rate* puede tomar de acuerdo a la definición del esquema de relación `account`.

Algunas veces las condiciones no son tan simples como las usadas en las reglas del ejemplo 5.1 y establecer su dominio no es una tarea sencilla. Por ejemplo, el dominio de la condición $c =$ "`account`.*balance* $>$ 500 and exists (select * from `low_acc` where `low_acc`.*num* = `account`.*num* and `low_acc`.*end* = null)" no se puede determinar directamente. Por esta razón, desarrollamos un método, detallado en el diagrama de flujo de la figura 6.20, que nos permite obtener el dominio de condiciones como la descrita.

El método toma como entrada una condición. Si esta tiene la forma: <*variable* **op** *variable*> o <*variable* **op** *constante*>, donde *variable* es un atributo o característica perteneciente al sistema (un atributo de un esquema de relación, por ejemplo), **op** puede ser un elemento del conjunto $\{=, != , >, \geqslant, <, \leqslant\}$ y *constante* es un valor dado, entonces se verifica si la condición está acotada, es decir, si tiene límites inferior y superior. Si la condición está acotada, el proceso termina. Si este no es el caso, se invoca al procedimiento `Acotar( )` para asignar los límites correspondientes usando la información del sistema. Cuando esta información no existe, se usan los símbolos $-\infty$

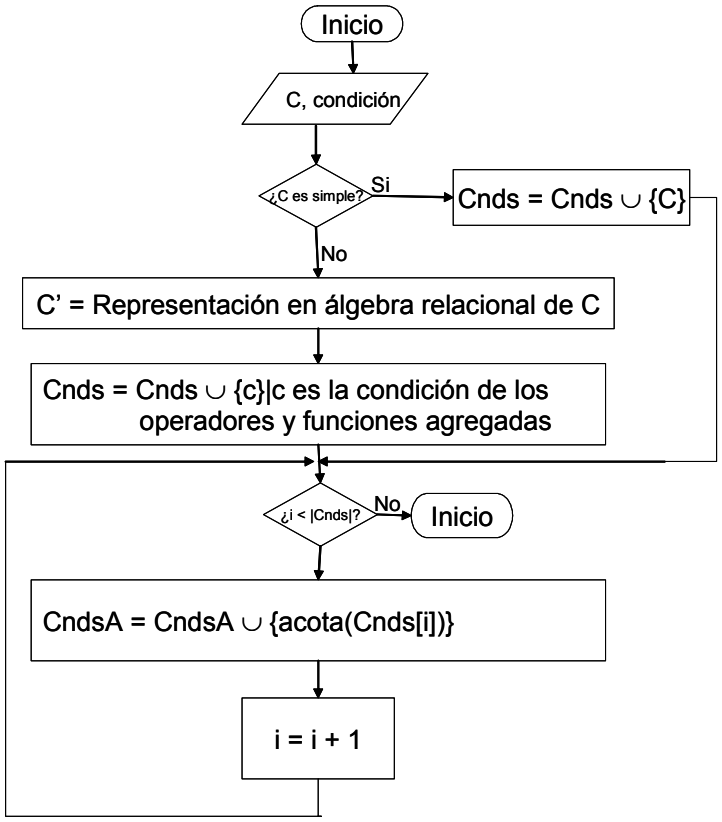


Figura 6.20: Diagrama de flujo que muestra los pasos para encontrar el dominio de una condición

y $+\infty$ como límite inferior y superior, respectivamente. En caso de que la condición no tenga la forma especificada inicialmente, se obtiene su representación en álgebra relacional y se toman las condiciones de los operadores que las posean (por ejemplo, reunión ($\bowtie$), selección (σ) y funciones agregadas). Finalmente, se revisa cada condición obtenida para acotarla y el proceso termina. Cabe mencionar que el formato en el que se presenta cada condición consta de tres elementos {({Relación}, {Atributo}, {Condición})} donde "Relación" especifica la fuente de información, "Atributo" pertenece a Relación y "Condición" es la condición acotada que se establece sobre Atributo.

El procedimiento para transformar una consulta en su representación equivalente en álgebra relacional se lleva a cabo usando un enfoque similar al reportado en la referencia [49], el cual está basado en el uso de gramáticas para identificar los tipos de consultas y llevar a cabo las transformaciones correspondientes.

Tomemos la condición c = "($\sigma_{(\texttt{account}.balance>500)}$ (account)) $\bowtie_{(\texttt{account}.num\,=\,\texttt{low\_acc}.num)}$ ($\sigma_{(\texttt{low\_acc}.end=null)}$ (low_acc))". Como c no tiene la forma especificada al inicio del procedimiento, se obtiene su representación en álgebra relacional. De manera que c' = "($\sigma_{(\texttt{account}.balance>500)}$ (account)) $\bowtie_{(\texttt{account}.num=\texttt{low\_acc}.num)}$ ($\sigma_{(\texttt{low\_acc}.end=null)}$ (low_acc))". El conjunto de condiciones obtenidas es el siguiente: {$balance$ >500, account.num = low_acc.num, low_acc.end = null}. Acotar cada condición produce el siguiente conjunto: {500<$balance$<$+\infty$, account.num = low_acc.num, low_acc.end = null}. En la parte final, para cada condición se obtiene: {({account}, {account.$balance$}, {500<account.$balance$<$+\infty$}), ({account, low_acc}, {account.num, account.num}, {account.num = low_acc.num}), ({low_acc}, {low_acc.end}, {low_acc.end = $null$})}, lo que significa que todas las tuplas en las relaciones account y low_acc que satisfagan las tres condiciones anteriores serán el dominio de c.

Una vez calculado el dominio de una condición, determinamos la contención de condiciones mediante el algoritmo mostrado en la figura 6.21.

El algoritmo toma el dominio de las condiciones c_1 y c_2 y analiza si ambas poseen al menos un atributo en común, At_C. Posteriormente, en cada dominio se toma la condición asociada al atributo At_C y se verifica si alguna de ellas es un subconjunto de la otra o si son iguales. Si no existe el atributo At_C significa que aunque ambas condiciones pueden ser verdaderas al mismo tiempo, no verifican el mismo conjunto de información.

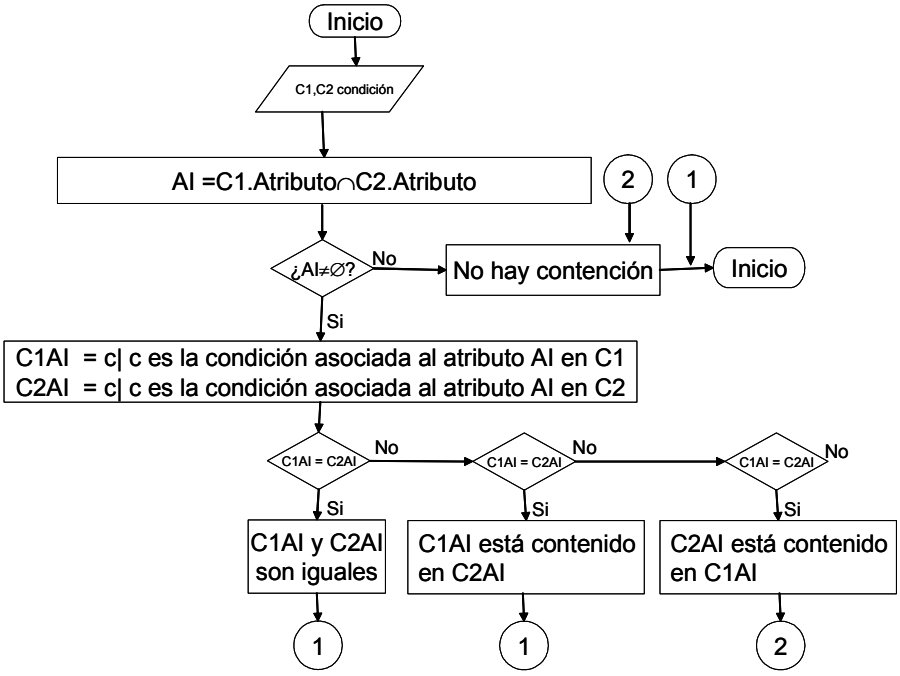


Figura 6.21: Diagrama de flujo para mostrar cuándo una condición está contenida en otra

Tomemos la condición c anterior y supongamos que la **Regla 1** del ejemplo 5.1 evalúa la condición c_1 = "insert.*balance* > 600 and insert.*balance* < 1000". Entonces, $Dom(c_1)$ = {({`account`}, {`account`.*balance*}, {600<`account`.*balance*<1000})}. El atributo que $Dom(c)$ y $Dom(c_1)$ tienen en común es At_C = {`account`.*balance*}. Las condiciones en cada dominio relacionadas con ese atributo son: $C_{At_C} = \{500 < \texttt{account}.balance < +\infty\}$ y $C_{1At_C} = \{600 < \texttt{account}.balance < 1000\}$. Como $C_{1At_C} \subset C_{At_C}$ entonces$Dom(c_1) \subset Dom(c)$ y $c_1 \subset c$.

El concepto de dominio de la condición también es útil para determinar si dos acciones ejecutan la misma acción.

Definición 6.7 *Igualdad de acciones. Dos acciones a_i y a_j son iguales si modifican, de la misma forma, la misma información.*

La información de la acción ejecutada por una regla se describe con las siguientes características:

1. **Comando**: la acción que se debe ejecutar.
2. **Relación**: la información sobre la cual se ejecutará el comando.
3. [**Atributo**]: el atributo afectado por la ejecución del comando.
4. [**Valor**]: el nuevo valor asignado al atributo.
5. [**Condición**]: especifica el conjunto de información modificada por la ejecución del comando.

Las características entre corchetes pueden o no estar presentes. A cada una de ellas nos referimos como a_i.Comando, a_i.Relación, etc.

Para determinar si dos acciones a_i y a_j ejecutan la misma acción verificamos que se cumplan las siguientes condiciones:

1. a_i.Comando = a_j.Comando
2. a_i.Relación = a_j.Relación
3. a_i.Atributo = a_j.Atributo
4. a_i.Valor = a_j.Valor

5. $Dom(a_i.\text{Condición}) \cap Dom(a_j.\text{Condición}) \neq \emptyset$

Si los requisitos 1 - 5 se cumplen, a_i y a_j llevan a cabo la misma tarea ya que ejecutan la misma operación (condición 1), en la misma fuente de datos (condiciones 2 y 3), usando los mismos valores (condición 4) y, sobretodo, en el mismo segmento de información (condición 5).

De acuerdo a las condiciones anteriores, las acciones de la **Regla 1** y la **Regla 2** del ejemplo 5.1 son iguales, pero diferentes de la acción de la **Regla 3** porque esta última no tiene el mismo dominio, es decir, no modifica las mismas tuplas que las anteriores. Además, incluso si lo hiciera, serían diferentes ya que usa un valor distinto para modificar el atributo *rate*.

Los últimos tipos de interacción que necesitamos describir son la activación y desactivación de reglas. Para ello necesitamos encontrar primero los siguientes datos acerca de la acción a_i y la condición c_j:

1. $\text{Rel} = a_i.\text{Relación} \cap \text{Rels}_{c_j}$
2. $\text{At} = a_i.\text{Atributo} \cap \text{At}_{c_j}$
3. $\text{Cond} = Dom(c_j)_{At}.\text{Condición}$
4. $\text{Dom} = Dom(a_i.\text{Condición})_{At} \cap Dom(c_j)_{At}$

donde:
$\text{Rels}_{c_j} = \cup_{k=1}^{|Dom(c_j)|} Dom(c_j)_i.\text{Relación}$,
$\text{At}_{c_j} = \cup_{k=1}^{|Dom(c_j)|} Dom(c_j)_i.\text{Atributo}$, y
$Dom(c_j)_{At}.\text{Condición}$ es la condición asociada al atributo At.

Hay que notar que si se cumplen los puntos 1 y 2, entonces Rel y At son iguales a a_i.Relación y a_i.Atributo, respectivamente.

Tal como se describió en la Sección 3.1.5, la regla r_i *activa* a la regla r_j siempre que la ejecución de la acción de la primera produzca datos que puedan satisfacer la condición de la segunda. Usando el concepto de dominio de la condición podemos definir la activación de reglas.

Definición 6.8 *Activación de reglas. La regla r_i activa la regla r_j si la ejecución de a_i puede hacer que $Dom(c_j)$ cambie de $Dom(c_j) = \emptyset$ a $Dom(c_j) \neq \emptyset$. Para que esto suceda es necesario que se cumplan las siguientes condiciones:*

1. a_i.Comando = "insert" o a_i.Comando = "update"
2. Rel $\neq \emptyset$
3. At $\neq \emptyset$
4. a_i.Valor $\in$ Cond
5. Dom $\neq \emptyset$

En la definición anterior, si después de que se ejecuta a_i, $Dom(c_j) \neq \emptyset$ quiere decir que a_i generó datos que pueden hacer que el resultado de la evaluación de c_j sea verdadero.

Para mostrar la activación de reglas, consideremos la **Regla 2** y la **Regla 3** del ejemplo 5.1 y verifiquemos si la **Regla 3** activa a la **Regla 2**. Para facilitar la lectura, la tabla 6.4 muestra la información tanto de la acción de la **Regla 3**, a_3, como del dominio de la condición de la **Regla 2**, c_2.

Tabla 6.4. Información de la acción de la **Regla 3** y de la condición de la **Regla 2**

Datos de a_3	**Dominio de c_2**
Comando = update Relación = `account` Atributo = `account`.*rate* Valor = 2.0 Condición = *rate* > 1.0 and *rate* < 3.0 $Dom(a_3$.Condición) = {({`account`}, {`account`.*rate*}, {1.0<`account`.*rate*<3.0})}	$Dom(c_2)$ = {({`account`}, {`account`.*balance*}, {0.0<`account`.*balance*<500.0}), ({`account`}, {`account`.*rate*}, {0<`account`.*rate*<+∞})}

Como ya se tiene la información del dominio de c_2, podemos obtener los conjuntos Rels_{c_j} y At_{c_j}.

$\text{Rels}_{c_2} = \cup_{k=1}^{|Dom(c_2)|} Dom(c_2)_i$.Relación = {`account`}

$\text{At}_{c_2} = \cup_{k=1}^{|Dom(c_2)|} Dom(c_2)_i$.Atributo = {`account`.*balance*, `account`.*rate*}

Con esta información se pueden encontrar los elementos Rel, At, Cond y Dom.

1. Rel = a_3.Relación $\cap$ Rels_{c_2} = {`account`} $\cap$ {`account`} = {`account`}

2. At = a_3.Atributo $\cap$ At_{c_2} = {account.*rate*}$\cap${account.*balance*, account.*rate*} = {account.*rate*}
3. Cond = $Dom(c_2)_{At}$.Condición = $Dom(c_2)_{\texttt{account}.rate}$.Condición = {0<account.*rate*<10}
4. Dom = $Dom(a_3.\text{Condición})_{At} \cap Dom(c_2)_{At}$ = $Dom(a_3.\text{Condición})_{\texttt{account}.rate} \cap Dom(c_2)_{\texttt{account}.rate}$ = {1.0<account.*rate*<3.0} $\cap$ {0.0<account.*rate*<10} = {1.0<account.*rate*<3.0}

Finalmente concluimos que r_3 activa a r_2 ya que:

1. a_3.Comando = update
2. Rel = {account}, entonces Rel $\neq \emptyset$
3. At = {account.*rate*}, entonces At $\neq \emptyset$
4. a_3.Valor $\in$ Cond ya que 2.0$\in$ {0.0<account.*rate*<10}
5. Dom ={1.0<account.*rate*<3.0}, entonces Dom $\neq \emptyset$

es decir, se cumplen las condiciones descritas en la Definición 6.8.
La desactivación de reglas se define de manera similar a la activación.

Definición 6.9 *Desactivación de reglas. La regla r_i desactiva la regla r_j si la ejecución de a_i puede hacer que el dominio de c_j cambie de $Dom(c_j) \neq \emptyset$ a $Dom(c_j) = \emptyset$. Las condiciones que a_i y c_j deben cumplir para que esto pase son las siguientes:*

1. a_i.Comando = "delete"
2. Rel $\neq \emptyset$
3. At $\neq \emptyset$
4. Dom $\neq \emptyset$

Las condiciones anteriores especifican que si a_i remueve datos que c_j verifica, entonces r_i desactiva a r_j.

Es fácil observar que en el ejemplo 5.1 la **Regla 4** desactiva a la **Regla 3** ya que elimina todas las tuplas en la relación `account` cuyo valor en el atributo *rate* está entre 1 y 3. Estos registros son necesarios para que la condición de la **Regla 3** sea verdadera.

Una vez que definimos la forma de determinar la interacción de las reglas, podemos establecer las condiciones para detectar exactamente los errores que puede contener una base de reglas. Enunciaremos estas condiciones como propiedades sobre la matriz de conexión y continuaremos la lista iniciada con las propiedades de la Sección 6.3.

Consideraremos que t_i y t_j, $1 \leq i, j \leq m, i \neq j$; son dos transiciones tipo regla que representan a las reglas r_i y r_j de R. Por conveniencia usamos la notación de transiciones pero es claro que en $A_{m \times n}$ las transiciones son representadas por las filas.

Ya que hemos definido los conceptos previos, presentamos el análisis para cada tipo de error descrito.

Determinación de errores

Después del análisis de las estructuras que representan errores, podemos concluir que las reglas que pueden tener errores presentan las siguientes características (una de ellas o ambas):

1. Reglas con la misma acción. Están representadas por aquellas transiciones tipo regla que tienen lugares primitivos de salida con más de un arco de entrada.

2. Reglas con eventos comunes. Están representadas por aquellas transiciones tipo regla que tienen lugares primitivos duplicados a través de una transición tipo copia.

Las características anteriores se pueden establecer como propiedades en la matriz de conexión de la forma siguiente:

Propiedad 1. Reglas con la misma acción. Si varias reglas de R ejecutan la misma acción, entonces:

1. $\exists p_i \in P$ tal que $p_i \in P_{prim}$ y $|^\bullet p_i| > 1$.

2. $\exists c_i$ en $A_{m \times n}$ con más de una a^+_{ji}, $1 \leq j \leq m$

Demostración. Sean t_i y t_j, $1 \leq i, j \leq m, i \neq j$; dos transiciones que representan a r_i y r_j de R, respectivamente, las cuales ejecutan la misma acción. Dado que ambas reglas son atómicas, entonces las transiciones t_i y t_j tienen sólo un arco de salida. Debido a que las acciones en la CCPN se representan por lugares de salida primitivos, entonces vamos a suponer que $t_i^\bullet = p_k$ y $t_j^\bullet = p_l$; sin embargo, dado que la misma acción no puede ser representada por dos diferentes lugares de salida, entonces $p_k = p_l$ y $\mid {}^\bullet p_i \mid > 1$.

Las transiciones t_i y t_j corresponden con la $i-ésima$ y la $j-ésima$ filas de $A_{m\times n}$, respectivamente, y cada una de ellas sólo tiene una entrada positiva debido a la normalización de reglas. Si $t_i^\bullet = t_j^\bullet = p_k$ entonces $a_{ik}^+ = a_{jk}^+$, i.e., sus correspondientes entradas positivas están en la $k-ésima$ columna de $A_{m\times n}$. Por lo tanto, la columna A_k tiene más de una entrada positiva.

Decimos que las transiciones contenidas en ${}^\bullet p_i$ representan las reglas que tienen la misma acción y, a su vez, las transiciones están representadas por sus filas correspondientes en $A_{m\times n}$.■

La siguiente propiedad se centra en la característica número dos.

Propiedad 2. Reglas con eventos comunes. Decimos que varias reglas de R tienen un evento común si:

1. $\exists t_m$ tal que $t_m \in T_{copia}$ y $\mid t_m^\bullet \mid > 1$.

2. $\exists r_m$ en $A_{m\times n}$ con más de una a_{mj}^+, $1 \leq j \leq m$

Demostración. Sean t_i y t_j, $1 \leq i, j \leq m, i \neq j$; dos transiciones que representan a r_i y r_j de R, respectivamente, las cuales tienen un evento en común. Vamos a suponer que ${}^\bullet t_i \subset P_{prim}$ y ${}^\bullet t_j \subset P_{prim}$ si ambas transiciones tienen un lugar en común entonces ${}^\bullet t_i \cap^\bullet t_j \neq \emptyset$. Vamos a considerar que $p_k \in^\bullet t_i \cap^\bullet t_j$ entonces dado que un lugar no puede ser usado como lugar de entrada para dos o más reglas, el lugar p_k debe estar duplicado en dos o más lugares. Dado que los lugares copia son generados por la misma transición, t_m, entonces $\mid t_m^\bullet \mid > 1$ y $t_m \in T_{copia}$.

Las transiciones t_i y t_j corresponden con la $i-ésima$ y la $j-ésima$ filas de $A_{m\times n}$, respectivamente. Vamos a suponer que $p_k \in {}^\bullet t_i \cap {}^\bullet t_j$ pero p_k debe ser duplicado, entonces existen al menos dos columnas A_{k_1} y $A_{k_2}, 1 \leq k_1, k_2 \leq n, k_1 \neq k_2$, en $A_{m\times n}$ las cuales representan dos lugares duplicados, p_{k_1} y p_{k_2}, respectivamente. Debido a que tales lugares provienen de la misma transición, entonces $a_{mk_1}^+ = a_{mk_2}^+$, y por lo tanto la fila r_m tiene más de una entrada positiva.

Decimos que ${}^\bullet t_m = p_l$ está representada por a^-_{ml}, y las reglas para las cuales p_l es un evento común están representadas por las transiciones obtenidas por el método OBTENER_TRANSICION_SALIDA( ), las cuales están representadas por sus correspondientes filas en $A_{m\times n}$.■

Redundancia

La siguiente propiedad y corolarios definen las condiciones para obtener conclusiones acerca del error de redundancia y sus diferentes subtipos.

Propiedad 3. Redundancia. Decimos que r_j está incluida en r_i si t_i y t_j satisfacen las siguientes condiciones:

1. OBTENER_TRANSICION_SALIDA($A^T_{m\times n}$, t_i, P_{prim}) $\subseteq$ OBTENER_TRANSICION_SALIDA($A^T_{m\times n}, t_j, P_{prim}$)
2. $Dom(t_i.Condición) \subseteq Dom(t_j.Condición)$
3. $t_i^\bullet.Acción = t_j^\bullet.Acción$

Cuando en las condiciones anteriores, la igualdad se cumple, decimos que r_i y r_j son redundantes.

Demostración. Queremos demostrar que t_i y t_j satisfacen las condiciones de la Definición 5.1.

Sean E_k y $C_k, 1 \leq k \leq n$, el conjunto de lugares de entrada primitivos de y el conjunto de cláusulas condicionales de t_k, respectivamente.

1. Si OBTENER_TRANSICION_SALIDA ($A^T_{m\times n}, t_i, P_{prim}$) $\subseteq$ OBTENER_TRANSICION_SALIDA ($A^T_{m\times n}$, t_j, P_{prim}) significa que OBTENER_TRANSICION_SALIDA($A^T_{m\times n}, t_i, P_{prim}$) $\cap$ OBTENER_TRANSICION_SALIDA($A^T_{m\times n}, t_j, P_{prim}$) $\neq \emptyset$ y, por la **Propiedad 2,** t_i y t_j tienen al menos un lugar común, de manera que podemos decir que $E_i \subseteq E_j$
2. Si $Dom\,(t_i.Condición) \subseteq Dom\,(t_j.Condición)$ entonces t_i y t_j son activadas por t_j.Condición y entonces $C_i \subseteq C_j$.
3. Si $t_i^\bullet.Acción = t_j^\bullet.Acción$ entonces t_i y t_j ejecutan la misma acción.

