CM

CINVESTAV

Centro de Investigación y de Estudios Avanzados del I.P.N.
Unidad Guadalajara

Síntesis de Controladores basados en Autómatas para Sistemas Dinámicos de Eventos Discretos a Gran Escala

Tesis que presenta:

Eduardo Isaac Ramírez Jiménez

para obtener el grado de:
Maestro en Ciencias

en la especialidad de:
Ingeniería eléctrica

Director de Tesis
Dr. Arturo del Sagrado Corazón Sánchez Carmona

Guadalajara, Jalisco, Diciembre del 2004.

Síntesis de Controladores basados en Autómatas para Sistemas Dinámicos de Eventos Discretos a Gran Escala

Tesis de Maestria en Ciencias
Ingeniería eléctrica

Por:

Eduardo Isaac Ramírez Jiménez

Ingeniero en Comunicaciones y Electrónica
Universidad de Guadalajara 1998-2002

Becario de CONACYT, expediente no. 171592

Director de Tesis

Dr. Arturo del Sagrado Corazón Sánchez Carmona

CINVESTAV del IPN Unidad Guadalajara, Diciembre del 2004.

Centro de Investigación y Estudios Avanzados del
Instituto Politécnico Nacional

Síntesis de Controladores basados en Autómatas para Sistemas Dinámicos de Eventos Discretos a Gran Escala

Tesis que presenta

Eduardo Isaac Ramírez Jiménez

Para obtener el grado de

Maestro en Ciencias

En la especialidad de

Ingeniería Eléctrica

Guadalajara, Jalisco, Diciembre 2004

Síntesis de Controladores basados en Autómatas para Sistemas Dinámicos de Eventos Discretos a Gran Escala

Tesis de Maestría en Ciencias

Ingeniería Eléctrica

por:

Eduardo Isaac Ramírez Jiménez

Ingeniero en Comunicaciones y Electrónica

Universidad de Guadalajara

Asesor de Tesis

Dr. Arturo del Sagrado Corazón Sánchez Carmona

Becado por CONACYT, Expediente No. **171592**

Síntesis de Controladores basados en Autómatas para Sistemas Dinámicos de Eventos Discretos a Gran Escala

Eduardo Isaac Ramírez Jiménez

Departamento de Ingeniería Eléctrica y Ciencias de la Computación

CINVESTAV del IPN

Guadalajara, Jalisco

Resumen

Este trabajo de tesis trata sobre el diseño de controladores lógicos para sistemas de eventos discretos a gran escala, con la finalidad de resolver problemas de interés industrial. Para el cálculo de los controladores lógicos se utilizó una técnica basada en la representación de las máquinas de estado finito (MEFs) mediante el uso de la lógica de predicados y su representación computacional con diagramas de decisión binarios (BDDs).

Dicha técnica de cálculo fue implementada anteriormente en una herramienta computacional llamada SSPC (Software for Synthesis of Procedural Controllers). En este trabajo se integran algoritmos incrementales basados en la teoría de control supervisor (TCS) mediante los cuales se logran calcular supervisores para espacios de estados potenciales mayores a los que se habían podido trabajar en las versiones anteriores del SSPC.

Este algoritmo funciona incrementalmente partiendo de las MEFs del sistema y de las especificaciones por separado, sin la necesidad de realizar previamente el producto síncrono entre la planta y la especificación. Además de este algoritmo, se implementaron técnicas de optimización en el ordenamiento de variables de los BDDs, lo cual permitió sintetizar controladores para un espacio de estados potencial del orden de 10^{27}

También, en cuanto a la teoría del control de procedimientos (TCP), se logró mediante la utilización de otro algoritmo incremental basado en el anterior, y las técnicas de optimización del ordenamiento de variables de los BDDs antes mencionadas, mejorar la capacidad de cálculo de la superestructura para el controlador de procedimientos, la cual puede trabajar con un espacio de estados potencial del orden de 10^{12}

En cuanto al cálculo del controlador de procedimientos, se consiguió obtener, mediante la aplicación de la optimización del ordenamiento de variables de los BDDs, la síntesis de controladores para espacios de estados de hasta 10^6.