De acuerdo a 1), 2) y 3) la transición t_j está incluida en la transición t_i, i.e., r_j está incluida en r_i. Cuando la igualdad se cumple r_j y r_i son redundantes.■

Los siguientes corolarios se obtienen directamente de la **Propiedad 3.** Cada uno de ellos es útil para detectar un tipo de redundancia (inclusión) parcial.

Corolario 3.1. Reglas parcialmente incluidas (redundantes) evento - condición. Supongamos las mismas condiciones de la **Propiedad 3**, pero en lugar de tener $t_i^\bullet.Acción = t_j^\bullet.Acción$ consideraremos $t_i^\bullet.Acción \neq t_j^\bullet.Acción$, i.e., r_i y r_j no ejecutan la misma acción. Tomando en cuenta estas condiciones, r_j es parcialmente incluida (redundante) evento - condición por r_i.■

Corolario 3.2. Reglas parcialmente incluidas (redundantes) evento - acción. Supongamos las mismas condiciones de la **Propiedad 3**, pero en lugar de considerar $Dom\,(t_i.Condición) \subseteq Dom(t_j.Condición)$ vamos a considerar $Dom\,(t_i.Condición) \subsetneq Dom(t_j.Condición)$. Bajo esta suposición, r_j es parcialmente incluida (redundante) evento - acción por r_i.■

Corolario 3.3. Reglas parcialmente incluidas (redundantes) condición - acción. Supongamos las mismas condiciones de la **Propiedad 3**, pero en lugar de considerar OBTENER_TRANSICION_SALIDA $(A^T_{m\times n}, t_j, P_{prim}) \subseteq$ OBTENER_TRANSICION_SALIDA $(A^T_{m\times n}, t_i, P_{prim})$ supondremos que OBTENER_TRANSICION_SALIDA $(A^T_{m\times n}, t_j, P_{prim}) \subsetneq$ OBTENER_TRANSICION_SALIDA $(A^T_{m\times n}, t_i, P_{prim})$. Entonces r_j es parcialmente incluida (redundante) condición - acción por r_i.■

En la Sección 6.3.1 encontramos que los pares de reglas (**Regla 1**, **Regla 2**), (**Regla 1**, **Regla 3**), (**Regla 1**, **Regla 5**), (**Regla 2**, **Regla 3**), (**Regla 2**, **Regla 5**), (**Regla 3**, **Regla 5**) y (**Regla 3**, **Regla 4**) podían tener errores de redundancia. Tomemos el par (**Regla 1**, **Regla 5**) y analicemos sus respectivas transiciones tipo regla t_0 y t_6 (ver tabla 6.2).

De acuerdo a la Propiedad 3, primero tenemos que encontrar los lugares primitivos de entrada de cada transición.

E_0 = OBTENER_TRANSICION_SALIDA($A^T_{m\times n}$, t_0, P_{prim}) = {0}

E_6 = OBTENER_TRANSICION_SALIDA($A^T_{m\times n}$, t_6, P_{prim}) = {0}

Posteriormente, encontramos su dominio.

$Dom(t_0)$ = {({`account`},{`account`.*balance*},{0<`account`.*balance*<500}), ({`account`}, {`account`.*rate*}, {0.0<`account`.*rate*<10.0})}

$Dom(t_6)$ = {({`account`},{`account`.*balance*},{0<`account`.*balance*<500}), ({`account`}, {`account`.*rate*}, {0.0<`account`.*rate*<10.0})}

Finalmente, verificamos si $t_0^\bullet$ y $t_6^\bullet$ almacenan la misma acción.

Como en ambos lugares la acción que se ejecuta modifica de la misma forma la misma información, entonces $t_0^\bullet.Acción$ y $t_6^\bullet.Acción$ son iguales.

Dado que se cumplen las tres condiciones establecidas en la Propiedad 3, t_0 y t_6 representan reglas redundantes, es decir, la **Regla 1** y la **Regla 5** del ejemplo 5.1 son redundantes.

Procediendo de la misma forma, encontramos los resultados mostrados en la tabla 6.5:

Tabla 6.5. Reglas redundantes

Reglas	Tipo de redundancia
Regla 1, **Regla 5**	Redundantes
Regla 3, **Regla 4**	Parcialmente redundantes evento - condición
Regla 1, **Regla 2**	Parcialmente redundantes condición - acción
Regla 2, **Regla 5**	Parcialmente redundantes condición - acción

Inconsistencia

Para detectar reglas en conflicto, el usuario tiene que definir las acciones contradictorias que desea examinar. Este tipo de reglas anómalas es muy similar a las reglas redundantes porque la premisa de las reglas en conflicto tiene que cumplir las mismas condiciones que las reglas redundantes, sin embargo, las reglas en conflicto se contradicen mutuamente.

Propiedad 4. Inconsistencia. r_j está en conflicto con r_i si t_i y t_j satisfacen las siguientes condiciones:

1. Obtener_Transicion_Salida $(A^T_{m\times n}, t_i, P_{prim}) \subseteq$ Obtener_Transicion_Salida $(A^T_{m\times n}, t_j, P_{prim})$
2. $Dom(t_i.Condición) \subseteq Dom(t_j.Condición)$
3. $t_i^\bullet$ y $t_j^\bullet$ representan acciones contradictorias.

Cuando en las condiciones anteriores se cumple la igualdad, r_i y r_j están en conflicto, en otro caso, cuando la regla más restrictiva suceda el conflicto también sucederá.

Demostración. Queremos demostrar que t_i y t_j satisfacen las condiciones especificadas en la Definición 5.4.

Dado que las reglas en conflicto son similares a las reglas redundantes, la demostración es idéntica a la presentada en la **Propiedad 3**. ■

De la **Propiedad 4** se obtienen los siguientes corolarios, los cuales son útiles para detectar reglas en conflicto potencial.

Corolario 4.1. Reglas en conflicto potencial evento - acción. Supongamos las mismas condiciones de la **Propiedad 4** pero consideremos $Dom(t_i.Condición) \subsetneq Dom(t_j.Condición)$ en lugar de la condición 2). Entonces r_j está en conflicto potencial con r_i porque sólo cuando $t_i.Condición$ y $t_j.Condición$ sean evaluadas como verdaderas, el conflicto sucederá. ■

Corolario 4.2. Reglas en conflicto potencial condición - acción. Supongamos las condiciones 2) y 3) de la **Propiedad 4**. Entonces r_j está en conflicto potencial con r_i. ■

Previamente, se detectaron como inconsistentes los pares de reglas (**Regla 4**, **Regla 1**), (**Regla 4**, **Regla 2**), (**Regla 4**, **Regla 3**) y (**Regla 4**, **Regla 5**).

Analizando las transiciones que representan cada una de las reglas anteriores para determinar si cumplen con las condiciones de la Propiedad 4, se obtuvo que sólo el par (t_4, t_3) ejecutan acciones contradictorias, es decir, las reglas 4 y 3 son inconsistentes.

Incompletitud

La incompletitud considera diferentes tipos de reglas, así que enunciaremos propiedades para cada una de ellas.

Propiedad 5. Incompletitud.

(Reglas aisladas). Si t_i satisface las siguientes condiciones:

1. $\forall p \in \text{Obtener\_Transicion\_Salida}(A^T_{m\times n}, t_i, P_{prim}),\ {}^\bullet p = \emptyset$
2. $\forall p \in t_i^\bullet,\ p^\bullet = \emptyset$

Entonces r_i es una regla aislada.

Demostración. Queremos demostrar que r_i satisface las condiciones de la Definición 5.6.

Supongamos que $\exists p_i$ tal que $p_i \in \text{Obtener\_Transicion\_Salida}(A^T_{m\times n}, t_i, P_{prim})$ y ${}^\bullet p_i \neq \emptyset$. Si esto sucede, entonces $\exists t_k$ tal que $t_k \in T_{regla}$ y $t_k^\bullet = p_i$. Esto significa que p_i representa la acción de la regla representada por t_k, y el evento de la regla representada por t_i.....(1)

De igual manera, supongamos que $\exists p_j$ tal que $p_j \in t_i^\bullet$ y ${}^\bullet p_j \neq \emptyset$. Si ${}^\bullet p_j \neq \emptyset$ entonces $\exists t_l$ tal que $t_l \in T_{copia} \cup T_{comp} \cup T_{regla}$ y $t_l^\bullet = p_j$. Esto significa que p_j representa un evento usado para disparar otra regla representada por t_l y también es la acción de la regla representada por t_i. .....(2)

Tomando en cuenta (1) y (2), t_i no puede representar una regla aislada ya que está interactuando con otras reglas. Por lo tanto, para que t_i represente una regla aislada se deben cumplir las condiciones 1 y 2. ■

En el ejemplo 5.1 no se detectaron reglas aisladas.

(Reglas de punto muerto) Si t_i satisface $(t_i^\bullet)^\bullet = \emptyset$ entonces r_i es una regla de punto muerto.

Demostración. Si $(t_i^\bullet)^\bullet = \emptyset$ entonces $t_i^\bullet$ no tiene arcos de salida, de manera que $t_i^\bullet$ no es un lugar de entrada a ninguna transición. Por lo tanto, t_i es una regla de punto muerto.

En la matriz de conexión la columna i contiene una entrada positiva $a_{i\times k}$ y la columna k sólo contiene entradas positivas.■

En la Sección 6.3.1 se detectó que $(t_4^\bullet)^\bullet = \emptyset$, entonces la **Regla 4** es una regla de punto muerto.

(Reglas inalcanzables). Si la acción almacenada en $t_i^\bullet$ no activa (o desactiva) a t_j.*Condición* entonces t_j representa una regla inalcanzable, es decir, r_j es inalcanzable.

Demostración. Directa. Si t_i.Acción no produce datos (o incluso los elimina) que hagan que t_i.Condición pueda ser verdadera, entonces t_j.Acción no se puede ejecutar. Por lo tanto, t_j representa una regla inalcanzable. ■

En el ejemplo 5.1, la **Regla 2** es inalcanzable porque t_2.Condición no puede ser verdadera después de que t_2.Acción se ejecuta.

Circularidad

Propiedad 6. Circularidad Sea S una secuencia de disparo en la CCPN. Si en S cada p_i.*Acción* activa a t_{i+1}.*Condición* entonces S es una secuencia de disparo circular.

Demostración. Es claro que si p_i.Acción activa a t_{i+1}.*Condición* entonces hay un ciclo que no termina debido a que la condición almacenada en la transición siempre es verdadera depués de que se ejecuta la acción. ■

Vamos a considerar la secuencia de disparo $S = (p_1, t_2) \rightarrow (t_2, p_1)$ dado que p_1 no activa a t_2 entonces la **Regla 2** no se puede ejecutar indefinidamente.

Todos los errores detectados en la base de reglas del ejemplo 5.1 se listan en la tabla 6.6.

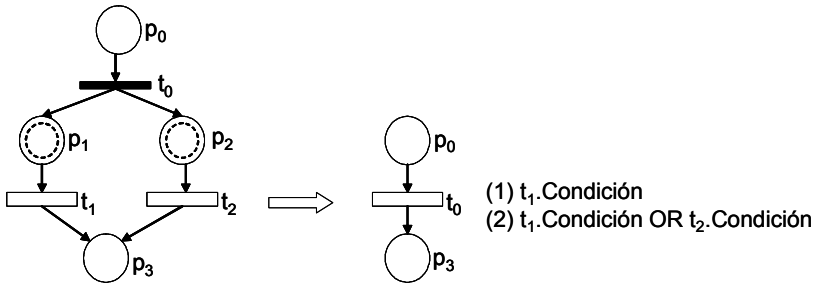


Figura 6.22: Sustitución para reglas redundantes

Tabla 6.6. Lista de errores en las reglas del ejemplo 5.1

Reglas	**Tipo de error**
Regla 1, **Regla 5**	Redundancia
Regla 3, **Regla 4**	Reglas parcialmente redundantes evento - condición
Regla 1, **Regla 2**	Reglas parcialmente redundantes condición - acción
Regla 2, **Regla 5**	Reglas parcialmente redundantes condición - acción
Regla 3, **Regla 4**	Reglas en conflicto
Regla 4	Regla de punto muerto
Regla 2	Regla inalcanzable
Regla 2	Regla circular

6.4. Mejorando la base de reglas

Una vez que los errores han sido detectados, el diseñador de las reglas ECA puede mejorar la base de reglas. Las siguientes son algunas sugerencias para mejorarla:

Primero, si hay n reglas redundantes podemos eliminar $n-1$ de ellas y sólo mantener una. Por ejemplo, veamos la figura 6.22, del lado izquierdo tenemos el patrón para reglas redundantes y del lado derecho tenemos una estructura que puede sustituirla. Cuando ambas reglas evalúan la misma condición, la nueva transición t_0 almacenará ya sea la condición almacenada en la transición t_1 o la condición almacenada en la transición t_2.

Cuando tenemos reglas parcialmente redundantes evento - acción, entonces la nueva transición

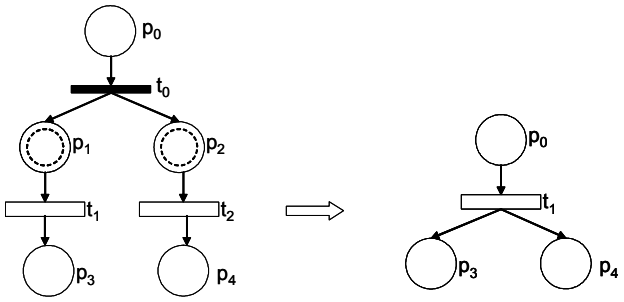


Figura 6.23: Sustitución para reglas parcialmente redundantes evento - condición

almacenará tanto la condición almacenada en la transición t_1 como la condición almacenada en la transición t_2 conectadas por un operador `OR`.

Ahora veamos las siguientes reglas:

`ON` E_1	`ON` E_1
`IF` C_1	`IF` C_1
`THEN` A_1	`THEN` A_2

Las reglas anteriores son parcialmente redundantes evento - condición y este patrón se muestra en la figura 6.23. También del lado derecho de esa figura mostramos la estructura que puede sustituir el patrón para este tipo de reglas. Como se puede observar, dado que las reglas tienen la misma premisa, sus acciones se pueden mezclar por el operador `OR`. La nueva regla se ve como sigue:

`ON` E_1
`IF` C_1
`THEN AND(`A_1,A_2)

Finalmente, veamos las siguientes reglas:

`ON` E_1	`ON` E_2
`IF` C_1	`IF` C_1
`THEN` A_1	`THEN` A_1

Las reglas anteriores son parcialmente redundantes condición - acción. Se pueden simplificar en una única regla si mezclamos sus eventos por medio del operador `OR`. La nueva regla es como la siguiente:

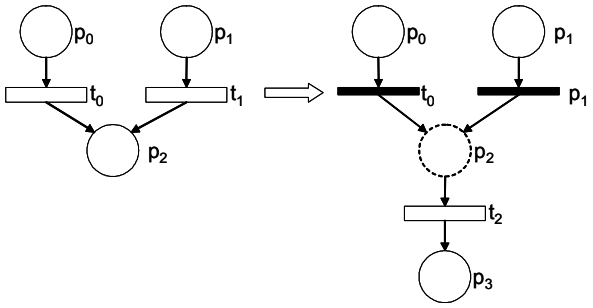


Figura 6.24: Sustitución para reglas parcialmente redundantes condición - acción

`ON OR`(E_1, E_2)
`IF` C_1
`THEN` A_1

La figura 6.24 muestra la reducción en la CCPN de este tipo de reglas.

Las reglas incluidas y parcialmente incluidas son muy similares a las mostradas para reglas redundantes y parcialmente redundantes. Sin embargo, en este caso el diseñador de las reglas ECA debe elegir la regla más (menos) restrictiva de acuerdo a las especificaciones de la base de reglas.

Para corregir las reglas en conflicto, el diseñador de las reglas ECA debe verificar las reglas anómalas contra las especificaciones de la base de reglas. Lo mismo sucede para corregir las reglas incompletas y circulares.

6.5. Caso de estudio: Desarrollo de casas inteligentes

Para demostrar nuestro método de verificación utilizaremos un conjunto de reglas desarrollado para aplicaciones de casas inteligentes [5]. Estas reglas fueron propuestas por diferentes equipos de trabajo para satisfacer necesidades de seguridad, confort y ahorro de energía. En esta aplicación se supervisa el estado de una habitación de la casa así como de uno de los aparatos electrodomésticos (horno de microondas) que ahí se encuentran. La luz que ilumina la habitación y el sensor del despachador de papel de manos también se verifican. Los sensores colocados en la casa envían datos a una BD para mantener un registro de las actividades realizadas en la habitación.

Las políticas que se integraron para manejan en el sistema son las siguientes.

Política 1. Seguridad.

Si durante 15 minutos no se ha detectado ningún movimiento en la habitación y hay aparatos electrónicos encendidos, entonces estos se deben apagar.

Política 2. Confort, ahorro de energía.

Cuando una persona ingresa a una habitación, si la luz está apagada, esta se debe encender automáticamente.

Política 3. Confort.

Cuando se detecte movimiento frente al despachador de papel, si el tiempo de detección es mayor a 10 segundos, entonces se debe de dar la cantidad adecuada de papel.

Política 4. Confort.

Cuando la intensidad de la luz natural cambie, si está por debajo de 50, incrementar la intensidad de la luz de la casa.

Política 5. Seguridad.

Si durante 15 minutos no se ha detectado ningún movimiento en la habitación y hay aparatos electrónicos encendidos, entonces estos se deben apagar.

Política 6. Seguridad.

Si durante 15 minutos no se ha detectado ningún movimiento en la habitación, entonces se debe encender la luz automáticamente para evitar que entren intrusos.

Política 7. Ahorro de energía.

Si durante 15 minutos no se ha detectado ningún movimiento en la habitación, entonces se debe apagar la luz para no desperdiciar energía.

Las reglas anteriores se representan como reglas ECA de la siguiente forma:

Regla 1

`ON AND`(insert `HabitaciónVacía`, insert `AparatosEncendidos`)

`IF HabitaciónVacía`.$tiempoTranscurrido > 15$

`THEN` insert into `AparatosApagados` values (“Microondas”, ‘time’)

Regla 2

`ON` insert `IngresaPersona`

`IF` insert.$estadoLuz = 0$

`THEN` insert into `EnciendeLuz` values (50, ‘time’)

Regla 3

`ON` insert `MovimientoDespachadorPapel`

`IF` insert.*tiempoDetectado* > 10

`THEN` insert into `DespachaPapel` values (10, 'time')

Regla 4

`ON` insert `IntensidadLuzNatural`

`IF` insert.*intensidad* < 50

`THEN` insert into `IncrementaLuz` values (55, 'time')

Regla 5

`ON AND`(insert `HabitaciónVacía`, insert `AparatosEncendidos`)

`IF` `HabitaciónVacía`.*tiempoTranscurrido* > 15

`THEN` insert into `AparatosApagados` values ("Microondas", 'time')

Regla 6

`ON` insert `HabitaciónVacía`

`IF` `HabitaciónVacía`.*tiempoTranscurrido* > 15

`THEN` insert into `EnciendeLuz` values (50, 'time')

Regla 7

`ON` insert `HabitaciónVacía`

`IF` `HabitaciónVacía`.*tiempoTranscurrido* > 15

`THEN` insert into `ApagaLuz` values ('time')

Paso 1. Normalización de reglas.

Todas las reglas de ese ejemplo están en forma normal.

Paso 2. Modelado de reglas.

Ya que las reglas están normalizadas necesitamos obtener su CCPN y matriz de conexión.

Paso 2.1 Conversión de ECA - CCPN La CCPN de esta base de de reglas se muestra en la figura 6.25. Los lugares p_{12}, p_{14}, p_{13}, p_{10} y p_{11} son lugares tipo copia, el lugar p_2 es de tipo compuesto, el resto de los lugares son primitivos. Todas las transiciones son del tipo regla excepto

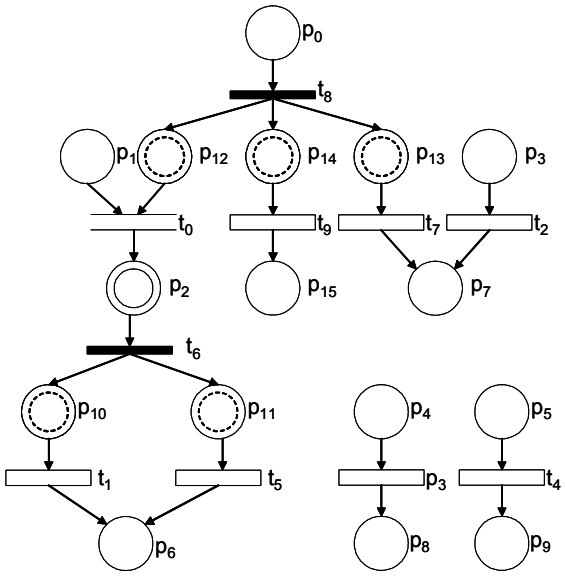


Figura 6.25: CCPN de la base de reglas para el desarrollo de casas inteligentes

t_6, t_8 (las cuales son de tipo copia) y t_0 (la cual es de tipo compuesto*)*. La tabla 6.7 muestra la correspondencia entre los eventos (y acciones) de las reglas ECA y los lugares de la CCPN.

Tabla 6.7. Correspondencia entre eventos/acciones y lugares

Lugar	**Evento/Acción**
p_0	insert `HabitaciónVacía`
p_1	insert `AparatosEncendidos`
p_2	`AND`(insert `HabitaciónVacía`, insert `AparatosEncendidos`)
p_3	insert `IngresaPersona`
p_4	insert `MovimientoDespachadorPapel`
p_5	insert `IntensidadLuzNatural`
p_6	insert `AparatosApagados`
p_7	insert `EnciendeLuz`
p_8	insert `DespachaPapel`
p_9	insert `IncrementaLuz`
p_{15}	insert `ApagaLuz`

La tabla 6.8 resume la correspondencia entre las reglas ECA (así como sus condiciones) y las transiciones.

Tabla 6.8. Correspondencia entre transiciones y reglas

Transición	**Regla**	**Condición**
t_1	**Regla 1**	`HabitaciónVacía`.*tiempoTranscurrido* > 15
t_2	**Regla 2**	insert.*estadoLuz* $= 0$
t_3	**Regla 3**	insert.*tiempoDetectado* > 10
t_4	**Regla 4**	insert.*intensidad* < 50
t_5	**Regla 5**	`HabitaciónVacía`.*tiempoTranscurrido* > 15
t_7	**Regla 6**	`HabitaciónVacía`.*tiempoTranscurrido* > 15
t_9	**Regla 7**	`HabitaciónVacía`.*tiempoTranscurrido* > 15

Paso 2.2 Generación de la matriz de conexión La matriz de conexión de la CCPN de la figura 6.25 es la siguiente:

$A_{m\times n} =$

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	-1	1	0	0	0	0	0	0	0	0	0	-1	0	0	0
1	0	0	0	0	0	0	1	0	0	0	-1	0	0	0	0	0
2	0	0	0	-1	0	0	0	1	0	0	0	0	0	0	0	0
3	0	0	0	0	-1	0	0	0	1	0	0	0	0	0	0	0
4	0	0	0	0	0	-1	0	0	0	1	0	0	0	0	0	0
5	0	0	0	0	0	0	1	0	0	0	0	-1	0	0	0	0
6	0	0	-1	0	0	0	0	0	0	0	1	1	0	0	0	0
7	0	0	0	0	0	0	0	1	0	0	0	0	0	-1	0	0
8	-1	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0
9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-1	1

La correspondencia entre los elementos de la CCPN de la figura 6.25 y las entradas de la matriz de conexión anterior se pueden observar fácilmente. Por ejemplo, si queremos analizar la transición tipo regla t_9, tomamos la fila 9 de la matriz de conexión. La entrada positiva en la fila 9 se localiza en la columna 15, entonces el lugar de salida de la transición t_9 es p_{15}, es decir, $t_9^\bullet = p_{15}$. Por otro

lado, la entrada negativa de esa misma fila está en la columna 14, el cual representa el lugar de entrada de t_9, esto es, ${}^\bullet t_9 = p_{14}$.

De acuerdo con los axiomas que propusimos para la detección de errores, primero identificamos aquellas reglas con potencial de error usando las Propiedades 1 y 2. Posteriormente, proponemos algunas mejoras para la base de reglas.

Paso 3. Verificación de errores

Identificamos las reglas que posiblemente tengan errores y posteriormente analizamos su interacción.

Paso 3.1 Detección de estructuras de error en la CCPN

1. Estructuras de redundancia Por medio de la **Propiedad 1** detectamos reglas que ejecutan la misma acción. Las columnas 6 y 7 de la matriz de conexión anterior cumplen las condiciones descritas en esa propiedad. De manera que, las filas 1 y 5 en la columna 6 representan las reglas que ejecutan la misma acción, es decir, $t_1^\bullet = t_5^\bullet = p_6$ mientras que en la columna 7, las filas 2 y 7 representan las reglas que ejecutan la misma condición, es decir, $t_2^\bullet = t_7^\bullet = p_7$. Las transiciones que se deben analizar debido a que tienen el mismo lugar primitivo de salida son: $RMA = \{(t_1, t_5), (t_2, t_7)\}$.

En la matriz de conexión previa, las filas 6 y 8 satisfacen las condiciones establecidas en la propiedad 2.

La entrada negativa localizada en $A_{6\times 2}$ representa el lugar que ha sido duplicado. Con el método REGLAS_MISMO_EVENTO( ) sabemos que las transiciones t_1 y t_5 comparten ese lugar. De manera que el par de transiciones (t_1, t_5) también debe analizarse.

Para la fila 8, la entrada negativa se localiza en $A_{8\times 0}$ la cual representa el evento que las reglas representadas por las transiciones en REGLAS_MISMO_EVENTO( ) comparten, es decir, las filas 1, 5, 7 y 9 comparten el lugar representado por la columna 0. Por lo tanto, las transiciones t_1, t_5, t_7 y t_9 comparten el lugar p_0. De forma hay que analizar los pares de transiciones $RME = \{(t_1, t_5), (t_1, t_9), (t_1, t_7), (t_5, t_9), (t_5, t_7), (t_9, t_7)\}$.

Finalmente, los pares de transiciones que se tienen que analizar para detectar algún error de redundancia son: $Analizar = \{(t_1, t_5), (t_1, t_9), (t_1, t_7), (t_2, t_7), (t_5, t_9), (t_5, t_7), (t_9, t_7)\}$

2. Estructuras de inconsistencia Para identificar las reglas en conflicto, el usuario debe definir primero las acciones contradictorias. En este ejemplo, suponemos que las acciones "insert into `ApagaLuz`" e "insert into `EnciendeLuz`" son contradictorias.

Primero, invocando al método LUGAR( ) identificamos los lugares primitivos en los cuales se almacenan las acciones contradictorias. Para la acción "insert into `ApagaLuz`" tenemos LUGAR("insert into `ApagaLuz`") = p_{15} y para la acción "insert into `EnciendeLuz`" tenemos LUGAR("insert into `EnciendeLuz`") = p_7. Posteriormente, para cada uno de esos lugares obtenemos sus transiciones (tipo regla) de entrada, es decir, ${}^\bullet p_{15} = \{t_9\}$ y ${}^\bullet p_7 = \{t_2, t_7\}$. Las transiciones que tenemos que analizar son: $Analizar = \{(t_2, t_9), (t_7, t_9)\}$.

3. Estructuras de incompletitud El error de incompletitud se caracteriza por las estructuras de reglas aisladas, reglas de punto muerto y reglas inalcanzables.

3.1 Reglas aisladas De la matriz de conexión anterior podemos ver que la fila 3 representa una transición cuyos lugares de entrada y salida están en las columnas 4 y 8, respectivamente. Examinando la columna 4 observamos que sólo contiene una entrada negativa. Ahora, analicemos la columna 8. Esta columna sólo contiene una entrada positiva localizada en la posición $A_{3\times 8}$. Entonces t_3 se tiene que analizar para ver si es una regla aislada. También se debe verificar la fila 4, t_4, dado que tiene características similares.

3.2 Reglas de punto muerto Las columnas 6, 7, 8, 9 y 15 sólo tienen valores positivos, por lo tanto hay que analizarlos.

3.3 Reglas inalcanzables Las secuencias de disparo que encontramos son las siguientes:
$(p_0, t_1) \rightarrow (t_1, p_6)$
$(p_0, t_5) \rightarrow (t_5, p_6)$
$(p_0, t_9) \rightarrow (t_9, p_{15})$
$(p_0, t_7) \rightarrow (t_7, p_7)$
$(p_3, t_2) \rightarrow (t_2, p_7)$
$(p_4, t_3) \rightarrow (t_3, p_8)$
$(p_5, t_4) \rightarrow (t_4, p_9)$

De acuerdo a estas secuencias tenemos que analizar las transiciones t_1, t_5, t_9, t_7, t_2, t_3 y t_4 para ver si se activan o desactivan.

4. Estructuras circulares En la CCPN de la figura 6.25 no existen ciclos.

De las transiciones detectadas previamente se tiene que analizar su interacción.

1. Análisis de redundancia Las transiciones que se deben analizar para detectar errores de redundancia son $Analizar = \{(t_1, t_5), (t_1, t_9), (t_1, t_7), (t_2, t_7), (t_5, t_9), (t_5, t_7), (t_9, t_7)\}$.

Tomaremos el par de transiciones (t_1, t_5).

Las reglas redundantes deben satisfacer las condiciones especificadas en la Propiedad 3. Primero, buscamos sus lugares primitivos de entrada. No es difícil observar que:

1. OBTENER_TRANSICION_SALIDA($A^T_{m\times n}$, t_1, P_{prim})=
   OBTENER_TRANSICION_SALIDA($A^T_{m\times n}, t_5, P_{prim}$)
2. $Dom(t_1.Condición) = Dom(t_5.Condición)$
3. $t_1^{\bullet}.Acción = t_5^{\bullet}.Acción$

De manera que las transiciones t_1 y t_5 representan reglas redundantes. Entonces la **Regla 1** y la **Regla 5** son redundantes.

Procediendo de la misma forma con el resto de las transiciones obtenemos la siguiente lista de errores:

1. **Regla 1**, **Regla 5** redundantes.
2. **Regla 1**, **Regla 7** parcialmente incluidas evento - condición
3. **Regla 1**, **Regla 6** parcialmente incluidas evento - condición
4. **Regla 5**, **Regla 7** parcialmente incluidas evento - condición
5. **Regla 5**, **Regla 6** parcialmente incluidas evento - condición
6. **Regla 7**, **Regla 6** parcialmente redundantes evento - condición

La **Regla 2** y la **Regla 6** no tienen ningún problema porque aún cuando ejecutan la misma acción, su evento y condición son diferentes.

2. Análisis de inconsistencia Las transiciones que tenemos que analizar son: $Analizar = \{(t_2, t_9), (t_7, t_9)\}$.

Las transiciones (t_2, t_9) no tienen ningún problema porque su evento y condición son diferentes.

Ahora examinaremos las transiciones (t_7, t_9). De acuerdo a la Propiedad 4, si OBTENER_TRANSICION_SALIDA $(A^T_{m\times n}, t_7, P_{prim})$ = OBTENER_TRANSICION_SALIDA $(A^T_{m\times n}, t_9, P_{prim})$ y $Dom(t_7.\text{Condición}) = Dom(t_9.\text{Condición})$ entonces esas transiciones están en conflicto. Previamente demostramos que esas transiciones satisfacen tales condiciones, por lo tanto, representan reglas en conflicto, es decir, la **Regla 6** y la **Regla 7** son reglas en conflicto.

3. Análisis de incompletitud

3. 1 Reglas aisladas La transición t_3 satisface la primera condición de la Propiedad 5 porque ${}^\bullet({}^\bullet t_3) = \emptyset$ y también satisface la segunda condición ya que $(t_3^\bullet)^\bullet = \emptyset$. Debido a esto, t_3 representa una regla aislada, es decir, la **Regla 3** es una regla aislada. Lo mismo sucede con t_4, por lo tanto, la **Regla 4** es una regla aislada.

3.2 Reglas de punto muerto Los lugares 6, 7, 8, 9 y 15 no tienen arcos de salida (valores negativos en la matriz de conexión), por lo tanto, las transiciones t_1, t_2, t_3, t_4, t_5, t_7 y t_9 representan reglas de punto muerto.

3.3 Reglas inalcanzables Las transiciones t_1, t_5, t_9, t_7, t_2, t_3 y t_4 no se activan ni se desactivan a sí mismas, por lo tanto, que sean alcanzables o no depende del contexto en el cual ocurran los eventos.

Todos los errores que hemos detectado en la base de reglas del Ejemplo 6.25 se listan a continuación:

1. **Regla 1**, **Regla 5** redundantes.
2. **Regla 1**, **Regla 7** parcialmente incluidas evento - condición
3. **Regla 1**, **Regla 6** parcialmente incluidas evento - condición
4. **Regla 5**, **Regla 7** parcialmente incluidas evento - condición

5. **Regla 5**, **Regla 6** parcialmente incluidas evento - condición
6. **Regla 7**, **Regla 6** parcialmente redundantes evento - condición
7. **Regla 6** y **Regla 7** son reglas en conflicto.
8. **Regla 3** y **Regla 4** son reglas aisladas.
9. **Regla 1** punto muerto
10. **Regla 2** punto muerto
11. **Regla 3** punto muerto
12. **Regla 4** punto muerto
13. **Regla 5** punto muerto
14. **Regla 6** punto muerto
15. **Regla 7** punto muerto

Mejoramiento Para superar los problemas generados por las reglas anómalas, el diseñador de las misma debe tomar en cuenta la lista anterior. Usamos nuestras políticas para corrección de errores descritas en la Sección 6.4 para obtener una base de reglas libre de errores.

Primero, sugerimos eliminar las reglas redundantes. De esta forma, eliminamos la **Regla 1**. Posteriormente, el diseñador de las reglas debe examinar las reglas en conflicto. Puesto que la **Regla 6** y la **Regla 7** son reglas en conflicto y, revisando las especificaciones de las reglas preferimos la seguridad de las personas sobre el ahorro de energía, entonces eliminamos la **Regla 6**. Finalmente, dado que la **Regla 5** es una regla parcialmente incluida evento - condición en la **Regla 7**, podemos combinar sus acciones por medio del operador `AND`. La base de reglas mejorada queda de la siguiente forma:

Regla 1
`ON` insert `IngresaPersona`
`IF` insert.$estadoLuz = 0$
`THEN` insert into `EnciendeLuz` values (50, 'time')

Regla 2

`ON` insert `MovimientoDespachadorPapel`

`IF` insert.*tiempoDetectado* > 10

`THEN` insert into `DespachaPapel` values (10, 'time')

Regla 3

`ON` insert `IntensidadLuzNatural`

`IF` insert.*intensidad* < 50

`THEN` insert into `IncrementaLuz` values (55, 'time')

Regla 4

`ON` insert `HabitaciónVacía`

`IF` `HabitaciónVacía`.*tiempoTranscurrido* > 15

`THEN` insert into `ApagaLuz` values ('time') and insert into `AparatosApagados` values ("Microondas", 'time')

Vale la pena notar que el diseñador de las reglas ECA debe verificar el comportamiento completo de la base de reglas y determinar si el conjunto de reglas obtenido cumple con los objetivos de la aplicación, debido a esto, el procedimiento de mejorar la base de reglas no se realiza de forma automática.

Una vez que hemos logrado que la base de reglas esté libre de errores, podemos analizar otras propiedades importantes, tales como terminación y confluencia. La verificación de las reglas ECA es importante para estos análisis debido a que algunas reglas anómalas pueden generar conclusiones erróneas acerca de esas características, por ejemplo, las reglas redundantes pueden llevar a la detección de posibles problemas de confluencia. Sin embargo, las reglas redundantes no pueden generar estos problemas debido a que se trata de la misma regla.

6.6. Comentarios finales

La verificación de errores es un aspecto crucial en el desarrollo de base de reglas ECA que se integrarán a algún sistema. Si las reglas contienen errores entonces no es posible asegurar un comportamiento correcto del sistema. En este capítulo mostramos el desarrollo de un método de detección de errores basado en la CCPN. Nuestro método consta de tres etapas: normalización,

modelado y verificación de errores. La normalización se realiza para simplificar el proceso de detección de errores. Posteriormente, se representa la base de reglas normalizada como un modelo de CCPN y se identifican las estructuras de error. Este paso es muy importante en nuestro método ya que nos evita analizar todas las reglas de la base para determinar la existencia de errores. Identificando las estructuras de error sólo nos centramos en revisar las características de las reglas que *pueden* tener errores. Para dar resultados precisos acerca de la ocurrencia de errores, efectuamos el paso de verificación. En este paso tomamos las transiciones identificadas en el paso anterior y analizamos las relaciones entre sus elementos (evento, condición, acción) y entre ellas. La ventaja de nuestro método es podemos obtener conclusiones precisas de error sin necesidad de examinar toda la base de reglas. Por ejemplo, examinar completamente la base de reglas del ejemplo 5.1 implica realizar 10 comparaciones para determinar si existen reglas inalcanzables (un tipo de reglas que caracteriza el error de incompletitud). Nuestro método, con sólo 4 comparaciones pudo obtener resultados acertados.

Capítulo 7

Análisis Estático de Base de Reglas ECA

El análisis estático de una base de reglas implica determinar si el proceso de ejecución de las reglas *termina* y es *confluente*. Estas propiedades se describieron en la sección 3.2.2, pero para facilitar la lectura de este capítulo se presentan nuevamente.

La propiedad de *terminación* garantiza que el proceso de ejecución de una base de reglas se realiza en un número finito de pasos, es decir, que no existe un subconjunto de reglas que se está disparando entre sí continuamente formando un ciclo [1]. La figura 7.1, en la cual los nodos representan reglas y las aristas la relación PUEDE-DISPARAR, muestra que el proceso de ejecución de la **Regla 2** y la **Regla 4** termina pero cualquier disparo de la **Regla 1** ó la **Regla 3** inicia una serie de disparos que pueden no terminar.

El análisis de *confluencia* determina si el orden de ejecución de reglas disparadas simultáneamente influye en el resultado final que se obtenga. Para explicar el concepto de confluencia consideremos la figura 7.2 en donde cada nodo representa un estado del sistema. El estado inicial `S1` tiene como sucesores a los estados `S2` y `S3`, los cuales se obtienen después de que se disparan las reglas i y j, respectivamente. Cada una de estas reglas pudo haber originado disparos subsecuentes de otras reglas (denotados con "*" en el grafo). Si al finalizar cada secuencia de disparo se obtiene un único estado final `S4` independientemente del orden en que se hayan ejecutado las secuencias, entonces la base de reglas es confluente.

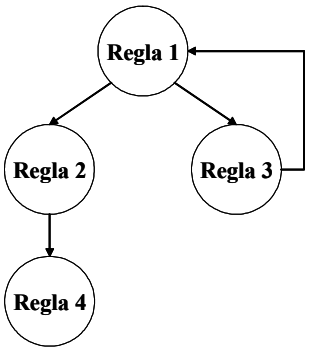


Figura 7.1: Grafo que muestra un posible disparo infinito de reglas

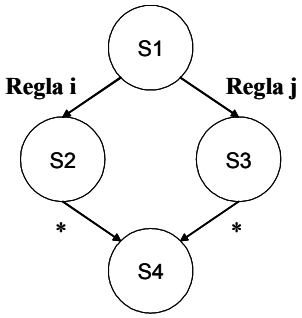


Figura 7.2: Grafo que muestra el comportamiento confluente de una base de reglas

Las propiedades de terminación y confluencia están estrechamente relacionadas con la forma en la que interactúan las reglas.

La definición de la propiedad de terminación coincide con la definición del error de circularidad descrito en el Capítulo 5. La detección de este error se describió en la Sección 6.3. Por lo tanto, ya no abordaremos su análisis en este capítulo, sólo mostraremos algunos ejemplos en la Sección 7.2.

Para ilustrar el concepto de confluencia utilizaremos la base de reglas del ejemplo 3.1. Para facilitar la lectura del capítulo, mostramos nuevamente tal ejemplo.

Ejemplo 7.1 *Base de reglas para ilustrar sus propiedades estáticas.*

Esquema de relación:

`account(`*num, name, balance, rate*`)`, *rate* $\in [1, 10]$

Base de reglas:

Regla 1

`ON` insert `account`

`IF` insert.*balance* < 500 and insert.*rate* > 0.0

`THEN` update `account` set *rate* $= 0.0$

where *balance* < 500 and *rate* > 0.0

Regla 2

`ON` update `account`.*rate*

`IF` update.*balance* < 500 and update.*rate* > 0.0

`THEN` update `account` set *rate* $= 0.0$

where *balance* < 500 and *rate* > 0.0

Regla 3

`ON` update `account`.*balance*

`IF` update.*rate* >1 and update.*rate* < 3.0

`THEN` update `account` set *rate* $= 2.0$

where *rate* > 1 and *rate* < 3.0

Rule 4

`ON` update `account`.*balance*

`IF` update.*rate* >1 and update.*rate* < 3.0

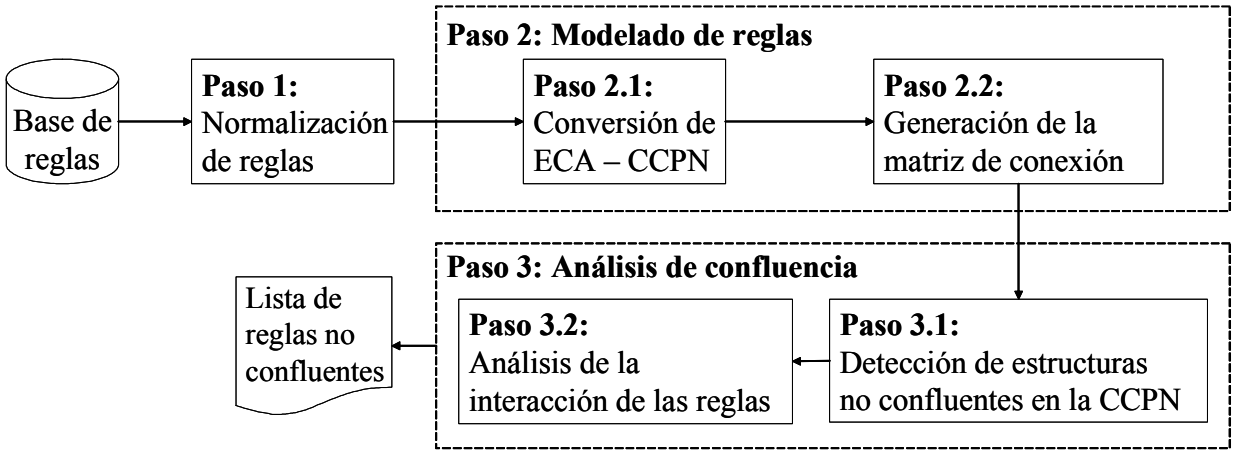


Figura 7.3: Proceso de análisis de confluencia de reglas basado en la CCPN

`THEN` delete from `account`

where $rate > 1$ and $rate < 3.0$

La propiedad de confluencia está basada en la *conmutatividad de reglas* [3]. Si en la fase de calendarización hay dos reglas r_i y r_j listas para ejecutarse y se puede obtener el mismo resultado al efectuar primero r_i y posteriormente r_j y viceversa, entonces el orden que se establezca es irrelevante ya que ambas secuencias de ejecución dejarán la información del sistema en el mismo estado final. La **Regla 3** y la **Regla 4** del ejemplo anterior son disparadas por el mismo evento y evalúan la misma condición, lo que significa que ambas pueden ejecutar su acción al mismo tiempo considerando el estado de la BD actual. Si la acción de la **Regla 4** se ejecuta primero, la acción de la **Regla 3** se verá afectada ya que las tuplas en la relación `account` que esta podía modificar fueron borradas por la acción previa. De manera que el orden de ejecución de estas reglas no se puede intercambiar. En la Sección 3.1.5 se pueden observar los resultados que se obtienen al ejecutar primero la **Regla 3** y luego la **Regla 4** y viceversa.

7.1. Análisis de confluencia basado en la CCPN

La propiedad de confluencia de una base de reglas se puede analizar usando la CCPN. El proceso para realizar tal análisis se muestra en la figura 7.3.

El primer paso es normalizar la base de reglas para simplificar el análisis. El segundo paso consiste en modelar la base de regla normalizada como una estructura de CCPN y obtener su correspondiente matriz de conexión (estos pasos se describieron con detalle en el Capítulo 5). En el tercer paso, primero, se detectan las estructuras en la CCPN que representan reglas con posibles problemas de no confluencia y, segundo, se analiza si las reglas identificadas previamente conmutan. Si se concluye que estas reglas conmutan, la base de reglas es confluente.

La ventaja de este método es que se centra en el análisis de aquellas reglas que pueden no ser confluentes, discriminando aquellas que no tienen ningún problema, por ejemplo, reglas cuyas acciones no se pueden ejecutar en el mismo instante de tiempo no se analizan.

7.1.1. Detección de estructuras no confluentes en la CCPN

Los problemas de no confluencia pueden presentarse por dos razones:

1. Existen diferentes secuencias de disparo que pueden ejecutarse simultáneamente.

2. Existen reglas que pueden ejecutarse al mismo tiempo y que modifican la misma información, pero de diferente forma.

En ambos casos, se pueden obtener resultados diferentes dependiendo del orden de ejecución elegido. Ambas situaciones se describen a continuación.

Siempre que se tengan *diferentes secuencias de disparo* para ejecutarse al mismo tiempo se pueden presentar problemas de no confluencia. Existen dos casos en los cuales se pueden generar estas diferentes secuencias. El primero de ellos es cuando dos o más reglas son disparadas por el mismo evento. La **Regla 3** y la **Regla 4** del ejemplo 7.1 cumplen esta condición, de manera que en un cierto momento se debe establecer un orden para la ejecución de sus acciones. El segundo caso se manifiesta cuando la condición de dos o más reglas se evalúa como verdadera al mismo tiempo. En el ejemplo 7.1 el resultado de la evaluación de la **Regla 1**, la **Regla 3** y la **Regla 4** puede ser verdadero al mismo tiempo ya que la **Regla 3** y la **Regla 4** evalúan exactamente la misma condición y esta, a su vez, es un subconjunto de la condición de la **Regla 1**. Por lo tanto, siempre que la condición de la **Regla 1** sea verdadera, la condición de la **Regla 3** y la **Regla 4** también será verdadera y se tendrán diferentes acciones para ejecutarse.