Para mostrar las mejoras en el SSPC se realizaron pruebas del desempeño de la herramienta, aplicándola a varios ejemplos y comparándola con otras herramientas existentes y con la versión anterior del SSPC. Estos resultados permiten observar un avance considerable en el potencial de cálculo de controladores de esta herramienta, en comparación con las herramientas desarrolladas anteriormente.

Finalmente, se realizaron otras mejoras para la facilitación del uso de la herramienta SSPC por usuarios, tales como: la generación de autolazos en las MEFs de las especificaciones de manera interna por el programa y la etiquetación de los todos los estados marcados en los archivos de salida.

Agradecimientos

Expreso mi agradecimiento a todas las personas que me brindaron su apoyo y ayuda durante la realización de este trabajo de tesis.

En primer lugar quiero agradecer a mis padres por haberme apoyado durante toda mi vida para que nunca me faltara nada.

A mi asesor de tesis el Dr. Arturo del Sagrado Corazón Sánchez Carmona, por haberme guiado y compartido sus conocimientos para la realización de este trabajo, así como por su amistad, interés y colaboración.

A mi hermana, mis abuelos, tíos, primos, amigos y a mis compañeros de la licenciatura por su amistad y apoyo que siempre me han brindado y me ha ayudado a cumplir todas mis metas.

A todos los profesores del departamento de Control Automático que ayudaron en mi formación, así como también a mis compañeros y amigos de la generación de control 2002 y del laboratorio de eventos discretos por su amistad y apoyo intelectual durante estos dos años.

Al CONACYT, al CINVESTAV y al IMP por haberme otorgado el apoyo económico para la realización de mis estudios.

Indice general

Indice de Tablas

Indice de figuras

Capítulo 1

Introducción

Este trabajo de tesis se desarrolla en el área de los sistemas de eventos discretos (SEDs). En este capítulo se presenta una breve introducción sobre la síntesis de controladores basados en autómatas para este tipo de sistemas. En particular se plantean las diferentes corrientes mediante las cuales se ha abordado el problema de la síntesis de controladores, como los algoritmos de cálculo explícito y los de cálculo simbólico. También se discuten los logros que se tienen hasta el momento en cuanto a la capacidad de cálculo alcanzada mediante diversas implementaciones y herramientas computacionales existentes. Especialmente se presenta la explicación de los trabajos previamente realizados por Reza [20] y Douriet [7] en los cuales se elaboró una herramienta llamada SSPC para el cálculo de controladores basados en autómatas de forma simbólica. Dichos trabajos se toman como punto de partida para la realización de este trabajo de tesis. Finalmente, se presentan los objetivos de este trabajo y el contenido de la tesis.

1.1. Síntesis de controladores basados en autómatas para sistemas de eventos discretos

La teoría de control supervisor (TCS) para sistemas de eventos discretos (SEDs) fue desarrollada inicialmente por Wonham y Ramadge [29] para el estudio de sistemas cuyo comportamiento es descrito por la ocurrencia de eventos; ejemplos de este tipo

de sistemas incluyen a sistemas de manufactura, redes de comunicación, sistemas de manejo de bases de datos, etc. El área de estudio de este tipo de sistemas es muy amplia y abarca distintas disciplinas relacionadas con las ciencias de la computación, ingeniería de sistemas, ingeniería de control, ingeniería de software, etc. [5, 28].

Este trabajo se enfoca particularmente a los sistemas de eventos discretos (SEDs) modelados por medio de máquinas de estados finitos (MEFs). Se estudia el cálculo de dos tipos de dispositivos de control. El primero, conocido como control supervisor [5, 29] el cual tiene la función de inhabilitar eventos controlables en cada estado para restringir el comportamiento del sistema a un comportamiento deseado. El segundo, conocido como control de procedimientos [25] el cual es un mecanismo de control que establece la secuencia de ejecución de comandos de control de forma determinística.

Un controlador de procedimientos define un mecanismo de control que establece una secuencia de eventos a ejecutar. Estos controladores han sido aplicados a la industria de procesos, tales como la farmacéutica, petroquímica, alimentos, etc. [22].