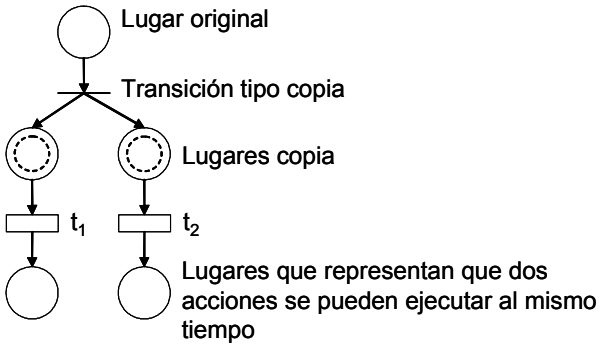


Figura 7.4: Estructura copia que representa que la acción de dos o más reglas se pueden disparar en el mismo instante de tiempo

En la CCPN cuando dos o más reglas son disparadas por el mismo evento, existe una transición tipo copia con más de un lugar (tipo copia) de salida. La estructura copia que se describió en el Capítulo 4 se muestra nuevamente aquí en la figura 7.4.

Las reglas disparadas por el mismo evento se pueden identificar usando la matriz de conexión de la CCPN. El método REGLAS_MISMO_EVENTO( ) descrito en el Capítulo 6 permite obtener esta información automáticamente.

Durante la construcción de la CCPN, el dominio de cada condición es examinado para determinar las transiciones que evalúan la misma condición y las que su condición está contenida en otra. El resultado de este análisis es un conjunto de subconjuntos, cada uno de los cuales contiene las transiciones que verifican la misma condición y las que su condición es un subconjunto de otra, respectivamente. Esta información no se representa gráficamente en la CCPN pero está disponible invocando al método CONDICIONES( ).

Puede existir un problema de no confluencia cuando *dos reglas ejecutan la misma acción empleando valores diferentes*. La **Regla 1** y la **Regla 3** del ejemplo 7.1 pueden no ser confluentes ya que ambas modifican el valor *rate* de la relación `account` pero con valores diferentes.

La figura 7.5 muestra la situación cuando varias reglas que ejecutan la misma acción pueden no generar el mismo estado final en la información del sistema.

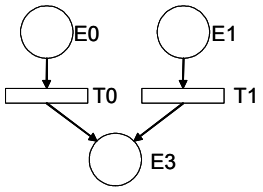


Figura 7.5: Estructura en la CCPN que muestra cuando dos o más reglas ejecutan la misma acción

El método Reglas_Misma_Acción( ) obtiene todos los lugares primitivos que tienen más de un arco de entrada ya que estos representan la acción ejecutada por varias reglas.

Cada conjunto de transiciones regresado por los métodos Reglas_Mismo_Evento( ), Condiciones( ) y Reglas_Misma_Acción( ), respectivamente, debe analizarse en pares para determinar si, efectivamente, el procesamiento de las reglas que representan es confluente o no.

La detección de las estructuras anteriores enfoca el análisis de confluencia en las reglas que pueden tener problemas. Sin embargo, en cada uno de los conjuntos de transiciones obtenidos aún pueden existir reglas que no necesiten ser examinadas. Por ejemplo, no es necesario revisar las transiciones t_i y t_j que representen reglas parcialmente redundantes condición - acción ya que ambas ejecutan exactamente la misma acción y, si existe una transición $t_k \in T_{regla}$ tal que el par $(t_i, t_k)/(t_j, t_k)$ deba ser analizado, el resultado obtenido se aplica también al par $(t_j, t_k)/(t_i, t_k)$. Las reglas parcialmente redundantes condición - acción también se pueden generar en el proceso de normalización, por lo tanto, es importante generar un registro de ellas.

El proceso de detección de estructuras no confluentes en la CCPN se puede resumir en los siguientes pasos:

1. $RME =$ Reglas_Mismo_Evento( ) (identificar las transiciones que tienen el mismo lugar de entrada).

2. $Cond =$ Condiciones( ) (identificar las transiciones que evalúan la misma condición).

3. $RMA =$ Reglas_Misma_Acción( ) (identificar las transiciones que tienen el mismo lugar de salida).

4. $Analizar = \binom{|RME|}{2} \cup \binom{|Cond|}{2} \cup \binom{|RMA|}{2}$

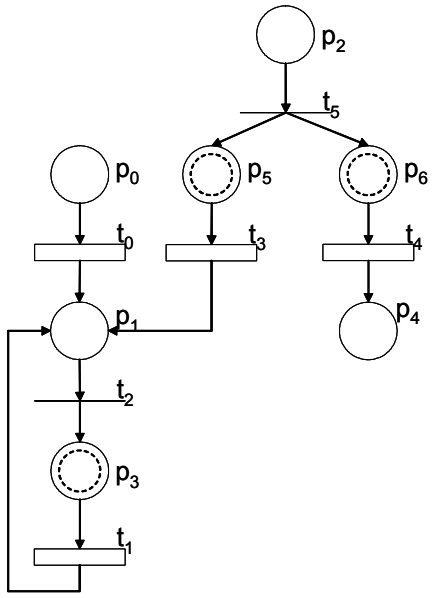


Figura 7.6: CCPN que representa la base de reglas del ejemplo 3.1

5. $Analizar = Analizar - RPRCA$

6. Si $\exists t_k \in T_{regla}$ y $\{(t_i, t_k), (t_j, t_k)\} \in Analizar$ entonces $Analizar = Analizar - \{(t_j, t_k)\}$

En los pasos anteriores, $RPRCA$ contiene los pares de transiciones que representan reglas parcialmente redundantes condición - acción.

Para demostrar la detección de estructuras no confluentes vamos a considerar la CCPN y su correspondiente matriz de conexión de la base de reglas del ejemplo 7.1.

Ejemplo 7.2 *CCPN y matriz de conexión correspondientes a la base de reglas del ejemplo 7.1.*

La figura 7.6, que es igual a la figura 4.8, muestra la CCPN que representa la base de reglas del ejemplo 7.1.

La matriz de conexión de la figura 7.6 se muestra a continuación.

		L	**U**	**G**	**A**	**R**	**E**	**S**
		0	1	2	3	4	5	6
T	0	-1	1	0	0	0	0	0
R	1	0	1	0	-1	0	0	0
A	2	0	-1	0	1	0	0	0
N	3	0	1	0	0	0	-1	0
S.	4	0	0	0	0	1	0	-1
	5	0	0	-1	0	0	1	1

Primero encontramos las diferentes secuencias de ejecución que existen en la CCPN. Para lograrlo, encontramos aquellas transiciones que tienen el mismo lugar primitivo de entrada, esto se hace identificando en la matriz de conexión las filas que tengan más de una entrada positiva. Como se puede observar, la fila 5 de la matriz de conexión anterior tiene esta característica. A partir de esta información, encontramos que las filas 3 y 4 representan las transiciones tipo regla que tienen el mismo lugar primitivo de entrada, es decir, t_3 (**Regla 3**) y t_4 (**Regla 4**) representan las reglas disparadas por el mismo evento. Estas transiciones se obtuvieron con el método Reglas_Mismo_Evento( ). De acuerdo a los resultados obtenidos, tenemos que analizar el par de transiciones (t_3, t_4).

Posteriormente, a través de la llamada al método Condiciones( ) obtenemos los siguientes conjuntos: $C_{iguales} = \{(t_0, t_1), (t_3.t_4)\}$ y $C_{subconjunto} = \{(t_0, t_3), (t_1, t_3), (t_0, t_4), (t_1, t_4)\}$. $C_{iguales}$ contiene los pares de transiciones que evalúan la misma condición y $C_{subconjunto}$ indica las transiciones cuya condición está contenida en otra. La condición del segundo elemento de cada par está contenido en la correspondiente del primer elemento. Tomando en cuenta esta información, se tienen que analizar los siguientes pares de transiciones: $(t_0, t_1), (t_0, t_3), (t_0, t_4), (t_1, t_3), (t_1, t_4), (t_3, t_4)$.

Segundo, encontramos los lugares que tienen más de un arco de entrada ya que representan reglas que ejecutan la "misma" acción. En la matriz de conexión una columna con más de una entrada positiva representa un lugar de este tipo. La columna 1 es la única que tiene más de una entrada positiva, sus transiciones, tipo regla, de entrada, t_0, t_1 y t_3, corresponden a las reglas que ejecutan la misma acción. En este caso, la **Regla 1**, la **Regla 2** y la **Regla 3** tienen la misma acción. Las transiciones anteriores se encontraron con el método Reglas_Misma_Acción( ). Los pares de transiciones que se deben evaluar como resultado de este análisis son: $(t_0, t_1), (t_0, t_3)$

y (t_1, t_3).

De esta forma, el conjunto *Analizar* queda de la siguiente manera: $Analizar = \{(t_0, t_1), (t_0, t_3), (t_0, t_4), (t_1, t_3), (t_1, t_4), (t_3, t_4)\}$.

El conjunto $RPRCA$ contiene los siguientes elementos $RPRCA = \{(t_0, t_1)\}$. Por lo tanto, tenemos que $Analizar = \{(t_0, t_3), (t_0, t_4), (t_1, t_3), (t_1, t_4), (t_3, t_4)\}$ porque eliminamos las reglas parcialmente redundantes condición - acción. Finalmente, como t_0 y t_1 están en $RPRCA$, $Analizar = \{(t_0, t_3), (t_0, t_4), (t_3, t_4)\}$.

7.1.2. Análisis de interacción de reglas

Después de que se han identificado los pares de transiciones que pueden no ser confluentes, se tiene que analizar su interacción para determinar si pueden conmutar.

Para decidir cuándo dos reglas r_i y r_j conmutan se tiene que analizar primero, si es posible intercambiar el orden de ejecución de a_i y a_j obteniendo el mismo resultado (conmutatividad de acciones) y, segundo, cual es efecto que la ejecución de a_i tiene sobre c_j y viceversa (activación/desactivación de reglas).

Definimos la conmutatividad de acciones de la siguiente manera:

Definición 7.1 *Conmutatividad de acciones. Dadas dos reglas r_i y r_j, las acciones a_i y a_j conmutan si y sólo si para todos los estados del sistema, la ejecución de a_i seguida por a_j y viceversa producen el mismo estado final en el sistema.*

Si se cumplen las siguientes condiciones, las acciones a_i y a_j conmutan.

1. INTERCAMBIA(a_i, a_j)
2. !ACTIVA(a_i, a_j.Condición)
3. !DESACTIVA(a_i, a_j.Condición)
4. !ACTIVA(a_j, a_i.Condición)
5. !DESACTIVA(a_j, a_i.Condición)

Nos referiremos a este conjunto de condiciones como CONMUTAN_ACCIONES( ). El método INTERCAMBIA(a_i, a_j) verifica que los efectos de a_i y a_j sean el mismo y las condiciones 2 - 5 examinan que la ejecución de una acción no tenga algún efecto sobre la información que la otra acción necesita.

El método INTERCAMBIA(a_i, a_j) verifica las siguientes condiciones:

1. si a_i.Relación = a_j.Relación
2. si a_i.Atributo$\cap a_j$.Atributo$\neq \emptyset$
3. si a_i.Valor $\neq a_j$.Valor
4. si $Dom(a_i.\text{Condición}) \cap Dom(a_j.\text{Condición})$
5. a_i y a_j se pueden intercambiar
6. a_i y a_j no se pueden intercambiar

Los métodos ACTIVA(a_i, c_j) y DESACTIVA(a_i, c_j) examinan si las condiciones descritas en las definiciones 6.8y 6.9, respectivamente, se cumplen.

Tomemos las acciones de la **Regla 3** y la **Regla 4** del ejemplo 7.1 para determinar si conmutan. Estas acciones no conmutan porque no se pueden intercambiar ya que (1) ambas actúan sobre la misma relación, `account`, (2) ambas modifican el atributo *rate*, (3) cada una modifica el atributo *rate* con un valor diferente, la acción de la regla 3 lo modifica con el valor 2.0 mientras que la acción de la regla 4 le "asigna" un valor nulo y (4) ambas acciones tienen efecto sobre la misma información.

Las condiciones que dos reglas tienen que cumplir para poder conmutar se enuncian en la siguiente definición:

Definición 7.2 *Conmutatividad de reglas. Dos reglas r_i y $r_j, i \neq j$, conmutan si: 1) r_i no puede activar a r_j, 2) r_i no puede desactivar a r_j, 3) condiciones 1) y 2) con los índices i, j invertidos, 4) a_i y a_j conmutan.*

Dos reglas r_i y r_j conmutan si se cumplen las siguientes condiciones:

1. $Dom(c_i) \cap Dom(c_j) \neq \emptyset$
2. CONMUTAN_ACCIONES(a_i, a_j)
3. !ACTIVA(a_i, c_j)
4. !ACTIVA(a_j, c_i)
5. !DESACTIVA(a_i, c_j)
6. !DESACTIVA(a_j, c_i)

La primera condición realiza una verificación de las condiciones de las reglas. Si el dominio de dos condiciones tiene elementos en común, significa que ambas pueden ser verdaderas al mismo tiempo y que sus acciones se pueden ejecutar simultáneamente. El resto de las condiciones verifica las condiciones de la definición 7.2.

A partir de los conceptos previos, podemos formular la siguiente propiedad.

Propiedad 1. Confluencia. Un conjunto de reglas R es confluente si todos los pares de reglas en R conmutan.

Demostración. De acuerdo a la Definición 7.2, dos reglas r_i y r_j conmutan si 1) r_i activa a r_j, 2) r_i desactiva a r_j, 3) condiciones 1) y 2) con los índices i y j invertidos, 4) a_i y a_j no conmutan. Las condiciones anteriores indican que r_i y r_j conmutan si no usan (modifican/consultan) la misma información.

Supongamos que r_i y r_j conmutan si se cumplen las siguientes condiciones: 1) r_i no puede activar a r_j, 2) r_i no puede desactivar a r_j, 3) condiciones 1) y 2) con los índices i y j invertidos, 4) a_i y a_j conmutan.

Si r_i activa a r_j significa que a_i produce datos que hacen que $Dom(c_j) \neq \emptyset$, por lo tanto, a_i y c_j utilizan la misma información.....(1)

Si r_i desactiva a r_j entonces a_i elimina datos que hacen que $Dom(c_j) = \emptyset$, por lo tanto, a_i y c_j utilizan la misma información.....(2)

Si a_i y a_j no conmutan, significa que ambas ejecutan sus respectivas tareas sobre el mismo conjunto de información.....(3)

Lo mismo sucede cuando i y j se invierten en los puntos anteriores.....(4)

Debido a que (1), (2), (3) y (4) contradicen la definición de conmutatividad de reglas, podemos concluir que r_i y r_j pueden conmutar ssi se cumplen las propiedades descritas en la Definición 7.2.

Si todos los pares de reglas en R conmutan, significa que cuando dos o más reglas se disparan y activan simultáneamente el resultado final será el mismo independientemente del orden de su ejecución. Con esto la **Propiedad 1** queda demostrada. ■

En el proceso de detección de estructuras no confluentes, detectamos los siguientes pares de transiciones: $\{(t_0, t_3), (t_0, t_4), (t_3, t_4)\}$.

Analizando el par (t_0, t_3) vemos que $Dom(t_3.\text{Condición}) \subset Dom(t_0.\text{Condición})$, por lo tanto, en un determinado momento ambas podrían evaluarse como verdaderas así que el análisis debe continuar. $t_0^\bullet$.Acción y $t_3^\bullet$.Acción no conmutan ya que esta última modifica de diferente forma un segmento de la información que la primera también modifica, por lo tanto, los resultados que se obtienen no son los mismos si se intercambian. De esta forma, concluimos que (t_0, t_3) no son confluentes. Como t_0 y t_1 representan reglas parcialmente redundantes condición - acción y el par (t_1, t_3) también debía ser analizado, entonces (t_1, t_3) tampoco son confluentes.

Procediendo de la misma forma, obtenemos los resultados mostrados en la tabla 7.1:

Tabla 7.1. Resultados de confluencia para las reglas del ejemplo 7.1

Reglas	**Resultado**	**¿Analizadas?**
Regla 1, Regla 2	Conmutan	No
Regla 1, Regla 3	No conmutan	Sí
Regla 1, Regla 4	No conmutan	Sí
Regla 2, Regla 3	No conmutan	No
Regla 2, Regla 4	No conmutan	No
Regla 3, Regla 4	No conmutan	Sí

Como se puede ver, con el método propuesto no necesitamos analizar todos los pares de reglas para obtener resultados precisos acerca de la propiedad de confluencia de una base de reglas. Con sólo 3 comparaciones fue posible determinar si se cumple esta propiedad.

7.2. Caso de estudio: Una BDA bancaria

En esta sección analizamos las propiedades de terminación y confluencia de la base de reglas descrita en la referencia [37].

Ejemplo 7.3 *Análisis de confluencia y terminación para la base de reglas de la referencia [37].*

El esquema relacional que se emplea consta de las siguientes relaciones:
`account`*(num, name, balance, rate)*
`customer`*(name, address, city)*
`low-acc`*(num, start, end)*
Las reglas activas buscan satisfacer las siguientes políticas:

Política 1. Esta política establece que cuando se ingrese un registro en la relación `account` o se actualicen los valores de `balance` o `rate` de la misma relación, si la cuenta tiene un balance menor a 500 y una tasa de interés mayor que 0 %, entonces la tasa de interés de esa cuenta se reduce a 0 %.

Política 2. Este punto se refiere a que cuando se inserte un registro en la relación `account` o se modifique el valor de `rate`, si la cuenta tiene una tasa de interés mayor a 1 % y menor a 2 %, entonces la tasa de interés de esa cuenta se incrementa al 2 %.

Política 3. Esta política describe que cuando se ingrese un nuevo cliente en la relación `customer` o se actualice el valor de `city`, si el número de clientes que viven en San Francisco es mayor a 1000, entonces la tasa de interés de las cuentas de todos los clientes que viven en San Francisco cuyo balance es mayor a 500 y cuya tasa de interés es menor a 3 % se actualiza a 1 %.

Política 4. Esta sentencia establece que cuando se ingrese un registro en la relación `account` o se modifique el valor de `balance`, si la cuenta tiene un balance menor a 500 y no se ha registrado con un valor nulo la fecha final en la relación `low-acc`, entonces esa cuenta se inserta en la relación `low-acc` con la fecha actual como fecha de inicio y un valor nulo en la fecha final.

Política 5. Establece que cuando se actualice el valor de `balance` de la relación `account`, si la cuenta de la relación `low-acc` con valor nulo en la fecha final tiene un balance mayor a 500 en la relación `account`, entonces su fecha final se actualiza con la fecha actual.

Política 6. Establece que cuando se registre una nueva cuenta en la relación `account` o se actualicen los valores de `balance` o `rate` de la misma relación, o se inserte un registro en la relación

low-acc, o se modifiquen los valores start o end de la misma, si el número total de días "malos" para una cuenta es mayor a 50 y su balance actual está entre 500 y 1000, entonces su tasa de interés se actualiza a 1 % en la relación account.

Las políticas anteriores se traducen en las siguientes reglas activas.

Regla 1

ON OR(insert account, update account.balance, update account.rate)

IF account.balance < 500 and account.rate > 0

THEN update account set rate = 0 where balance < 500 and rate > 0

Regla 2

ON OR(insert account, update account.rate)

IF account.rate > 1 and account.rate < 2

THEN update account set rate = 2 where rate > 1 and rate < 2

Regla 3

ON OR(insert customer, update customer.city)

IF 1000 < select count(name) from customer where city = 'SF'

THEN update account set rate = rate + 1 where balance > 5000 and rate < 3 and name in (select name from customer where city = 'SF'

Regla 4

ON OR(insert account, update account.balance)

IF account.balance < 500 and not exists (select * from low-acc where num=account.num and end is NULL)

THEN insert into low-acc(num, start, end)(select num, today(), NULL from account where balance < 500 and not exist (select * from low-acc where num = account.num and end is NULL)

Regla 5

ON update account.balance

IF account.balance > 500 and exists (select * from low-acc where num = account.num and end is NULL)

THEN update low-acc set end = today( ) where end is NULL and num IN (select num from account where balance > 500)

Regla 6

ON OR(update account.balance, update account.balance, update account.rate,
insert low-acc, update low-acc.start, update low-acc.end)

IF account.balance > 500 and exists (select * from low-acc where num = account.num and end is NULL)

THEN update low-acc set end = today( )where end is NULL and num IN (select num from account where balance > 500)

Paso 1. Normalización de reglas.

El primer paso en el análisis de confluencia es normalizar la base de reglas para obtener información precisa acerca de su interacción. Hay que recordar que se debe mantener un registro de las reglas parcialmente redundantes condición - acción.

Como resultado de la normalización obtenemos las siguientes reglas:

Regla 1

ON insert account

IF account.balance < 500 and account.rate > 0

THEN update account set rate = 0 where balance < 500 and rate > 0

Regla 2

ON update account.balance

IF account.balance < 500 and account.rate > 0

THEN update account set rate = 0 where balance < 500 and rate > 0

Regla 3

ON update account.rate

IF account.balance < 500 and account.rate > 0

THEN update account set rate = 0 where balance < 500 and rate > 0

Regla 4

ON insert account

IF account.rate > 1 and account.rate < 2

THEN update account set rate = 2 where rate > 1 and rate < 2

Regla 5

ON update account.rate

IF account.rate > 1 and account.rate < 2

THEN update account set rate = 2 where rate > 1 and rate < 2

Regla 6

ON insert customer

IF 1000 < select count(name) from customer where city = 'SF'

THEN update account set rate = rate + 1 where balance > 5000 and rate < 3 and
name in (select name from customer where city = 'SF'

Regla 7

ON update customer.city

IF 1000 < select count(name) from customer where city = 'SF'

THEN update account set rate = rate + 1

where balance > 5000 and rate < 3 and

name in (select name from customer where city = 'SF'

Regla 8

ON insert account

IF account.balance < 500 and not exists

(select * from low-acc where num = account.num and end is NULL)

THEN insert into low-acc(num, start, end)

(select num, today(), NULL from account

where balance < 500 and not exist (select * from low-acc

where num = account.num and end is NULL)

Regla 9

ON update account.balance
IF account.balance < 500 and not exists
(select * from low-acc where num = account.num and end is NULL)
THEN insert into low-acc(num, start, end)
(select num, today(), NULL from account
where balance < 500 and not exist (select * from low-acc
where num = account.num and end is NULL)

Regla 10

ON update account.balance
IF account.balance > 500 and exists
(select * from low-acc where num = account.num and end is NULL)
THEN update low-acc set end = today( )
where end is NULL and num IN (select num from account
where balance > 500)

Regla 11

ON insert account
IF (exists (select * from account where rate > 1 and balance > 500
and balance < 1000 and num IN (select num from low-acc
group by num having sum(end-start)>50)))
THEN update account set rate = 1
where rate > 1 and balance > 500 and balance < 1000
and num IN (select num from low-acc
group by num having sum(end-start)>50)

Regla 12

ON update account.balance
IF (exists (select * from account where rate > 1 and balance > 500
and balance < 1000 and num IN (select num from low-acc
group by num having sum(end-start)>50)))
THEN update account set rate = 1
where rate > 1 and balance > 500 and balance < 1000
and num IN (select num from low-acc
group by num having sum(end-start)>50)

Regla 13

ON update account.rate
IF (exists (select * from account where rate > 1 and balance > 500
and balance < 1000 and num IN (select num from low-acc
group by num having sum(end-start)>50)))
THEN update account set rate = 1
where rate > 1 and balance > 500 and balance < 1000
and num IN (select num from low-acc
group by num having sum(end-start)>50)

Regla 14

ON insert low-acc
IF (exists (select * from account where rate > 1 and balance > 500
and balance < 1000 and num IN (select num from low-acc
group by num having sum(end-start)>50)))
THEN update account set rate = 1
where rate > 1 and balance > 500 and balance < 1000
and num IN (select num from low-acc
group by num having sum(end-start)>50)

Regla 15

ON update low-acc.start
IF (exists (select * from account where rate > 1 and balance > 500
and balance < 1000 and num IN (select num from low-acc
group by num having sum(end-start)>50)))
THEN update account set rate = 1
where rate > 1 and balance > 500 and balance < 1000
and num IN (select num from low-acc
group by num having sum(end-start)>50)

Regla 16

ON update low-acc.end
IF (exists (select * from account where rate > 1 and balance > 500
and balance < 1000 and num IN (select num from low-acc
group by num having sum(end-start)>50)))
THEN update account set rate = 1
where rate > 1 and balance > 500 and balance < 1000
and num IN (select num from low-acc
group by num having sum(end-start)>50)

Los subconjuntos de reglas parcialmente redundantes condición - acción que se tienen son los siguientes: $S_1 = \{$**Regla 1**, **Regla 2**, **Regla 3**$\}$; $S_2 = \{$**Regla 4**, **Regla 5**$\}$; $S_3 = \{$**Regla 6**, **Regla 7**$\}$; $S_4 = \{$**Regla 8**, **Regla 9**$\}$; $S_5 = \{$**Regla 10**$\}$; $S_6 = \{$**Regla 11 - Regla 16**$\}$.