1.2. Trabajo relacionado

1.2.1. Teoría de control supervisor

La teoría de control supervisor aplicada a los sistemas de eventos discretos que introdujeron Wonham y Ramadge [29] está basada en la representación explícita de los comportamientos de la planta y la especificación mediante el uso de MEFs y sus respectivos lenguajes. También Wonham y Ramadge [29] encontraron un operador de punto fijo para el cálculo del supremo sublenguaje controlable que permite encontrar un controlador supervisor no bloqueante mínimamente restrictivo.

Esta teoría ha servido como base para el desarrollo de nuevas teorías e investigaciones y es reconocida como una de las más sólidas en el área del control de SEDs. Esto se debe a que se garantiza la existencia de un supremo lenguaje controlable, el cual satisface mínimamente el lenguaje de un conjunto de especificaciones.

También se han desarrollado trabajos basados en la teoría de control supervisor mediante el uso de redes de Petri (RP) [12, 13]. Aquí los algoritmos para el cálculo

de un control supervisor mínimamente restrictivo han sido bien desarrollados cuando, tanto la planta como el comportamiento deseado, están dados por lenguajes regulares. En estos trabajos se ha encontrado que el problema de determinar si un lenguaje determinístico de una red de Petri es controlable con respecto a otro lenguaje similar se reduce a un problema de alcanzabilidad de redes de Petri. Por otra parte, el problema de sintetizar un controlador supervisor mínimamente restrictivo capaz de generar el supremo sublenguaje controlable se reduce a un problema de marcados prohibidos de una RP.

1.2.2. Teoría de control de procedimientos

Un controlador de tipo supervisor tiene la función de inhabilitar eventos controlables en cada estado para restringir el comportamiento del sistema a un comportamiento de especificación dado. Sin embargo, existen aplicaciones de controladores para SEDs con fines diferentes a los que TCS puede ofrecer. Tal es el caso en que es necesario anticiparse a la ejecución de una transición incontrolable para que ésta no ocurra. Este comportamiento puede no ser controlable desde el punto de vista de TCS.

Con base en TCS surge la teoría de control de procedimientos (TCP). Un controlador de procedimientos es un dispositivo que controla la ejecución de transiciones controlables en un proceso dado en función de ciertas entradas.

Un controlador de procedimientos define un mecanismo de control que claramente establece la secuencia de ejecución de los comandos de control de forma determinística. El controlador de procedimientos no requiere de un mecanismo externo de decisión para manejar el proceso en forma segura [25]. Esto hace que la TCP tenga un mejor desempeño con respecto a TCS en sistemas donde se necesita prevenir la ocurrencia de eventos incontrolables.

El controlador de procedimientos que acepte todas aquellas secuencias que satisfacen la especificación dada y que además es aseada, se genera a partir de una superestructura maximal [25]. Dicha superestructura de controlador de procedimientos es una MEF que conjunta todas las posibles secuencias de ejecuciones de transiciones

candidatas a ser controlador de procedimientos y es caracterizada mediante la aplicación de la definición de controlabilidad en TCP.

1.2.3. Herramientas para la Síntesis de Controladores utilizando Técnicas de Calculo Explícito

Como ya se dijo, Ramadge y Wonham introdujeron la TCS basada en la representación explícita de los comportamientos de la planta y la especificación mediante el uso de MEFs y sus respectivos lenguajes.

El mismo Wonham desarrolló un software llamado CTCT (C Toy Control Theory) el cual es capaz de trabajar con sistemas de hasta 10^4 posibles estados del modelo de la planta. CTCT implementa en lenguaje C los algoritmos propuestos por Ramadge y Wonham usando cálculos explícitos con una sintaxis simple para la entrada de información, además de trabajar en diversos sistemas operativos.

Existen algunas otras implementaciones académicas que utilizan cálculos explícitos para los controladores basados en los algoritmos de Ramadge y Wonham. Una de ellas es Supremica [2], herramienta gráfica que fue desarrollada en la Universidad de la Tecnología en Suecia. Esta herramienta trabaja con archivos en lenguaje XML. Mediante Supremica pueden realizarse operaciones entre MEFs, tales como: sincronización, verificación y síntesis. Además, cada MEF, incluyendo aquellas obtenidas como resultado de las operaciones, puede ser analizada mediante un visor gráfico que permite explorar todos los estados. También, se puede simular la ejecución de transiciones para ir determinando el comportamiento de una MEF. Esta herramienta de cálculo explícito es capaz de realizar la síntesis de supervisores para espacios de estados de hasta 10^3 Dicha capacidad de cálculo fue superada por las herramientas de cálculo simbólico como se muestra en las pruebas de desempeño realizadas por Douriet [7].