Paso 2. Modelado de reglas.

El segundo paso es generar la CCPN y la correspondiente matriz de conexión de la base de reglas normalizada.

Paso 2.1 Conversión de ECA - CCPN La CCPN de la base de reglas anterior se muestra en la figura 7.7. Como se puede observar, existen 16 transiciones tipo regla cada una de las cuales representa a una regla activa de la base de reglas normalizada. El índice de la transición indica la regla que representa, por ejemplo, la transición t_1 representa la **Regla 1**, la transición t_2 la **Regla**

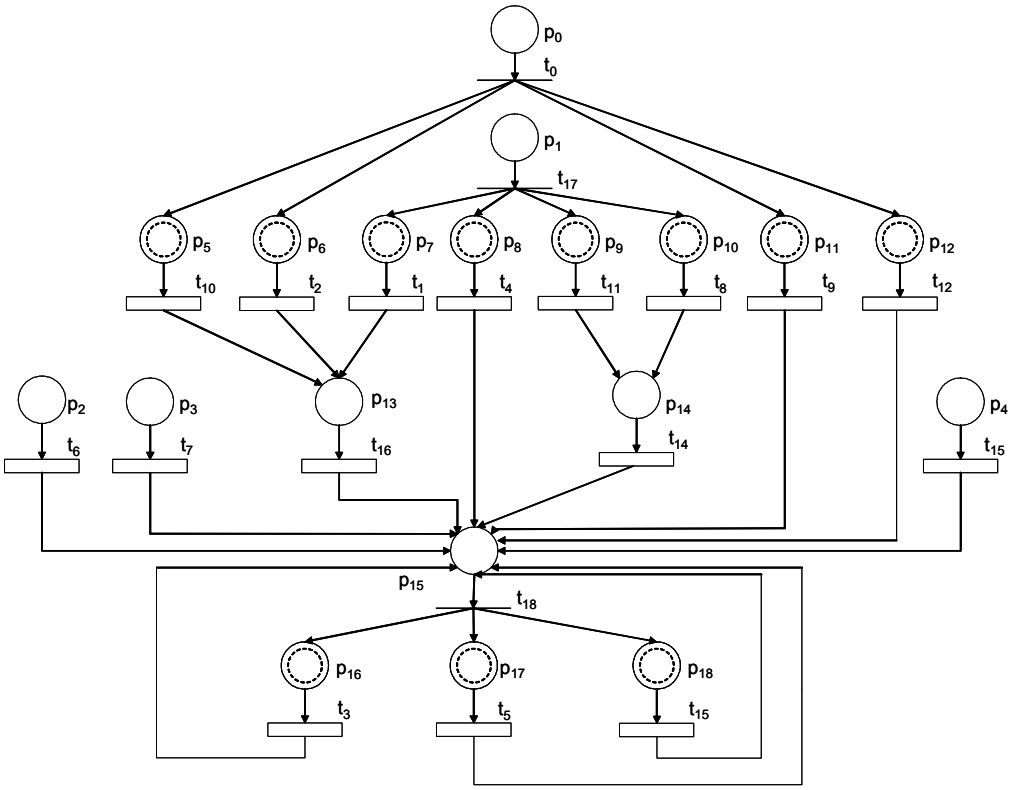


Figura 7.7: CCPN del ejemplo [37].

2 y así sucesivamente. Las transiciones t_0, t_{17} y t_{18} son transiciones tipo copia, útiles para modelar que dos o más reglas se disparan con el mismo evento.

Paso 2.2 Generación de la matriz de conexión La matriz de conexión de la CCPN mostrada en la figura 7.7 se muestra a continuación.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
0	-1	0	0	0	0	1	1	0	0	0	0	1	1	0	0	0	0	0	0
1	0	0	0	0	0	0	0	-1	0	0	0	0	0	0	0	1	0	0	0
2	0	0	0	0	0	0	-1	0	0	0	0	0	0	0	0	1	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	-1	0	0
4	0	0	0	0	0	0	0	0	-1	0	0	0	0	0	0	1	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	-1	0
6	0	0	-1	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
7	0	0	0	-1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
8	0	0	0	0	0	0	0	0	0	0	-1	0	0	0	1	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0	0	-1	0	0	1	0	0	0	0
10	0	0	0	0	0	-1	0	0	0	0	0	0	0	1	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	-1	0	0	0	0	0	1	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	0	-1	0	0	1	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	-1
14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-1	1	0	0	0
15	0	0	0	0	-1	0	0	0	0	0	0	0	0	0	0	1	0	0	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	-1	0	1	0	0	0
17	0	-1	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0
18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-1	1	1	1

Paso 3. Análisis de confluencia

A partir de la representación como una estructura de CCPN y su matriz de conexión de la base de reglas del ejemplo se realiza su análisis de confluencia. Hay que considerar que no se deben analizar las transiciones que se encuentren en los siguientes subconjuntos ya que representan reglas parcialmente redundantes condición - acción.

$S_1 = \{t_1, t_2, t_3\}$; $S_2 = \{t_4, t_5\}$; $S_3 = \{t_6, t_7\}$; $S_4 = \{t_8, t_9\}$; $S_5 = \{t_{11} - t_{16}\}$.

Paso 3.1 Detección de estructuras no confluentes en la CCPN. Lo primero que tenemos que hacer es detectar las estructuras que representen reglas que posiblemente no sean confluentes. Las tres estructuras que se deben identificar son: las transiciones que tienen el mismo lugar de

entrada, las transiciones que evalúan la misma condición y las transiciones que tienen el mismo lugar de salida.

A. Detección de reglas disparadas por el mismo evento Detectamos las transiciones tipo copia en la matriz de conexión.

Com se puede observar en la CCPN de la figura 7.7, $T_{copia} = \{t_0, t_{17}, t_{18}\}$.

Las transiciones tipo regla correspondientes a $T0$ son las siguientes: $T_{regla_{T0}} = \{t_2, t_9, t_{10}, t_{12}\}$.

Las transiciones tipo regla correspondientes a $T17$ son las siguientes: $T_{regla_{17}} = \{t_1, t_4, t_8, t_{11}\}$

Las transiciones tipo regla correspondientes a $T18$ son las siguientes: $T_{regla_{18}} = \{t_3, t_5, t_{13}\}$.

Las combinaciones que se deben verificar para $T_{regla_{10}}$ son las siguientes: $T_{analizar_{10}} = \{(t_2, t_9), (t_2, t_{10}), (t_2, t_{12}), (t_9, t_{10}), (t_9, t_{12}), (t_{10}, t_{12})\}$

Las combinaciones que se deben verificar para $T_{regla_{17}}$ son las siguientes: $T_{analizar_{17}} = \{(t_1, t_4), (t_1, t_8), (t_1, t_{11}), (t_4, t_8), (t_4, t_{11}), (t_8, t_{11})\}$

Las combinaciones que se deben verificar para $T_{regla_{17}}$ son las siguientes: $T_{analizar_{18}} = \{(t_3, t_5), (t_3, t_{13}), (t_5, t_{13})\}$

Eliminando los pares de transiciones cuyo resultado se puede obtener por el análisis de alguna transición de su subconjunto, tenemos que realizar las siguientes comparaciones para las reglas que son disparadas por el mismo evento:

$RME = \{(t_1, t_4), (t_1, t_8), (t_1, t_{11}), (t_2, t_{10}), (t_4, t_8), (t_4, t_{11}), (t_8, t_{11}), (t_9, t_{10}), (t_{10}, t_{12})\}$

B. Detección de reglas que evalúan la misma condición Analizando el dominio de cada condición, encontramos que las transiciones cuya condición puede evaluarse como verdadera en un momento determinado son las siguientes:

$Cond_1 = \{(t_1, t_4), (t_1, t_5), (t_2, t_4), (t_2, t_5), (t_3, t_4), (t_3, t_5)\}$

$Cond_2 = \{(t_1, t_8), (t_1, t_9), (t_2, t_8), (t_2, t_9), (t_3, t_8), (t_3, t_9)\}$

$Cond_3 = \{(t_{10}, t_{11}), (t_{10}, t_{12}), (t_{10}, t_{13}), (t_{10}, t_{14}), (t_{10}, t_{15}), (t_{10}, t_{16})\}$

Eliminando los pares de transiciones cuyo resultado se puede obtener por el análisis de alguna transición de su subconjunto, tenemos que realizar las siguientes comparaciones para las reglas que evalúan la misma condición, C_1, C_2 y C_3:

$Cond_1 = \{(t_1, t_4)\}$

Procediendo de la misma forma, tenemos

$Cond_2 = \{(t_1, t_8)\}$
$Cond_3 = \{(t_{10}, t_{11})\}$

C. Detección de reglas que ejecutan la misma acción Como se ve en la figura 7.7 los únicos lugares primitivos de salida que tienen más de un arco de entrada son p_{14} y p_{15}.

Las transiciones tipo regla de entrada para el lugar p_{14} son: $T_{regla_{E14}} = \{t_8, t_9\}$

No es necesario analizar el par en $T_{regla_{E14}}$ ya que estas transiciones representan reglas parcialmente redundantes condición - acción.

Las transiciones tipo regla de entrada para el lugar p_{15} son: $T_{regla_{E15}} = \{t_1, t_2, t_3, t_4, t_5, t_6, t_7, t_{11}, t_{12}, t_{13}, t_{14}, t_{15}, t_{16}\}$.

Dado que en T_{regla_E}tenemos transiciones que representan reglas parcialmente redundantes condición - acción podemos eliminar algunas transiciones:

$T_{regla_{E15}} = \{t_1, t_4, t_6, t_{11}\}$

Las combinaciones que se tienen que comparar son las siguientes: $T_{analizar_{e15}} = \{(t_1, t_4), (t_1, t_6), (t_1, t_{11}), (t_4, t_6), (t_4, t_{11}), (t_6, t_{11})\}$

Finalmente, uniendo los conjuntos de transiciones a analizar y eliminando los pares de transiciones cuyo resultado se puede obtener por el análisis de alguna transición de su subconjunto, tenemos que realizar las siguientes comparaciones para determinar si la base de reglas es confluente:

$Analizar = \{(t_1, t_4), (t_1, t_8), (t_1, t_{11}), (t_2, t_{10}), (t_4, t_8), (t_4, t_{11}), (t_8, t_{11}), (t_9, t_{10}), (t_{10}, t_{12}), (t_1, t_6), (t_4, t_6), (t_6, t_{11})\}$

Una vez obtenidas las transiciones que representan reglas con posibles problemas de no-confluencia, se debe analizar su interacción.

Paso 3.2 Análisis de la interacción de las reglas Para determinar si el proceso de ejecución de dos reglas es o no confluente, se tiene que analizar su interacción. Si las reglas no se activan/desactivan entre sí y sus acciones conmutan, entonces son confluentes.

Analizaremos sólo algunos de los pares de transiciones que encontramos en el punto anterior.

A. Análisis para las transiciones (t_1, t_4) Para analizar estas transiciones necesitamos obtener la información de $t_1.Condición$, $t_1^\bullet.Acción$ y $t_4.Condición$, $t_4^\bullet.Acción$.

$t_1.Condición$ = "insert.*balance* < 500 and insert.*rate* > 0"

$t_1^\bullet.Acción$ = "update `account` set *rate* = 0 where *balance* < 500 and *rate* > 0"

$t_4.Condición$ = "insert.*rate* > 1 and insert.*rate* < 2"

$t_4^\bullet.Acción$ = "update `account` set *rate* = 2 where *rate* > 1 and *rate* < 2"

Es fácil ver que:

$Dom(t_1.Condición)$ = {({`account`},{`account`.*rate*},{0 < `account`.*balance* < 500}), ({`account`}, {`account`.*rate*}, {0 < `account`.*rate* < $+\infty$})}

$Dom(t_4.Condición)$ = {({`account`},{`account`.*rate*},{1 < `account`.*rate* < 2})}

Y además, podemos separar $t_1^\bullet.Acción$ y $t_4^\bullet.Acción$ en los siguientes elementos:

Elementos de $t_2^\bullet.Acción$:

Comando = update

Relación = `account`

Atributo = *rate*

Valor = 0

Condición = *balance* < 500 and *rate* > 0

Elementos de $t_4^\bullet.Acción$:

Comando = update

Relación = `account`

Atributo = *rate*

Valor = 2

Condición = *rate* > 1 and *rate* < 2

En este caso, $Dom(t_1.Condición) = Dom((t_1^\bullet.Acción).Condición)$ y $Dom(t_4.Condición) = Dom((t_4^\bullet.Acción).Condición)$.

Además, dado que $Dom(t_1.Condición) \cap Dom(t_4.Condición) \neq \emptyset$ ya que 1 < `account`.*rate* < 2$\subset$0 < `account`.*rate* < $+\infty$ podemos continuar con el análisis.

1. Análisis de activación De acuerdo a la Definición 6.8, la regla r_i activa la regla r_j si la ejecución de a_i puede hacer que $Dom(c_j)$ cambie de $Dom(c_j) = \emptyset$ a $Dom(c_j) \neq \emptyset$. Para que esto suceda es necesario que se cumplan las siguientes condiciones:

1. a_i.Comando = "insert" o a_i.Comando = "update"

2. Rel $\neq \emptyset$

3. At $\neq \emptyset$

4. a_i.Valor $\in$ Cond

5. Dom $\neq \emptyset$

$t_1^\bullet.Acción$ no activa a $t_4.Condición$ ya que: $(t_1^\bullet.Acción)$.Comando = "update" (1), Rel = {account} (2), At = {rate} (3), $(t_1^\bullet.Acción)$.Valor = 0 y $0 \notin [1,2]$ y $Dom((t_1^\bullet.Acción).Condición) \cap Dom(t_4.Condición) \neq \emptyset$ (5)

Como se puede observar, la condición 4 no se cumple y $t_1^\bullet.Acción$ no activa a $t_4.Condición$.

Por otro lado,

$t_4^\bullet.Acción$ activa a $t_1.Condición$ ya que: $(t_4^\bullet.Acción)$.Comando = "update" (1), Rel = {`account`} (2), At = $\{rate\}$ (3), $(t_1^\bullet.Acción)$.Valor = 2 y $2 \in [0, +\infty]$ y $Dom((t_4^\bullet.Acción).Condición) \cap Dom(t_1.Condición) \neq \emptyset$ (5)

Como todas las condiciones se cumplen, $t_4^\bullet.Acción$ activa a $t_1.Condición$ .....(1)

2. Análisis de desactivación t_1 y t_4 no se desactivan entre sí porque ninguna ejecuta el comando "delete". .....(2)

3. Análisis de conmutatividad de acciones. Para ver si dos acciones conmutan es necesario que se cumplan las condiciones especificadas en la Definición 7.1.

$t_1^\bullet.Acción$ y $t_4^\bullet.Acción$ no conmutan porque no se pueden intercambiar ya que ambas modifican la misma información (todas las tuplas de la relación `account` cuyo valor de *rate* esté entre 1 y 2) con valores diferentes. .....(3)

4. Análisis de conmutatividad de reglas. Para que dos reglas conmuten es necesario que se cumplan las condiciones de la Definición 7.2. A partir de (1) y (3) podemos concluir que t_1 y t_4 no conmutan, es decir, la **Regla 1** y la **Regla 4** no conmutan.

Conclusión Como la **Regla 1** y la **Regla 4** no conmutan, entonces podemos concluir que los pares: (**Regla**1, **Regla**5) , (**Regla 2**, **Regla 4**), (**Regla 2**, **Regla 5**), (**Regla 3**, **Regla 4**) y (**Regla 3**, **Regla 5**) tampoco son confluentes.

Finalmente, la base de reglas del ejemplo 7.3 no es confluente.

El análisis del resto de las transiciones se lleva a cabo de manera similar. A continuación sólo mostramos las conclusiones a las que llegamos después de verificar cada par de transiciones.

B. Análisis para las transiciones (t_1, t_8) Las transiciones t_1 y t_8 representan reglas confluentes porque cada una de ellas modifica una relación diferente de la BD. Entonces los pares (**Regla 1**, **Regla 8**), (**Regla 2**, **Regla 8**), (**Regla 3**, **Regla 8**), (**Regla 1**, **Regla 9**), (**Regla 2**, **Regla 9**) y (**Regla 3**, **Regla 9**) también son confluentes.

C. Análisis para las transiciones (t_1, t_{11}) Las transiciones t_1 y t_{11} representan reglas confluentes porque sus condiciones no pueden ser verdaderas al mismo tiempo. De manera que en este caso no es ni siquiera necesario realizar el análisis de activación/desactivación y conmutatividad de acciones. Entonces los pares (**Regla 1**, **Regla 12**), (**Regla 1**, **Regla 13**), (**Regla 1**, **Regla 14**), (**Regla 1**, **Regla 15**), (**Regla 1**, **Regla 16**), (**Regla 2**, **Regla 11**), (**Regla 2**, **Regla 12**), (**Regla 2**, **Regla 13**), (**Regla 2**, **Regla 14**), (**Regla 2**, **Regla 15**), (**Regla 2**, **Regla 16**), (**Regla 3**, **Regla 11**), (**Regla 3**, **Regla 12**), (**Regla 3**, **Regla 13**), (**Regla 3**, **Regla 14**), (**Regla 3**, **Regla 15**) y (**Regla 3**, **Regla 16**) también son confluentes.

D. Análisis para las transiciones (t_2, t_{10}) Las transiciones t_2 y t_{10} representan reglas confluentes porque sus condiciones no pueden ser verdaderas al mismo tiempo. De manera que en este caso no es ni siquiera necesario realizar el análisis de activación/desactivación y conmutatividad de acciones. Entonces los pares (**Regla 2**, **Regla 10**) y (**Regla 3**, **Regla 10**) también son confluentes.

E. Análisis para las transiciones (t_4, t_8) Las transiciones t_4 y t_8 representan reglas confluentes porque cada una de ellas modifica una relación diferente de la BD. Entonces los pares (**Regla 4**, **Regla 8**), (**Regla 4**, **Regla 9**), (**Regla 5**, **Regla 8**) y (**Regla 5**, **Regla 9**) también son confluentes.

F. Análisis para las transiciones (t_4, t_{11}) Las transiciones t_4 y t_{11} representan reglas no confluentes porque ambas modifican las mismas tuplas de la relación account, pero con valores diferentes. De manera que las los pares (**Regla 4**, **Regla 11**), (**Regla 4**, **Regla 12**), (**Regla 4**, **Regla 13**), (**Regla 4**, **Regla 14**), (**Regla 4**, **Regla 15**), (**Regla 4**, **Regla 16**), (**Regla 5**, **Regla 11**), (**Regla 5**, **Regla 12**), (**Regla 5**, **Regla 13**), (**Regla 5**, **Regla 14**), (**Regla 5**, **Regla 15**) y (**Regla 5**, **Regla 16**) tampoco son confluentes.

G. Análisis para las transiciones (t_8, t_{11}) Las transiciones t_8 y t_{11} representan reglas confluentes porque cada una de ellas modifica una relación diferente de la BD. Entonces los pares (**Regla 8**, **Regla 11**), (**Regla 8**, **Regla 12**), (**Regla 8**, **Regla 13**), (**Regla 8**, **Regla 14**), (**Regla 8**, **Regla 15**), (**Regla 8**, **Regla 16**), (**Regla 9**, **Regla 11**), (**Regla 9**, **Regla 12**), (**Regla 9**, **Regla 13**), (**Regla 9**, **Regla 14**), (**Regla 9**, **Regla 15**) y (**Regla 9**, **Regla 16**) también son confluentes.

H. Análisis para las transiciones (t_9, t_{10}) Las transiciones t_9 y t_{10} representan reglas confluentes porque sus condiciones no pueden ser verdaderas al mismo tiempo. De manera que en este caso no es ni siquiera necesario realizar el análisis de activación/desactivación y conmutatividad de acciones. Entonces los pares (**Regla 8**, **Regla 10**) y (**Regla 9**, **Regla 10**) también son confluentes.

I. Análisis para las transiciones (t_{10}, t_{12}) Las transiciones t_{10} y t_{12} no son confluentes porque $t_{10}^{\bullet}$.Acción modifica información que $t_{12}^{\bullet}$.Acción verifica. Entonces los pares (**Regla 10**, **Regla 11**), (**Regla 10**, **Regla 12**), (**Regla 10**, **Regla 13**), (**Regla 10**, **Regla 14**), (**Regla 10**, **Regla 15**) y (**Regla 10**, **Regla 16**) tampoco son confluentes.

J. Análisis para las transiciones (t_1, t_6) Las transiciones t_1 y t_6 son confluentes porque el dominio de sus condiciones hace que modifiquen tuplas diferentes. Entonces los pares (**Regla 1**, **Regla 6**), (**Regla 2**, **Regla 6**), (**Regla 3**, **Regla 6**), (**Regla 1**, **Regla 7**), (**Regla 2**, **Regla 7**) y (**Regla 3**, **Regla 7**) son confluentes.

K. Análisis para las transiciones (t_4, t_6) Las transiciones t_4 y t_6 representan reglas no confluentes porque ambas modifican las mismas tuplas de la relación `account`, pero con valores

diferentes. De manera que las los pares (**Regla 4**, **Regla 6**), (**Regla 4**, **Regla 7**), (**Regla 5**, **Regla 6**) y (**Regla 5**, **Regla 7**) tampoco son confluentes.

L. Análisis para las transiciones (t_6, t_{11}) Las transiciones t_6 y t_{11} representan reglas confluentes porque sus condiciones no pueden ser verdaderas al mismo tiempo. De manera que en este caso no es ni siquiera necesario realizar el análisis de activación/desactivación y conmutatividad de acciones. Entonces los pares (**Regla 6**, **Regla 11**), (**Regla 6**, **Regla 12**), (**Regla 6**, **Regla 13**), (**Regla 6**, **Regla 14**), (**Regla 6**, **Regla 15**), (**Regla 6**, **Regla 16**), (**Regla 7**, **Regla 11**), (**Regla 7**, **Regla 12**), (**Regla 7**, **Regla 13**), (**Regla 7**, **Regla 14**), (**Regla 7**, **Regla 15**) y (**Regla 7**, **Regla 16**) también son confluentes.

De acuerdo a los resultados que obtuvimos, concluimos que la base de reglas del ejemplo 7.3 no es confluente.

Análisis de Terminación El análisis de terminación coincide con el error de circularidad descrito en los capítulos 5 y 6.

El primer paso para detectar este error es identificar la estructura de error, en este caso, los ciclos que existen en la CCPN. En la figura 7.7 existen las siguientes secuencias de disparo circulares: $(p_{15}, t_3) \rightarrow (t_3, p_{15})$, $(p_{15}, t_5) \rightarrow (t_5, p_{15})$ y $(p_{15}, t_{13}) \rightarrow (t_{13}, p_{15})$. Las transiciones que debemos analizar para saber si el ciclo termina son t_3, t_5 y t_{13}. Esto quiere decir que existen autociclos en la base de reglas.

El segundo paso en el análisis de terminación es determinar si las transciones anteriores se activan a sí mismas.

Vamos a realizar el análisis de activación para la transición t_3.

De acuerdo al análisis de terminación descrito, debemos verificar en este caso si la regla representada por la transición t_3 se activa a sí misma. La condición t_3 es la siguiente $t_3.Condición =$ `account`.*balance* < 500 and `account`.*rate* > 0. La acción de t_3 es $t_3^{\bullet}.Acción =$ update `account` set *rate* $= 0$ where *balance* < 500 and *rate* > 0. Como se puede observar, tanto t_3.Condición como $t_3^{\bullet}.Acción$ tienen un atributo es común, `account`.*rate*, ambas tienen el mismo dominio, pero el valor que se asigna al atributo `account`.*rate* no pertenece al dominio de la condición que verifica ese atributo, es decir, `account`.*rate* $= 0$ no pertenece a $0 <$ `account`.*rate* $< +\infty$. Por lo tanto, el proceso de ejecución de este ciclo termina. Lo mismo sucede con el proceso de las transiciones t_5 y

t_{13}.

Como los ciclos detectados terminan,el proceso de ejecución de las reglas del ejemplo 7.3 termina.

7.3. Comentarios finales

Las propiedades de confluencia y terminación de una base de reglas nos ayudan a determinar su comportamiento en tiempo de ejecución. Estas propiedades son relevantes ya que, por un lado, mantienen la consistencia del sistema, es decir, que siempre se obtenga el mismo resultado ante la misma situación y, por otro lado, aseguran que el proceso de ejecución de las reglas se realiza en un número finito de pasos.

En este capítulo mostramos el desarrollo de un método que permite evaluar si una base de reglas posee esas propiedades. Nuestro método se efectúa en tres etapas, primero se normaliza la base de reglas para simplificar nuestro análisis. Segundo, la base de reglas normalizada se representa como un modelo de CCPN y se identifican las estructuras que indican no-confluencia y no-terminación. En el tercer paso, se analiza la interacción de las reglas para determinar si se cumplen o no tales propiedades. Nuestro método no necesita analizar todos los pares de reglas para realizar la evaluación de las propiedades, esta es nuestra gran ventaja. Como se observó a lo largo del capítulo, para una base de reglas con cinco reglas era necesario realizar seis comparaciones para examinar la propiedad de confluencia. Con nuestro método, sólo realizamos tres comparaciones para revisar el cumplimiento de esta propiedad.

Capítulo 8

Desarrollo de ECAPNVerifier

Dado que hay casos en los cuales una base de reglas consiste de varias reglas, desarrollamos una herramienta que verifica una base de reglas automáticamente. Esta herramienta la denominamos ECAPNVerifier.

ECAPNVerifier se integró a la herramienta ECAPNSim, la cual permite representar una base de reglas como un modelo de CCPN y genera su correspondiente matriz de conexión. Para situar nuestra herramienta en el sistema ECAPNSim su mostramos la arquitectura en la figura 8.1.

El módulo "Editor de reglas" permite al usuario escribir reglas con la sintaxis general `ON - IF. THEN`. Entonces, el módulo "Convertidor ECA - CCPN" genera automáticamente la CCPN de la base de reglas introducidas así como su correspondiente matriz de conexión. El módulo de "Verificación de reglas" usa la matriz de conexión generada para detectar las transiciones que representan reglas que probablemente tengan errores y problemas de no-confluencia y no-terminación. Posteriormente, en este mismo módulo se analiza la interacción de las transiciones detectadas y se genera una lista con las reglas que contienen errores y el tipo de estos. También se genera un reporte de las reglas que no son confluentes y/o no terminan. La última acción que realiza este módulo es proporcionar al usuario sugerencias para la corrección de los errores encontrados. El módulo "Editor de CCPN" muestra la CCPN generada y el módulo "Componente de explicación" muestra al usuario la forma en la que interactúan las reglas.

El núcleo de ECAPNSim contiene módulos para manejar las reglas ECA y la ocurrencia de eventos en tiempo de ejecución. No ahondaremos en estos módulos ya que nuestros métodos de

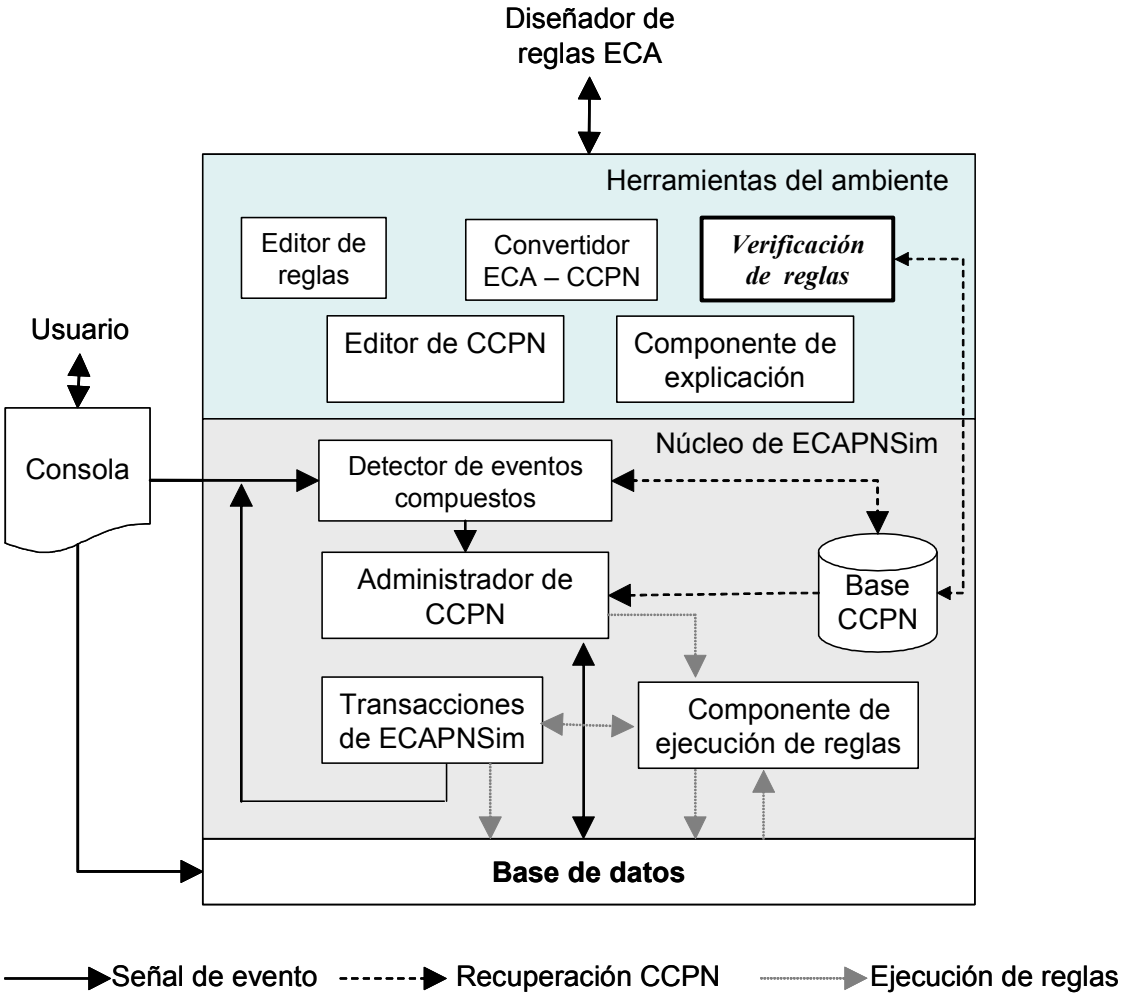


Figura 8.1: Arquitectura de ECAPNSim. ECAPNVerifier se integra en esta arquitectura

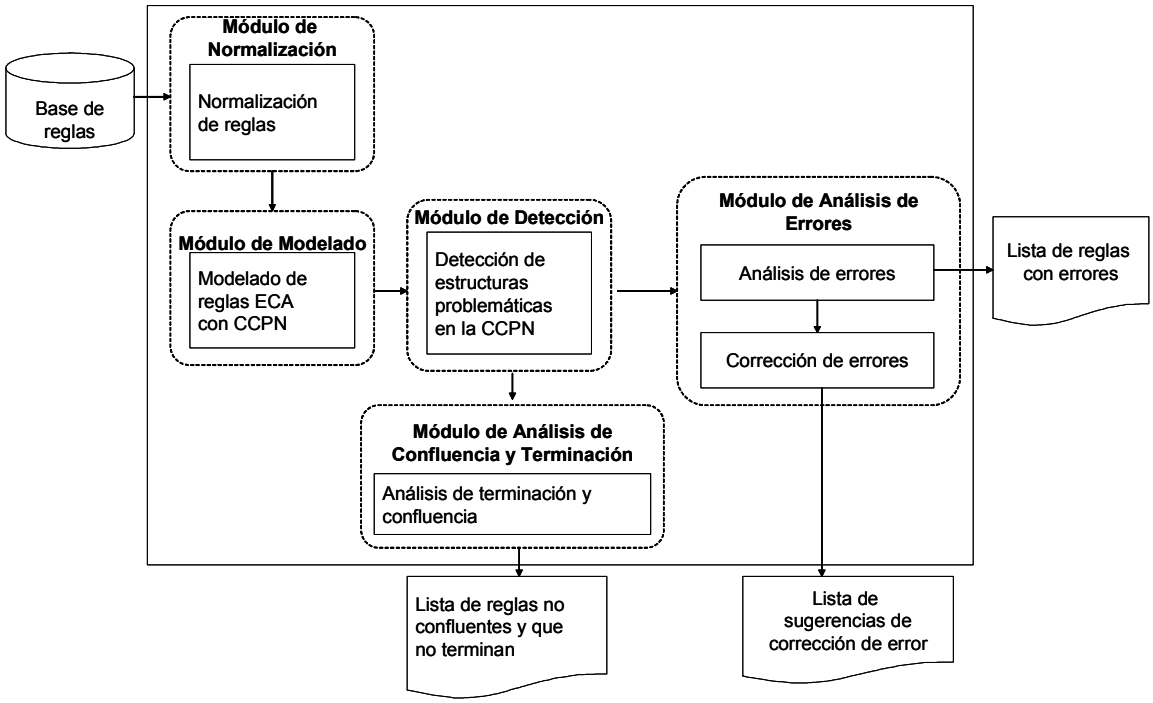


Figura 8.2: Arquitectura de ECAPNVerifier

análisis no lo requieren.

8.1. Arquitectura de ECAPNVerifier

ECAPNVerifier consiste de cinco módulos principales, los cuales se muestran en la figura 8.2. A continuación describimos cada módulo:

1. Módulo de Normalización.

   Este módulo toma la base de reglas original y dadas las características de las reglas ECA que empleamos, aplica leyes del álgebra de Boole para transformar el evento y la condición de la regla a su forma normal disyuntiva. Posteriormente, divide cada regla para obtener reglas atómicas.

   El módulo cuenta con dos métodos principales:

- FND( ). Transforma el evento/condición en su forma normal disyuntiva.
- NORMALIZA( ). Separa los elementos obtenidos con FND( ) y las acciones de cada regla para formar reglas atómicas.

2. Módulo de Modelado.

   Este módulo construye una CCPN y su matriz de conexión a partir de una base de reglas. Contiene dos métodos principales:

   - GENERA_CCPN( ). Este método toma la base de reglas atómica obtenida por el módulo Normalización y genera su modelo de CCPN. El algoritmo que implementa este método está descrito en la figura 4.6.
   - GENERA_MATRIZINCIDENCIA( ). Este método genera la matriz de conexión de la CCPN generada por el método GENERA_CCPN( ). La matriz de conexión se crea de acuerdo a las reglas que se describieron en la Definición 4.4.

3. Módulo de Detección.

   Este módulo identifica las estructuras de error y las estructuras que representan reglas con problemas de no-confluencia y no-terminación usando la matriz de conexión obtenida en el módulo de Modelado. Cuenta con los siguientes métodos:

   - REGLAS_MISMA_ACCION( ). Este método obtiene las transiciones tipo regla que tienen el mismo lugar primitivo de salida, es decir, las reglas que ejecutan la misma acción.
   - OBTENER_LUGARES_INCIDENTES( ). Recibe como entrada una matriz $A_{m\times n}$. Para cada columna obtiene el número de entradas positivas usando el método BUSCAPOSITIVOS( ). Si para la i-ésima columna este número es mayor a 1, significa que varias transiciones tipo regla tienen a pi como lugar de salida. Si cuando se llama a este método se envía $A^T_{m\times n}$ se obtienen las transiciones que tipo copia que sirven para duplicar lugares.
   - OBTENER_TRANSICION_ENTRADA( ). Recibe como entrada una matriz $A_{m\times n}$ y un índice p que pertenece a un conjunto P_{prim}. Si P_{prim} contiene los índices de los lugares primitivos, este método busca las entradas positivas en la columna p. Dado que un lugar primitivo sólo puede ser lugar de salida de una transición tipo regla, es suficiente

encontrar los valores positivos en la columna p para obtener los índices de las filas que representan las transiciones tipo regla de entrada del lugar representado por esta columna. Si el conjunto P_{prim} que se envía a este método contiene los índices de las transiciones tipo regla, entonces se obtiene el índice del lugar primitivo de salida de la transición representada por la fila p.

- BUSCA_POSITIVOS( )/BUSCA_NEGATIVOS( ). Recibe como entrada una matriz $A_{m\times n}$ y un índice, p. Si cuando se llama a este método se envía la matriz de conexión $A_{m\times n}$ y el índice de una columna, busca los valores positivos/(negativos) en todas las filas de la columna p. Si se envía $A^T_{m\times n}$ y el índice de una transición, se buscan los valores positivos/(negativos) en todas las columnas de la fila p.
- REGLAS_MISMO_EVENTO( ). Este método obtiene las transiciones tipo regla cuyo lugar primitivo de entrada es duplicado por medio de una transición tipo copia, es decir, se identifican las reglas que son disparadas por el mismo evento.
- OBTENER_LUGARES_DUPLICADOS( ). Recibe como entrada una matriz Amn y envía $A^T_{m\times n}$ al método OBTENER_LUGARES_INCIDENTES( ) para obtener las transiciones tipo copia.que duplican lugares de entrada. Para cada transición identificada busca su entrada negativa para obtener el índice del lugar que se está duplicando.
- OBTENER_TRANSICION_SALIDA( ). Recibe una matriz $A_{m\times n}$, el índice de un lugar, p, y el conjunto de índices de las transiciones tipo regla. Proporciona la primera transición tipo regla que se encuentra a partir del lugar p. Si lo que se envía es $A^T_{m\times n}$, p que es el índice de una transición y el conjunto de índices de lugares primitivos, se obtienen los lugares de entrada primitivos de la transición p.
- REGLAS_INCONSISTENCIA( ). Este método obtiene las transiciones tipo regla cuyos lugares de salida representan acciones contradictorias.
- LUGAR( ). Regresa el índice del lugar donde se encuentra almacenado(a) un(a) evento(acción).
- REGLA_AISLADA( ) Este método obtiene las transiciones que representan reglas aisladas.
- REGLA_PUNTO_MUERTO( ). Este método indaga en la matriz de conexión para encontrar las reglas de punto muerto.

- Secuencia_Disparo( ). Este método busca alternadamente valores positivos y negativos en la matriz de conexión hasta que encuentra una columna que no tiene valores negativos o hasta que identifica que el índice de un lugar o transición tipo regla ya se encuentra en la secuencia.
- Dominio( ). Este método encuentra el dominio de una condición compleja escrita en SQL. Usa una gramática para identificar el tipo de consulta escrito y dependiendo del resultado realiza las transformaciones correspondientes en álgebra relacional.
- Verificar_Dominio( ). Este método recibe dos condiciones y determina cuál es la relación entre ellas. Verifica si son iguales, diferentes o si alguna de ellas está contenida en la otra.
- Condiciones( ). Este método toma una expresión en álgebra relacional y obtiene las condiciones de los operadores que lo requieran, por ejemplo, selección y reunión.
- Invariantes( ). Este método permite encontrar los ciclos en la matriz de conexión.

  Los algoritmos de cada uno de estos métodos se describieron en capítulos anteriores.

  La salida de este módulo es un conjunto de transiciones que deben ser analizadas posteriormente junto con el error que se debe verificar.

4. Módulo de Análisis de Errores.

   Este módulo toma las transiciones identificadas previamente y las analiza para determinar si tienen algún error. Después de encontrar los errores proporciona al usuario algunas sugerencias para corregirlos. La salida de este módulo es una lista de errores y una lista de sugerencias para corregirlos.

   Cuenta con los siguientes métodos:

   - Redundancia( ). Verifica que se cumplan las condiciones del error de redundancia y sus subtipos.
   - Inconsistencia( ). Verifica que se cumplan las condiciones del error de inconsistencia.
   - Incompletitud( ). Verifica que se cumplan las condiciones del error de incompletitud.
   - Circularidad( ). Verifica que se cumplan las condiciones del error de circularidad.

- Activacion( ). Determina si una acción activa a una condición.
- Desactivacion( ). Determina si una acción desactiva a una condición.
- Corregir( ). Este método determina algunas sugerencias de corrección de errores.

5. Módulo de Análisis de Confluencia y Terminación.

   Este módulo analiza las transiciones identificadas con posibles problemas de no-confluencia y no-terminación y verifica si tienen o no este problema. La salida de este módulo es una lista de reglas indicando si tienen algún problema y el tipo de este. Cuenta con los siguientes métodos:

   - Conmutatividad_Acciones( ). Verifica si dos acciones pueden conmutar.
   - Conmutatividad_Reglas( ). Verifica si dos reglas pueden conmutar.
   - Confluencia( ). Verifica si dos transiciones cumplen las condiciones para tener problemas de no-confluencia.
   - Terminacion( ). Verifica si dos transiciones cumplen las condiciones para tener problemas de no-terminación.

8.2. Ejemplo

A continuación mostraremos el funcionamiento de ECAPNVerifier usando el caso de estudio de la sección 6.5. La figura 8.3 muestra la CCPN de la base de reglas. Cuando se selecciona en el memú principal de ECAPNSim la opción "Verification" se invoca nuestro sistema ECAPNVerifier y se ejecutan los pasos descritos en la figura 6.1. Si después de la normalización de reglas se generan más, se muestra la nueva CCPN. En este caso, todas las reglas son atómicas, por lo tanto, no se muestra otra CCPN. La matriz de conexión también se genera automáticamente dado que se usa como entrada al módulo Detección. La matriz de conexión de la CCPN en la figura 8.3 se muestra en la figura 8.4.

El módulo Detección toma la matriz de conexión e identifica todas las estucturas que indican algún tipo de error. Posteriormente el módulo Análisis de Errores toma las transiciones identificadas y analiza su interacción para concluir si tienen algún error. La lista de errores detectados se muestra en la figura 8.5.