UKDES [6] es una herramienta gráfica hecha en Java para el diseño, análisis y control de SEDs implementada en la Universidad de Kentucky (UK). UKDES se comprende de un analizador y un diseñador de MEFs, los cuales son usados para la parte de análisis y de diseño respectivamente. Los modelos de las MEFs son dibujados

por el diseñador en un entorno visual y pueden usarse para la síntesis de supervisores utilizando las funciones del analizador basadas en la teoría de control supervisor. Los algoritmos explícitos de la TCS están codificados con el lenguaje de programación C y son utilizados por la herramienta para los cálculos correspondientes. Al igual que Supremica, esta herramienta es de cálculo explícito, y también es capaz de realizar la síntesis de supervisores para espacios de estados de hasta 10^3 De igual forma que para Supremica, su capacidad de cálculo fue superada por las herramientas de cálculo simbólico como se muestra en las pruebas de desempeño realizadas por Douriet [7].

Venegas [27] desarrolló un algoritmo de cálculo incremental que se utiliza para sintetizar un controlador a partir de un conjunto de especificaciones de seguridad consistentes y controlables. Con este algoritmo se logró calcular controladores de procedimientos para sistemas de hasta 10^7 estados potenciales. El algoritmo se caracteriza por la forma incremental en que se va construyendo el controlador. Esto consiste en la construcción en profundidad de la MEF estado por estado. Un estado forma parte del controlador si es alcanzable y controlable. Además, si es un estado final, éste debe ser marcado. El proceso incremental estado por estado se lleva a cabo por evolución natural del proceso, es decir, cada uno de ellos se genera en función del estado antecesor y una transición habilitada por el proceso. Antes de aplicar el crecimiento por evolución natural, primeramente se verifica si existe alguna especificación para crecer. Si es así, entonces el crecimiento es por especificación. En caso contrario, el crecimiento se dará por evolución natural. El problema de este algoritmo es que no garantiza la obtención de un controlador no bloqueante.

Uno de los principales problemas que existen al momento de sintetizar un mecanismo de control utilizando estas técnicas es la explosión de estados. Es decir, que si se utilizan MEFs para modelar el comportamiento de la planta con una enumeración explícita del espacio de estados, al aumentar el número de componentes de la planta, el espacio de estados puede crecer de manera exponencial, haciendo inmanejable computacionalmente las representaciones de los sistemas y, por lo tanto, difícil de resolver problemas grandes como los que se encuentran en la práctica industrial.

1.2.4. Herramientas para la Síntesis de Controladores utilizando Técnicas de Cálculo Simbólico

Con el fin de disminuir la explosión de estados que surge con las herramientas de cálculo explícito al momento de llevar a cabo la síntesis de los controladores y de aumentar la capacidad de cálculo computacional, surgen varias implementaciones para la síntesis de controladores mediante el uso de técnicas de cálculo simbólico que permitan caracterizar la información estructural del sistema y, en cierta medida, ayuden a evitar la ya mencionada explosión de estados.

Li y Wonham [14, 15] propusieron el uso de vectores para SEDs. Con este método, los estados del sistema son representados mediante un vector de componentes enteros, y las transiciones entre los estados se representan mediante una adición de vectores. De esta manera, el problema de sintetizar un controlador puede ser resuelto bajo ciertas restricciones utilizando técnicas de programación lineal entera. Este método se aplicó al modelo de una planta de manufactura con vehículos autoguiados (AGV's), cuyo espacio de estados potenciales es de 8.2×10^7 Esta es una alternativa para manejar sistemas de mayores dimensiones. El problema con el uso de redes de Petri y de vectores para SED es que no se puede garantizar que el controlador obtenido sea no bloqueante. Es decir, existe la posibilidad que exista algún estado a partir del cual el sistema ya no pueda moverse hacia un estado marcado.

Kumar [11] *et al.* proponen utilizar el espacio de estados como concepto fundamental en vez del alfabeto y el conjunto de eventos. Se propone un método mediante predicados y transformaciones de predicados. En este trabajo se define la noción de controlabilidad y control supervisor por medio de predicados. Se muestra que el uso de predicados es una alternativa para manejar sistemas con espacios de estados de gran tamaño pero no se reportan resultados acerca del alcance o la complejidad del método.

Una de las principales técnicas simbólicas utilizadas para la representación de las MEFs es el uso de diagramas de decisión binarios (BDDs) [3]. Su principal ventaja es que su representación suele ser mucho más pequeña que la representación explícita de las funciones booleanas.

Hoffman y Wong-Toi [8] desarrollaron un método de síntesis simbólico usando BDDs para la representación de las MEFs. Este método se aplicó a un modelo de manufactura de semiconductores con un espacio de 2.3×10^6 estados potenciales. El proceso consiste en el crecimiento de óxido en obleas de silicio. La planta y la especificación se modelan como predicados y se proponen un operador de punto fijo sobre predicados para obtener el supremo sublenguaje controlable de una especificación para una planta dada. Los predicados de la planta y la especificación son representadas por funciones booleanas las cuales se codifican como BDDs logrando así una representación más compacta y eficiente. En este trabajo no se describe en forma precisa como es que se convierten las MEFs a predicados. El controlador que se obtiene es un predicado, el cual a su vez debe ser convertido a máquina de estados finitos.

Posteriormente, Balemi [4] *et al.* estudiaron la síntesis del controlador supervisor para este mismo sistema de manufactura. El cálculo del supervisor y de un controlador que realiza una tarea específica fueron implementados en un programa escrito en C. Este programa lee la información del comportamiento de un autómata y describe el controlador sintetizado para una tarea específica. La implementación de los algoritmos de síntesis también se basa en BDDs. Se reportó la síntesis de controladores con un espacio de estados potencial de 4.2×10^8

Tronci [26] trabajó con un sistema de manufactura de semiconductores, el cual se describe en [1], usando técnicas simbólicas basadas en BDDs. El escenario de trabajo que utilizó fue el de encontrar un control óptimo, el cual consiste en encontrar el camino más corto para llegar a un estado final de la planta, con una especificación dada y habilitando para cada estado sólo una transición. Se llegaron a calcular controladores para plantas de 2.94×10^9 estados.

En un trabajo más reciente, Zhang y Wonham [30] proponen un algoritmo simbólico basado en la TCS para obtener el conjunto de estados alcanzables y coalcanzables de la MEF que genera el control supervisor. En este trabajo se aprovecha la composición modular de la planta y las especificaciones para calcular los controladores de forma incremental. Se encuentra un ordenamiento óptimo de componentes elementales de la planta y la especificación para reducir el uso de memoria de la computadora y el tiempo de procesamiento. Se caracteriza la información estructural mediante el

uso de predicados. Se introducen predicados para la coalcanzabilidad, así como para la controlabilidad. Se propone un predicado P_{sup}, el cual abarca a todos los estados y posteriormente va eliminando aquellos estados que no cumplan con la controlabilidad o coalcanzabilidad hasta que se obtiene un predicado que contiene sólo los estados controlables y coalcanzables del sistema. Según Zhang [31], con este conjunto de estados es posible sintetizar un controlador supervisor, pero esta parte del método aún no ha sido desarrollada. La eficiencia de este método se debe en gran medida al uso de ciertas estructuras de datos llamadas diagramas de decisión enteros (IDDs) y al uso de técnicas de reordenamiento de variables aplicadas sobre los IDDs, logrando así una representación compacta y eficiente de los predicados. Las técnicas de reordenamientos son de gran importancia, ya que al igual que con los BDDs el orden elegido de variables determina en gran medida el tamaño de los IDDs. En este trabajo se presentan varios ejemplos, presentando resultados de síntesis de controladores para sistemas de hasta 10^{23} estados potenciales.