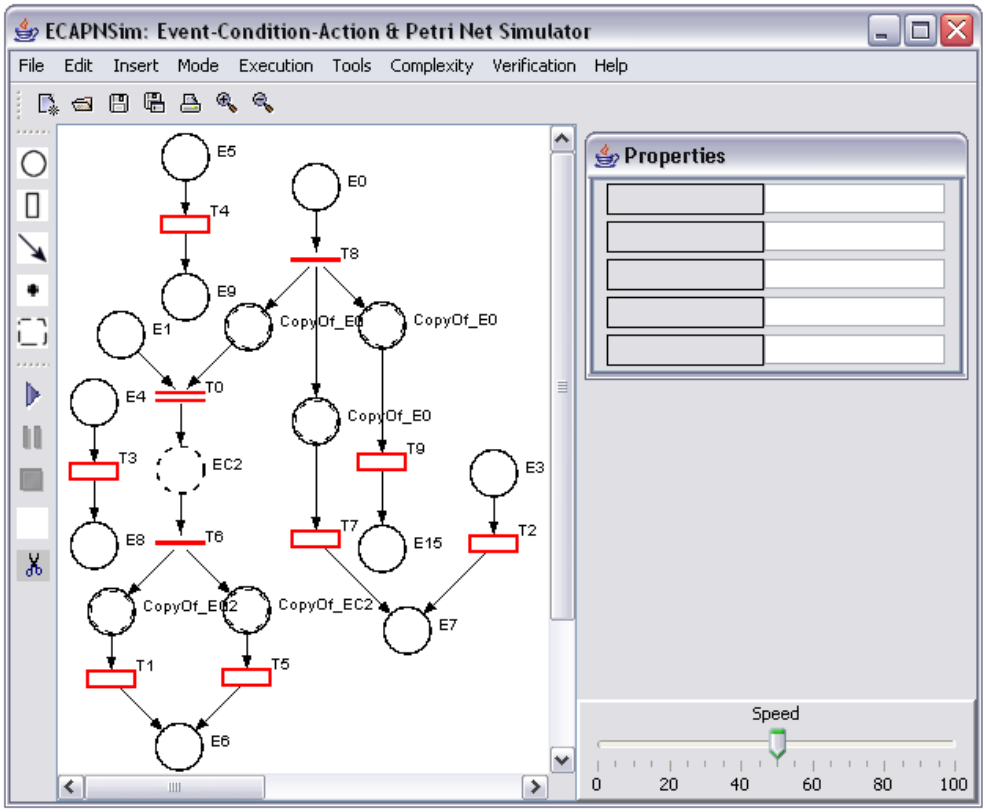
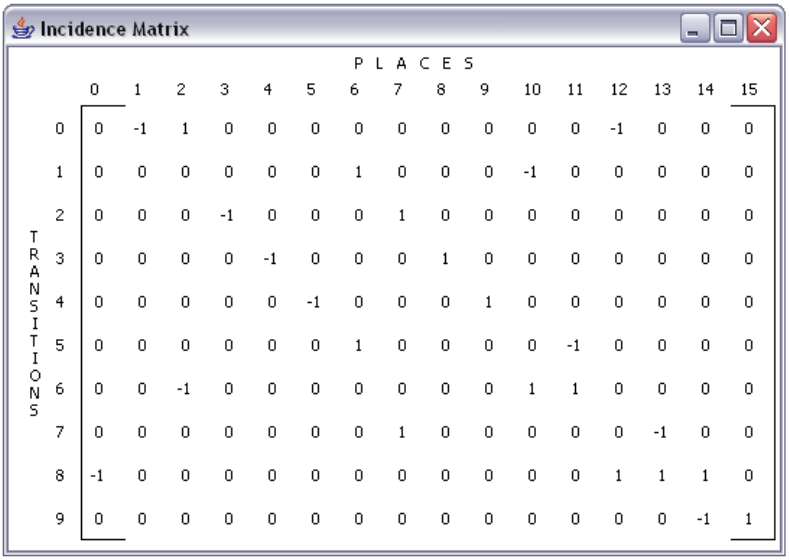


Figura 8.3: CCPN de la base de reglas activas descrita en la Sección 6.5



	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	-1	1	0	0	0	0	0	0	0	0	0	-1	0	0	0
1	0	0	0	0	0	0	1	0	0	0	-1	0	0	0	0	0
2	0	0	0	-1	0	0	0	1	0	0	0	0	0	0	0	0
3	0	0	0	0	-1	0	0	0	1	0	0	0	0	0	0	0
4	0	0	0	0	0	-1	0	0	0	1	0	0	0	0	0	0
5	0	0	0	0	0	0	1	0	0	0	0	-1	0	0	0	0
6	0	0	-1	0	0	0	0	0	0	0	1	1	0	0	0	0
7	0	0	0	0	0	0	0	1	0	0	0	0	0	-1	0	0
8	-1	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0
9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	-1	1

Figura 8.4: Matriz de incidencia de la CCPN de la figura 8.3

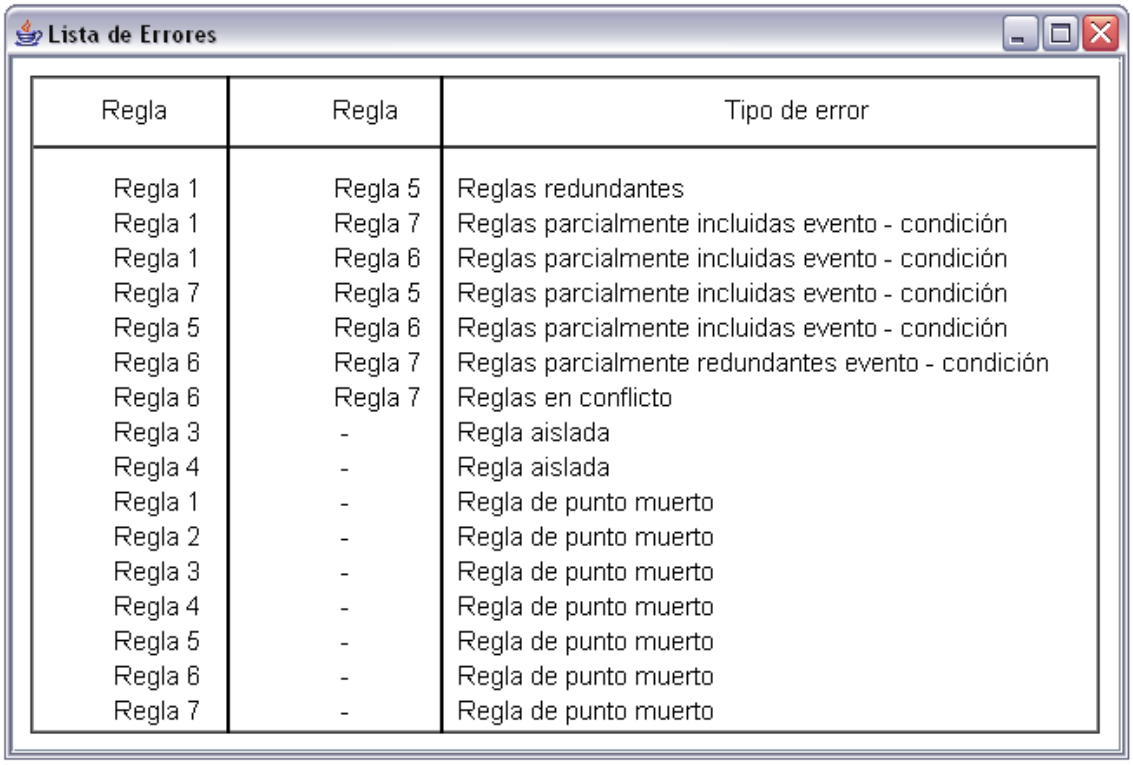
Finalmente, se muestran algunas sugerencias para corregir los errores detectados. Estas sugerencias se muestran en la figura 8.6.

Las correcciones ejecutadas tomando en cuenta las sugerencias, se mostraron en la Sección 6.5. Las correcciones no se ejecutan automáticamente.

La figura 8.7 muestra la CCPN que genera ECAPNSim de la base de reglas del ejemplo 7.1. Existen cuatro transiciones tipo regla que representan a cada una de las reglas activas de ese ejemplo.

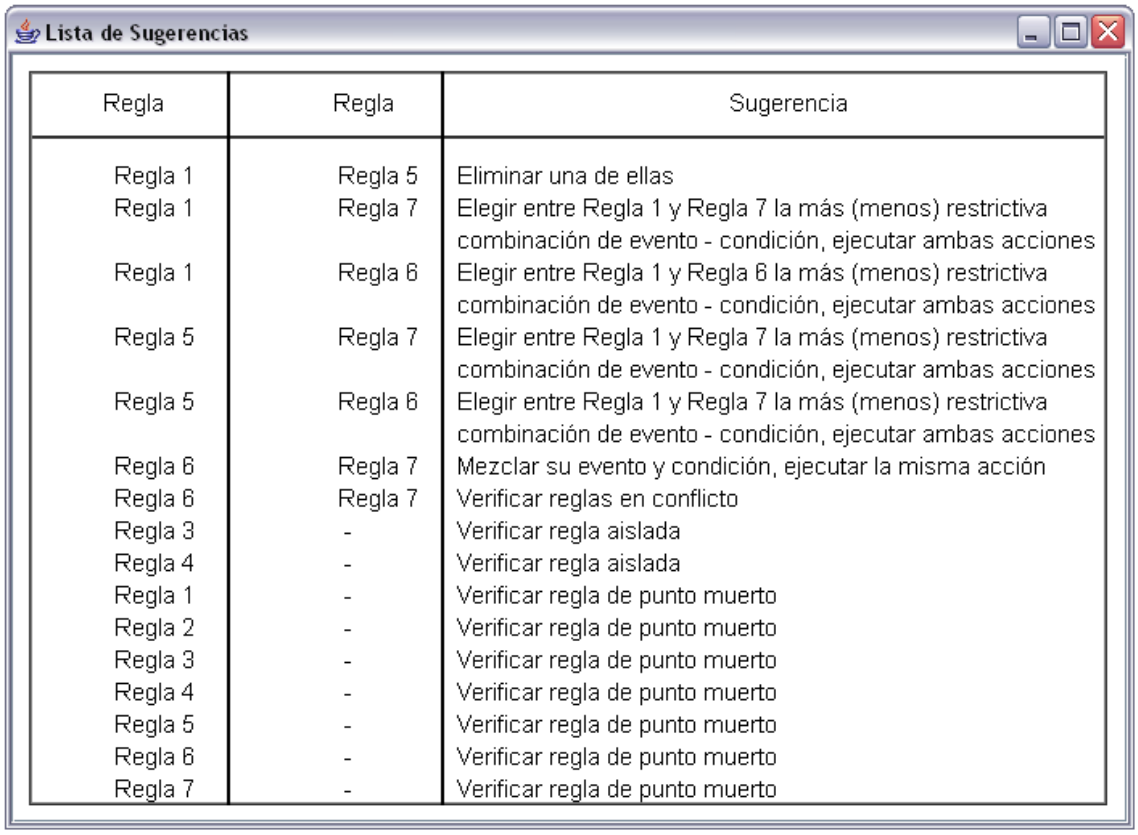
La matriz de conexión de la CCPN de la figura 8.7 se muestra en la figura

La figura 8.9 muestra el resultado del análisis de confluencia y terminación. La base de reglas no es confluente, pero su proceso de ejecución sí termina. También se muestran las reglas que conmutan/no conmutan y las que su proceso de ejecución termina/no termina.



Regla	Regla	Tipo de error
Regla 1	Regla 5	Reglas redundantes
Regla 1	Regla 7	Reglas parcialmente incluidas evento - condición
Regla 1	Regla 6	Reglas parcialmente incluidas evento - condición
Regla 7	Regla 5	Reglas parcialmente incluidas evento - condición
Regla 5	Regla 6	Reglas parcialmente incluidas evento - condición
Regla 6	Regla 7	Reglas parcialmente redundantes evento - condición
Regla 6	Regla 7	Reglas en conflicto
Regla 3	-	Regla aislada
Regla 4	-	Regla aislada
Regla 1	-	Regla de punto muerto
Regla 2	-	Regla de punto muerto
Regla 3	-	Regla de punto muerto
Regla 4	-	Regla de punto muerto
Regla 5	-	Regla de punto muerto
Regla 6	-	Regla de punto muerto
Regla 7	-	Regla de punto muerto

Figura 8.5: Lista de errores de la base de reglas ECA del caso de estudio de la Sección 6.5



Regla	Regla	Sugerencia
Regla 1	Regla 5	Eliminar una de ellas
Regla 1	Regla 7	Elegir entre Regla 1 y Regla 7 la más (menos) restrictiva combinación de evento - condición, ejecutar ambas acciones
Regla 1	Regla 6	Elegir entre Regla 1 y Regla 6 la más (menos) restrictiva combinación de evento - condición, ejecutar ambas acciones
Regla 5	Regla 7	Elegir entre Regla 1 y Regla 7 la más (menos) restrictiva combinación de evento - condición, ejecutar ambas acciones
Regla 5	Regla 6	Elegir entre Regla 1 y Regla 7 la más (menos) restrictiva combinación de evento - condición, ejecutar ambas acciones
Regla 6	Regla 7	Mezclar su evento y condición, ejecutar la misma acción
Regla 6	Regla 7	Verificar reglas en conflicto
Regla 3	-	Verificar regla aislada
Regla 4	-	Verificar regla aislada
Regla 1	-	Verificar regla de punto muerto
Regla 2	-	Verificar regla de punto muerto
Regla 3	-	Verificar regla de punto muerto
Regla 4	-	Verificar regla de punto muerto
Regla 5	-	Verificar regla de punto muerto
Regla 6	-	Verificar regla de punto muerto
Regla 7	-	Verificar regla de punto muerto

Figura 8.6: Lista de sugerencias

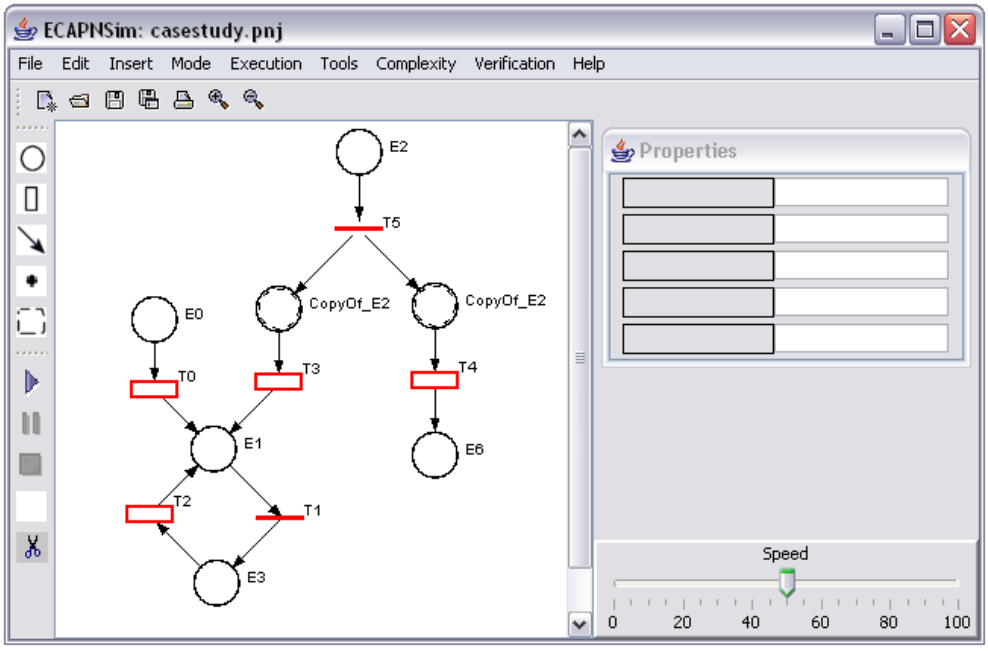


Figura 8.7: CCPN de la base de reglas del ejemplo 7.1

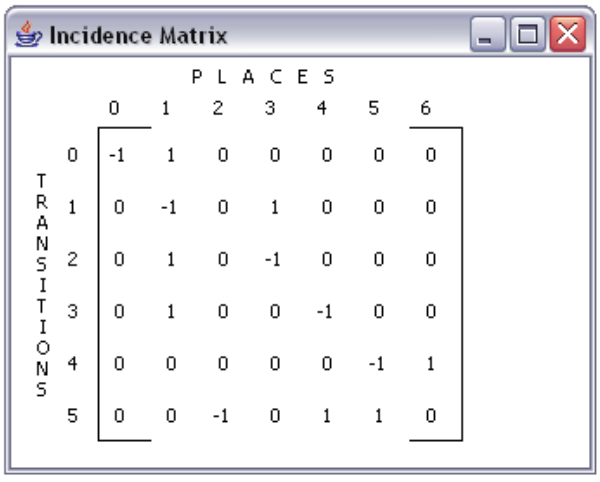


Figura 8.8: Matriz de incidencia de la CCPN de la figura 8.7

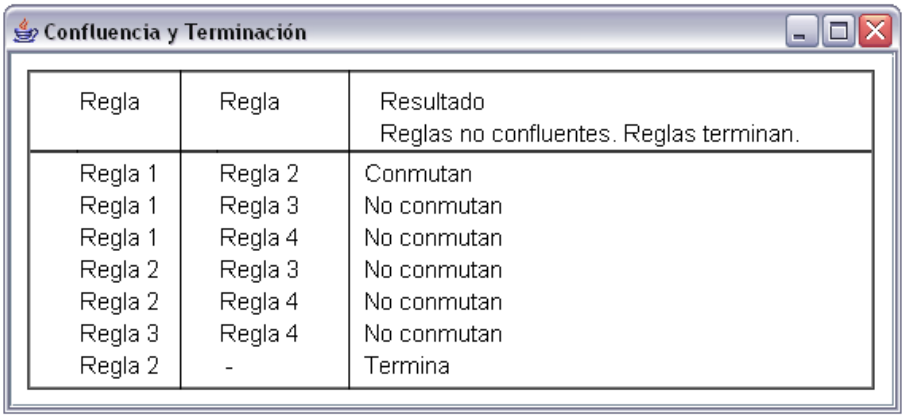


Figura 8.9: Resultado del análisis de las propiedades de confluencia y terminación

8.3. Comentarios finales

ECAPNVerifier permite verificar los errorses contenidos en una base de reglas así como deteminar si esta posee las propiedades de confluencia y terminación. En esta herramienta se implementan los algoritmos descritos a lo largo de los capítulos 6 y 7.

ECAPNVerifier se añadió a la herramienta ECAPNSim para complementar su funcionalidad. Ahora ECAPNSim no sólo puede representar una base de reglas ECA, obtener su matriz de conexión y convertir una BD pasiva en una BDA a través de la ejecución de las reglas, sino que además, puede verificar que antes de poner a funcionar el sistema, se hayan detectado los errores en la base de reglas y determinado si el proceso de ejecución terminará y será confluente. Si se detectan errores en la base de reglas se pueden tomar diferentes acciones para corregirlos.

Capítulo 9

Conclusiones y Trabajo Futuro

En este capítulo presentamos de forma general los resultados que obtuvimos con el desarrollo de este trabajo de tesis. Asimismo, describimos las actividades que se pueden seguir a fin de mejorarlo.

9.1. Discusión de resultados

En este trabajo de tesis se abordaron los problemas de *verificación de errores* y *análisis de confluencia y terminación* de reglas ECA, los cuales son fundamentales para desarrollar bases de reglas que cumplan correctamente con el propósito para el que fueron diseñadas.

Los tipos de errores verificados son *redundancia*, *inconsistencia*, *incompletitud* y *circularidad*. La definición de estos errores fue introducida originalmente en el contexto de reglas de producción, las cuales se pueden ver como un caso particular de las reglas ECA ya que sólo consisten de la parte de la condición y acción. Así que uno de los primeros retos que tuvimos que enfrentar fue redefinir estos conceptos para considerar el elemento del evento presente en las reglas ECA. En la redefinición que hicimos están presentes todos los elementos de las reglas ECA y, además, consideramos la interacción entre ellas para hacer más precisas las definiciones. Así, por ejemplo, para que exista un error de circularidad no es suficiente con que se tenga una secuencia de disparo de reglas circular r_i, r_j, ..., r_p, r_i sino que también r_i active r_j y así sucesivamente hasta que r_p active a r_i. Si alguna regla r_k activa a r_{k+1} pero no la activa, entonces aunque exista un ciclo entre las reglas su proceso de ejecución terminará.

En las nuevas definiciones también aparecen algunos subtipos de errores que son útiles para preveer posibles situaciones problemáticas. Tal es el caso de las reglas en conflicto potencial, que en un cierto momento pueden causar comportamientos anómalos de la base de reglas. Conociendo estas situaciones se pueden generar planes de contingencia para resolver el conflicto en caso de que este ocurra.

Hasta donde tenemos conocimiento, no existen definiciones de los errores anteriores en el contexto de reglas activas, de manera que esta es una aportación original del área.

Una vez obtenidas las definiciones de error, nos dimos a la tarea de diseñar un método capaz de detectar esos errores. El método que desarrollamos está basado en una extensión de las redes de Petri llamada Red de Petri Coloreada Condicional (CCPN, por sus siglas en inglés) porque representa claramente cada parte de las reglas ECA y la interacción de las mismas. Además, cuenta con técnicas de análisis que permiten examinar características relevantes del sistema modelado. Nuesto método de verificación de errores puede resumirse de la siguiente manera: primero, normalizar las reglas, segundo, representar las reglas como una CCPN y obtener su matriz de conexión, tercero, analizar la estructura de CCPN en busca de patrones que indiquen la presencia de algún error y extraer las reglas involucradas en tales estructuras y, cuarto, analizar la interacción de las reglas extraídas para concluir acerca de la presencia de errores. Realizar el proceso de verificación de esta manera, nos da una ventaja muy importante: no es necesario analizar todas las reglas del conjunto para determinar si existe algún error. Por ejemplo, supongamos que en la base existen dos reglas disparadas por eventos diferentes, que evalúan condiciones diferentes y ejecutan acciones sobre distintos porciones de la información del sistema. Claramente, estas reglas no tendrían ningún problema de redundancia, entonces no deberían ser analizadas. Lo que hace nuestro método es evitar este tipo de análisis y centrarse en revisar sólo aquellas reglas que pudieran tener algún error. Por otro lado, dado que la matriz de conexión no representa la información semántica de las reglas desarrollamos un método basado en el concepto que denominamos dominio de una condición para analizar precisamente el tipo de interacción de las reglas. Mediante este método, condiciones que podrían parecer a simple vista difíciles de evaluar, se transforman en condiciones simples que pueden ser evaluadas fácilmente.

Una de las limitaciones principales de nuestro método de verificación está asociado precisamente con la representación de las reglas como una estructura de CCPN. Entre más reglas existan en la

base de reglas, mayor será el tamaño de la red (y será difícil de visualizar) y las dimensiones de la matriz de conexión. Además, debemos recordar que se introducen lugares y transiciones especiales siempre que dos o más reglas se disparan por la ocurrencia del mismo evento o cuando existen eventos compuestos. Otras limitaciones están asociadas con la estructura de las reglas, si, por ejemplo, todas las reglas se disparan por la ocurrencia del mismo evento o todas las reglas ejecutan la misma acción, entonces no obtendremos mucho ahorro en cuanto al número de evaluaciones que se tienen que hacer para formular conclusiones acerca de la existencia de errores.

Respecto al análisis de las propiedades de confluencia y terminación de la base de reglas, también desarrollamos un método para examinar estas características basado en la CCPN. Nuestro método es similar al descrito para la verificación de errores, sólo que en lugar de identificar patrones de errores en la CCPN, detectamos estructuras que puedan ser no-confluentes y, posteriormente, analizamos la interacción de las reglas involucradas en tales patrones. La propiedad de terminación se puede ver como el error de circularidad.

Los resultados que obtuvimos con estos métodos fueron altamente satisfactorios ya que en un conjunto de 16 reglas, en donde otros métodos tienen que hacer 120 evaluaciones para determinar si su proceso de ejecución es confluente, nosotros sólo tuvimos que realizar 12 evaluaciones para formular conclusiones acerca de su propiedad de confluencia..

Una vez más, una de las limitaciones de este enfoque está asociada con la estructura de las reglas. Si, en el peor de los casos, todas las reglas evalúan la misma condición entonces tendremos que examinar todos los pares de reglas. Esto nos colocaría en el mismo nivel, en cuanto al número de evaluaciones, de los trabajos que resuelven este problema actualmente.

En conclusión, este trabajo permite obtener resultados similares a los reportados en la literatura, pero en un número menor de evalauciones.

9.2. Trabajo futuro

Las actividades que proponemos para posteriores mejoras a este trabajo giran alrededor de los siguientes puntos:

1. Revisar las definiciones de error.

   En este momento consideramos que una reglas aislada es una característica del error de

incompletitud. Sin embargo, en algunas aplicaciones es posible que no exista una única regla aislada como tal, sino que se tenga un subconjunto de reglas conectadas entre sí y que, a su vez, no tenga ninguna interacción con el resto de las reglas. Esto también podría tratarse de un error de incompletitud. En este caso, se tendría que, primero, extender la definición de regla aislada para considerar esta situación, y, segundo, desarrollar un método que tome en cuenta las secuencias de disparo y la relación entre ellas.

2. Considerar eventos compuestos con tiempo.

   En este trabajo sólo consideramos los eventos compuestos sin tiempo `AND`, `OR` y `NOT`. Sin embargo, en algunas aplicaciones es importante conocer el tiempo en que se ocurrieron los eventos para que el sistema sepa si debe o no ejecutar alguna acción.

   A nuestro parecer, introducir eventos compuestos con tiempo en este trabajo es una acción casi directa ya que es posible determinar cuándo un intervalo de tiempo es igual a otro o cuándo uno está incluido en alguno más. De nueva cuenta, esta característica debe verse reflejada previamente en las definiciones de errores

3. Desarrollar un algoritmo de asignación de prioridades.

   Hasta este momento, al analizar una base de reglas podemos concluir si posee la propiedad de confluencia o no. Sin embargo, no podemos resolver el problema. Asignar prioridades es una forma sencilla de resolverlo, ya cuando dos reglas deban ser ejecutadas al mismo tiempo siempre se ejecutará de mayor a menor prioridad. Aunque parece fácil establecer prioridades a las reglas, si no se asignan adecuadamente, acciones importantes pueden no ser ejecutadas a tiempo o incluso no ejecutarse. Por lo tanto, es necesario desarrollar un algoritmo que tome en cuenta diferentes aspectos para hacer una asignación justa.

9.3. Comparación con trabajos relacionados

En esta sección discutiremos algunos de los resultados que obtuvimos en este trabajo de tesis en relación a los que se han reportado en la literatura.

Es importante decir que, hasta donde sabemos, sólo existe un trabajo que aborda de manera colateral la verificación de erroren en reglas ECA. La mayor parte del trabajo está relacionado con

detectar errores en reglas de producción.

En la referencia [5] los autores, Augusto y Nugent, verifican las propiedades de: 1) consistencia, se determina si las acciones de las reglas disparadas son consistentes. Además, de manera separada en tiempo de ejecución, también se verifica la consistencia de los eventos. 2) Disparo: se examina si la especificación de cada cláusula `ON` puede satisfacerse alguna vez. 3) Reunión: determina si cualquier subconjunto de reglas se puede disparar. Debido a la complejidad computacional de esta propiedad sólo se consideran combinaciones de n reglas y n es un número pequeño. 4) Cobertura: se verifica si existen pares de reglas (r_1, r_2) tales que las condiciones de disparo de r_1 son un subconjunto de las de r_2. 5) Cascada: analiza si existe una secuencia de activaciones entre reglas establecidas desde el diseño. 6) Cumplimiento de la poscondición: examina si se puede ejecutar una determinada acción después de la activación de una regla.

Nuestro trabajo puede verificar estas mismas propiedades:

1. La propiedad de consistencia se verifica en nuestro trabajo. Además, se defienen reglas en conflicto que preveen la ocurrencia de situaciones anómalas durante la ejecución de las reglas. De esta manera se pueden plantear soluciones antes de que el conflicto suceda.

2. La propiedad de disparo se verifica en nuestro trabajo a través de las secuencias de disparo. Todas las reglas que participen en una secuencia son susceptibles de dispararse ya que el evento de una regla es originado por la acción de otra. Nuestra verificación es aún más precisa cuando detectamos reglas inalcanzables, ya que estas nos indican si el proceso de ejecución termina porque la acción de la regla previa no activa la condición de la siguiente regla en la secuencia de disparo.

3. La propiedad de reunión puede implementarse en nuestro trabajo usando las secuencias de disparo.

4. La propiedad de cobertura implica nuestro análisis de redundancia.

5. La propiedad de cascada también se puede verificar analizando las secuencias de disparo en la CCPN.

6. La propiedad de cumplimiento de la postcondición se puede verificar realizando nuestro análisis para reglas inalcanzables.

Las propiedades de circularidad e incompletitud que nosotros también verificamos no están consideradas en ese trabajo.

Existe otro trabajo que no está directamente relacionada con la verificación de errores, sino con la validación de reglas ECA a través de planes [47]. En este trabajo básicamente se verifica si, dados un objetivo inicial y una meta final, existe una secuencia de actividades (plan) que lleve del objetivo inicial a la meta final. Nosotros también podemos hacer la misma validación ya que en la CCPN es posible representar los objetivos inicial y final como lugares primitivos. Si existe una secuencia de disparo que una ambos lugares entonces es posible ir del objetivo inicial al objetivo final. Además, el conjunto de transiciones tipo regla representan las actividades del plan a seguir.

Dadas las características de la CCPN, también es posible simular el comportamiento del plan. Si se colocan tokens en los lugares primitivos que representan los objetivos se puede visualizar su comportamiento. Por ejemplo, en la figura 4.9 se muestran las transiciones que se van ejecutando a partir de un estado inicial (dado por la marca inicial) de la CCPN.

En cuanto a la comparación de los métodos de detección de errores, usaremos el trabajo de la referencia [20] ya que también emplean un enfoque de verificación basado en red de Petri.

En este trabajo (y algunos otros más), el enfoque de verificación que usan está basado en el problema de alcanzabilidad de la red de Petri, es decir, dado un estado inicial determinan si es posible llegar a otro estado y así sucesivamente con cada nueva marca alcanzada. Los errores se detectan si, durante este recorrido de la red, aparecen algunas estructuras especiales. Es fácil observar que este enfoque de verificación necesita elegir una marca incicial adecuada para detectar errores o examinar la red bajo todas las posibles marcas iniciales.

Con nuestro método de verificación, este ciclo se detectaría sin la necesidad de considerar marcas iniciales.

La tabla 9.1 muestra la comparación que realizamos de nuestro trabajo con respecto a otros trabajos similares.

Tabla 9.1. Comparación con trabajos relacionados con la verificación de errores

Trabajo	**Tipo de reglas empleadas**	**Enfoque de verificación**	**Tipos de errores detectados**
Augusto J. C., y Nugent C. (2004) [5]	Reglas ECA	Lógica métrica temporal	Consistencia Disparo Reunión Cobertura Cascada Cumplimiento de la poscondición
He X., Chu W., y Yang H. (2003) [20]	Reglas de producción	Redes de Petri. Enfoque dependiente de la marca inicial.	Redundancia Inconsistencia Incompletitud Circularidad
Nuestro trabajo	Reglas ECA	Redes de Petri. Enfoque independiente de la marca inicial.	Redundancia Inconsistencia Incompletitud Circularidad * También es posible verificar los que se describen en la referencia [5]

En cuando al trabajo de confluencia y terminación los trabajos [48], [38] y [3] son representativos de los enfoques de análisis de estas propiedades.

En el trabajo reportado en la referencia [48] se aborda el problema de análisis de confluencia por medio de la técnica de reescritura de transacciones. El procedimiento que siguen es tomar una transacción de BD inicial y, usando las reglas activas, la transforman en una regla activa inducida. Por ejemplo, la transacción $t = u_1; ...; u_n$ se transforma en $t_I = u_1; \mu_1^P; ..., u_n; \mu_n^P$, donde μ_i^P denota la secuencia de actualizaciones generadas como reacción inmediata de la operación u_1 con respecto al conjunto de reglas activas P. Es decir, si u_1 corresponde con el evento de la regla r_i en la base

de reglas, μ_n^P corresponderá a la acción de r_i y así sucesivamente hasta que no haya reglas que se puedan disparar. Después de realizar esta reescritura, los autores, Montesi y Torlone, verifican ciertas propiedades para determinar si la base de reglas es confluente. Es necesario notar que los resultados de confluencia que se obtienen están directamente relacionados con la transacción inicial de entrada, por lo que para obtener resultados para toda la base de reglas es necesario examinar una gran cantidad de transacciones iniciales.

Nuestro método, a diferencia del reportado en la referencia [48], es independiente del conjunto inicial de transacciones.

En la referencia [38], las autoras analizan las propiedades de confluencia y terminación traduciendo las reglas activas en reglas deductivas y luego aplican resultados conocidos (usando los grafos de disparo y activación) para formular conclusiones. La diferencia de nuestro trabajo y el de esta referencia radica en que nosotros no realizamos trabajo adicional al transformar las reglas activas en algún otro tipo de regla. Además, al realizar su evaluación necesitan evaluar la interacción de todas las reglas. La ventaja que tiene este trabajo con respecto al nuestro es que asigna prioridades a las reglas que no son confluentes para asegurar que siempre se van a ejecutar de la misma manera. En nuestro trabajo, la asignación de prioridades está considerado como un trabajo futuro.

Finalmente, en la referencia [3], los autores, Comani y Tanca, desarrollaron un algoritmo de propagación para analizar el efecto que tendrá la ejecución de una condición sobre la posterior evaluación de una condición. Este algoritmo les es útil para determinar qué arcos se deben incluir en los grafos de disparo y activación y así determinar si la base de reglas termina. Para analizar confluencia, se basan en el concepto de conmutatividad de reglas el cual es analizado nuevamente con su algoritmo de propagación. El problema en este punto es que su algoritmo algunas veces puede producir la respuesta "puede no conmutar" cuando de hecho las reglas conmutan. También, tienen que evaluar todos los pares de reglas para examinar si conmutan o no. Nuestro método también se basa en la conmutatividad de reglas, sin embargo, para analizarlo desarrollamos el concepto de dominio de una condición y, además, consideramos cada elemento que conforma a una regla. De esta manera, nuestro algoritmo produce respuestas más precisas en cuanto a si una par de reglas conmutan. También, no necesitamos analizar todos los pares de reglas para llegar a una cierta conclusión. De hecho, el caso se estudio de la Sección 7.2 está basado en esta referencia, y aún

cuando no normalizáramos la base de reglas, tendríamos que evaluar 12 combinaciones para llegar al mismo resultado que estos autores obtienen con 15 evaluaciones.

La tabla 9.2 muestra una comparación con los trabajos relacionados.

Tabla 9.2. Comparación con trabajos relacionados con el análisis de confluencia y terminación

Trabajo	**Enfoque de análisis**	**Número de comparaciones**	**Propiedades analizadas**
Montesi D., y Torlone R. (2002) [48]	Reescritura de transacciones	Depende de las acciones generadas por la transacción inicial.	Confluencia, con respecto a una transacción específica.
Comani S., y Tanca L. (2003) [38]	Transformación de reglas ECA en reglas deductivas.	$\binom{n}{2}$ donde n es el número total de reglas	Confluencia Terminación
Baralis E., y Widom J. (2000) [3]	Grafos de disparo y activación. Algoritmo de propagación	$\binom{n}{2}$ donde n es el número total de reglas	Confluencia Terminación
Nuestro trabajo	Red de Petri. Análisis de la interacción de reglas basado en el concepto de dominio de la condición.	$< \binom{n}{2}$ donde n es el número total de reglas	Confluencia Terminación

Bibliografía

[1] Paton N. and Díaz O. Active Database System. *ACM Computer Surveys*, Vol. 31, No. 1, pp. 63-103, 1999.

[2] Paton N., *Active Rules in Database Systems*, Springer, 1999

[3] Baralis E. and Widom J. "An Algebraic Approach to Static Analysis of Active Database Rules", *ACM Transactions On Database System*, Vol. 25, No. 3, pp. 269-332, 2000.

[4] Widom J. and Ceri S., *Active Database Systems*, Morgan Kaufmann Publishers, 1996

[5] Augusto J. C., and Nugent C., "A New Architecture for Smart Homes Based on ADB and Temporal Reasoning", In *Toward a Human Friendly Assistive Environment (Proc. of 2nd International Conference On Smart homes and health Telematic, ICOST2004)*, Assistive Technology Research Series, Vol. 14, pp. 106-113, IOS Press, Singapore, September 15-17, 2004.

[6] Zhang D., Nguyen D., "PREPARE: A Tool for Knowledge Base Verification", *IEEE Trans. on Knowl. and Data Eng.,* vol. 6, no. 6, pp. 983-989, 1994.

[7] Guisheng Y., Qun L., Jianpei Z., Jie L., Daxin L., "Petri Based Analysis Method for Active Database Rules", *IEEE International Conference on Systems, Man, and Cybernetics*, vol. 2, 14-17 Oct, pp.858 - 863, 1996

[8] Chandra R., Segev A., "Active Databases for Financial Applications", in *Proceedings of the Fourth International Workshop on Research Issues in Data Engineering*, 14 - 15 Feb., pp. 46-52, 1994.

[9] Elmasri R., Navathe S. B., *Fundamentos de Sistemas de Bases de Datos*, Addison Wesley Iberoamericana, 2002.

[10] González A. J., Dankel D. D., *The engineering of knowledge-based systems. Theory and practice*, Prentice-Hall, 1993

[11] Wu C. and Lee S., "A Token-Flow Paradigm for Verification of Rule-Based Expert Systems", *IEEE Trans. on Systems, Man and Cybernetics- Part B: Cybernetics*, Vol. 30, No. 4, pp.616-624, 2000.

[12] Van Melle W., Shortliffe H., and Buchanan G., "EMYCIN: A Knowledge Engineer's Tool for Constructing Rule-Based Systems", *Rule-Based Expert Systems*, Addison-Wesley, pp. 301-313, 1984.

[13] Suwa M., Scott A.C., and Shortliffe H., "An Approach to Verify Completeness and Consistency in a Rule Based Expert System", *AI Magazine*, pp 16-21, 1982.

[14] Wu C. H., and Lee S. J., "Knowledge Verification with an Enhanced High-Level Petri-Net Model", *IEEE Expert: Intelligent Systems and Their Applications*, vol. 12 , No. 5, pp: 73 - 80, 1997

[15] Wu Q., Zhou C., Wu J., and Wang C., "Study on Knowledge Base Verification Based on Petri Nets", *International Conference on Control and Automation (ICCA2005)* Budapest, Hungry, June, pp. 27-29, 2005.

[16] Yang S. J. H., Lee A. S., Shu W. C., and Yang H., "Rule Base Verification Using Petri Nets", in *Proceedings of the 22nd International Computer Software and Applications Conference*, pp. 476 - 485, 1998

[17] Nazareth D. L., "Investigating the Applicability of Petri Nets for Rule-Based System Verification", *IEEE Transactions on Knowledge and Data Engineering,* Vol. 5, No. 3, pp. 402 - 415, 1993.

[18] Zaidi A. K., and Levis A. H., "Validation and Verification of Decision Making Rules", *Automatica*, Vol. 33, No.2, pp. 155-169, 1997.

[19] Yang S., "Fuzzy Rule Base Systems Verification Using High-Level Petri Nets", *IEEE Trans. on Knowledge and Data Engineering*, Vol. 15, No. 2, pp. 457-473, 2003.

[20] He X., Chu W., and Yang H., "A New Approach to Verify Rule-Based Systems Using Petri Nets", *Inf. and Soft. Tech.,* Vol. 45, No. 10, pp. 663-670, 2003.

[21] Ramaswamy M., Sarkar S., and Chen Y.S., "Using Directed Hypergraphs to Verify Rule-Based Expert Systems", *IEEE Trans. on Knowl. and Data Eng.*, Vol. 9, No. 2, pp. 221-237, 1997.

[22] Valiente G., "Verification of Knowledge Base Redundancy and Subsumption using Graph Transformations", *Intl. Journal of Expert Systems*, Vol. 6, No. 3, pp. 341-355, 1993.

[23] Loiseau S. , and Rousset M., "Formal Verification of Knowledge Bases Focused on Consistency: Two Experiments Based on ATMS Techniques", *Intl. Journal of Expert Systems*, Vol. 6, No. 3, pp. 341-355, 1993.

[24] Fujiwara Y., and Honiden S. , "On Logical Foundations of the ATMS", Workshop on Truth Maintenance Systems, *LNCS 515*, pp. 125 -135, 1990.

[25] Prece A., "Validation of Knowledge-Based Systems: The State-of-the-art in North America", *Communication and Cognition - Artificial Intelligence*, Vol. 11, No. 4, pp.381-413, 1994.

[26] A. De Antonio, J. Ramírez, and J. Clemente, "CRIB: A Method for Integrity Constraint Checking on Knowledge Bases", *Revista Iberoamericana de Computación y Sistemas*, Vol. 8, No. 1, 2004.

[27] Coenen F.P., "An Advance Binary Encoded Matrix Representation for Rule Base Verification", *Knowledge-Based System*, Vol. 8, No. 4, pp. 201 - 210, 1998.

[28] Santos J., Ramos C., Vale Z., and Marques A., "VERITAS: An Application for Knowledge Verification", in *IEEE International Conference on Tools with Artificial Intelligence*, Illinois, USA, pp. 441 - 444, 1999.

[29] Spreeuwenberg S., and Gerrits R., "Requirements for Successful Verification in Practice", in *The Florida Artificial Intelligence Research Society*, Florida, USA, pp. 221 - 225, 2002.

[30] Aiken A., Widom J., and Hellrstein J., "Behavior of Database Production Rules: Termination, Confluence and Observable Determinism", In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pp. 59–68, San Diego, California, June 1992

[31] Baralis E., Ceri S., and Paraboschi S., "Run-Time Detection of Non-Terminating Active Rule Systems", in *Proc. of the 4th Intl. Conf. on Deductive and Object-Oriented Databases, DOOD 95,* LNCS, Vol. 1013, pp. 38 - 54, Singapore, 1995.

[32] Vaduva A., Gatziu S., and Dittrich K.R., "Investigating Termination in Active Database Systems with Expressive Rule Languages", in A. Geppert and M. Berndtsson, editors, in *Proc. 3rd. Int. Wshp. on Rules In Database Systems*, Skovde, pp. 149–164, Springer-Verlag, 1997.

[33] Weik T., and Heuer A., "An Algorithm for the Analysis of Termination of Large Trigger Sets in an OODBMS", in *Proceedings of the International Workshop on Active and Real-Time Database Systems*, June 1995.

[34] Harb H., Kelash H., and Shedata A., "Termination Analysis in Active Databases by Using Evolution Graphs", in *3th International Conference on Information & Communications Technology*, pp. 781 - 791, 2005.

[35] Montesi D., Bertino E., Bagnato M., and Dearnley P., "Rules Termination Analysis Investigating the Interaction between Transactions and Triggers", in *Proc. of the International Database Engineering and Applications Symposium (IDEAS'02)*, pp. 285 - 294, 2002.

[36] Montesi D., Bagnato M., Dallera C., "Termination Analysis in Active Databases", in *Proceedings of the International Symposium on Database Engineering & Applications*, pp. 288 - 297, 1999

[37] Baralis E., and Widom J., "An Algebraic Approach to Static Analysis of Active Database Rules", *ACM Transactions on Database Systems*, Vol. 25 , No. 3, pp. 269 - 332, 2000.

[38] Comani S., and Tanca L., "Termination and Confluence by Rule Prioritization", *IEEE Trans. on Knowl. and Data Eng.*, Vol. 15, No.2, pp. 257-270, 2003.

[39] Leahu I., and Tiplea F., "The Confluence Property for Petri Nets and its Applications", in *Proc. of the Eighth Intl. Symposium on Symbolic and Numeric Algorithms for Scientific Computing*, pp. 430 - 436, 2006

[40] Medina-Marín J., *Desarrollo de Reglas ECA en Bases de Datos Activas, un Enfoque de Red de Petri,* Sección de Computación, Departamento de Ingeniería Eléctrica, CINVESTAV-IPN, México. Octubre 2005.

[41] Zimmer D., Meckenstock A., and Unland R., "Using Petri Nets for Rule Termination Analysis", in *Proceedings of the workshop on on Databases: active and real-time*, Rockville, Maryland, United States, pp. 29 - 32, 1996.

[42] Díaz O., Piattini M. and Calero C., "Measuring Triggering-Interaction Complexity on Active Databases", *Information Systems*, vol. 26, No. 1, pp 15-34, 2001.

[43] Chavarría-Báez L., *Medición de la complejidad de la interacción de las reglas ECA en BDA vía CCPN,* Sección de Computación, Departamento de Ingeniería Eléctrica, CINVESTAV-IPN, México. Octubre de 2004

[44] Murata T., "Petri Nets: Properties, Analisys and Applications", *Proceedings of the IEEE*, Vol. 77, No. 4, pp. 541-580, 1989.

[45] Baralis E., Ceri S., and Paraboschi S., "Improved Rule Analysis by Means of Triggering and Activation Graphs", in *Proc. of 2nd. Intl. Workshop on Rules in Database Systems, RIDS'95*, LNCS Vol. 985, pp. 163 - 181, 1995.

[46] X. Li, J. Medina-Marín, and Chapa S., "Applying Petri Nets on Active Database Systems", *IEEE Trans. on System, Man, and Cybernetics, Part C: Applications and Reviews,* Vol. 37, No. 4, pp. 482 - 493, 2007.

[47] Fraternali P., Teniente E., Urpí T., "Validating Active Rules by Planning", In *Proceedings of the 3rd International Workshop on Rules in Database Systems*, Springer LNCS Vol. 1312, pp. 181-196, 1997.

[48] Montesi D., and Torlone R., "Analysis and Optimization of Active Databases", *Data & Knowledge Engineering*, Vol. 40, pp. 241 - 271, 2002.

[49] Ceri S., and Gottlob G., "Translating SQL into Relational Algebra: Optimization, Semantics, and Equivalence of SQL Queries", *IEEE Trans. on Software Engineering*, Vol. 11, No. 4, pp. 324 - 345, 1994.

[50] Jin Y., Urban S. D., and Dietrich S. W., "A Concurrent Rule Scheduling Algorithm for Active Rules", *Data & Knowledge Engineering*, Vol. 60, pp. 530 - 546, 2007.

[51] DeMara R. F., Tseng Y., and Ejnioui A., "Tiered Algorithm for Distributed Process Quiescence and Termination Detection", *IEEE Trans. on Parallel and Distributed Systems*, Vol. 18, No. 11, pp 1529 - 1538, November 2007.

[52] Hanson E. N., and Widom J., "Rule Processing in Active Database Systems", *Intl. Journal of Expert Systems*, Vol. 6, No. 1, pp. 83-119, 1993.

[53] O'Neal M.B., and Edwards W.R., "Complexity Measures for Rule-Based Programs", *IEEE Trans. on Knowledge and Data Engineering*, Vol. 6, No. 5, pp. 669-680, 1994.

[54] Baralis E., Ceri S., and Paraboschi S., "Compile-Time and Runtime Analysis of Active Behaviors", *IEEE Trans. on Knowledge and Data Engineering*, Vol. 10, No. 3, pp. 353-370, 1998.

Publicaciones

Las siguientes son las publicaciones que hemos desarrollado durante el trabajo de tesis:

1. Lorena Chavarría-Báez and Xiaoou Li , Knowledge Verification of Active Rule-Based Systems, *ICIC2006: Intelligent Control and Automation*, Springer-Verlag, Lecture Notes in Control and Information Sciences (LNCIS), vol. 344, 676-687,2006

2. Lorena Chavarría Baez, Xiaoou Li, Static Verification of Active Rule-Based System, the *10th IASTED International Conference on SoftwareEngineering and Applications, (SEA 2006)*, Dallas, Texas, USA, 514-084, November 13 - 15, 2006

3. Lorena Chavarría Baez, Xiaoou Li, Structural Error Verification in Active Rule-Based Systems using Petri Nets, In Alexander Gelbukh and Carlos Alberto Reyes-García (editors), *Fifth Mexican International Conference on Artificial Intelligence (MICAI 2006)*, IEEE Computer Science, pp. 12-21, November 13-17, Apizaco, Mexico, 2006.

4. Lorena Chavarría-Báez, Xiaoou Li, Verification of ECA Rule Base via Conditional Colored Petri Nets, *IEEE International Conference on Systems, Man, and Cybernetics*, Montreal, Quebec, Canada, October 7-10, 2007

5. Lorena Chavarría-Báez and Xiaoou Li, Analyzing Termination and Confluence in Active Rule Base via a Petri Net Approach, the *20th International Conference on Software Engineering and Knowledge Engineering (SEKE'08)*, Redwood City, California, USA, July 1-3, 2008

6. Lorena Chavarría-Báez and Xiaoou Li, Active Database System Realized by a Petri Nets Approach, the *6th International Workshop on Modelling, Simulation, Verification and Validation of Enterprise Information Systems (MSVVEIS'08)*, 12-13 June, 2008, Barcelona, Spain

7. Lorena Chavarría-Báez, Xiaoou Li, Integrating Active Rules into Database System via a Petri Net Approach, *International Workshop on Petri Nets and Distributed Systems (PNDS'08)*, Xi'an, China, June 23-24, 2008