Este trabajo realizado por Zhang se tomó como una de las principales bases para esta tesis. Las características principales las cuales se consideraron son: la composición modular que permiten realizar el cálculo de los controladores en partes, y de esta forma utilizar BDDs pequeños que ahorren espacio en memoria y tiempo de cálculo, ya que las operaciones realizadas entre estructuras de datos pequeñas son más fáciles y rápidas de llevar a cabo que las de gran tamaño; el uso de los predicados que define para la controlabilidad y coalcanzabilidad son fáciles de interpretar y de implementar; además, el uso de solamente estados en el algoritmo para el cálculo del control supervisor permite no requerir de todo la relación de transición en ciertos pasos.

1.2.5. Síntesis de Controladores mediante el Software para la Síntesis de Controladores de Procedimientos (SSPC)

Reza [20] desarrolló una herramienta computacional para el cálculo de controladores la cual fue programada en lenguaje C. Esta herramienta utiliza predicados para representar la información estructural del sistema y definir las operaciones que actúan sobre el conjunto de estados. Además, para lograr una representación eficiente

y compacta de los predicados, se combinó el uso de predicados con el uso de BDDs al igual que algunas de las herramientas simbólicas mencionadas en la sección anterior. Esta combinación es una alternativa de cálculo para la síntesis de controladores para SEDs con la cual se puede caracterizar la información estructural del sistema. Así pues, todos los predicados y sus operaciones lógicas fueron implementados mediante BDDs.

Este software también implementa el uso de predicados para agregar tanto información causal al modelo como para especificar situaciones de estados prohibidos. También tiene un operador de punto fijo para calcular el supremo sublenguaje controlable y su equivalente para obtener la superestructura maximal; así mismo, este operador de punto fijo genera la superestructura maximal del controlador de procedimientos. Además, se implementaron algoritmos para obtener la alcanzabilidad y coalcanzabilidad de los estados en las MEFs.

SSPC versión 1

Reza desarrollo la primera versión de la herramienta llamada "Software para la Síntesis de Controladores de Procedimientos"(SSPC) [20], donde presentó las bases para la representación simbólica de las MEFs por medio de funciones booleanas. Además, presentó los métodos de cálculo simbólico para el análisis de alcanzabilidad, coalcanzabilidad, producto síncrono y producto asíncrono.

Esta herramienta trabaja de manera simbólica con todo el espacio de estados tanto de la planta como de la especificación, y en esta versión se logró trabajar con espacios del orden de 10^5 estados y sintetizar controladores para plantas de hasta 3.2×10^4 estados.

Estos resultados no superaron por mucho los resultados obtenidos por medio de herramienta basadas en algoritmos de cálculo explícito. Esto se debió en gran parte a una mala implementación de los BDDs en el código de la herramienta. Sin embargo, estos algoritmos sirvieron de base para versiones posteriores de la herramienta.

SSPC versión 2

La segunda versión de la herramienta SSPC, fue desarrollada por Douriet [7]. En esta versión se utilizó un buen ordenamiento de variables para los BDDs, el cual permitía trabajar con BDD más compactos. Dicho orden consistía en colocar las variables que se usaban para representar a las transiciones al inicio, luego las que representaban a los estados actuales del sistema, a los estados siguientes del sistema, y después a las que representaban a los estados actuales de las especificaciones, y finalmente, a las que se usaban para los estados siguientes de las especificaciones. Además, se estandarizó la escritura tanto de los archivos de entrada como de los archivos de salida.

Esta versión 2 del SSPC logró superar todas las herramientas para el cálculo de la síntesis de controladores existentes en ese momento [7, 24], tales como Supremica y UKDES. Mediante la implementación de las mejoras hechas a esta herramienta se obtuvo la generación de archivos de salida más fáciles de interpretar para los usuarios, así como gran capacidad de cálculo. Se consiguieron realizar productos asíncronos para 10^{43} estados potenciales, productos síncronos exactos para 10^{8} estados potenciales, controladores supervisores y superestructuras maximales para espacios de estados potenciales de 10^{6}, y controladores de procedimientos para espacios de estados potenciales de 10^{5}.

1.3. Objetivos y contenido de la tesis

1.3.1. Motivación y objetivos del trabajo de tesis

Como ya se mencionó, uno de los problemas principales que existe en la síntesis de controladores para SEDs es el ocasionado por la explosión de estados, lo cual dificulta la síntesis de controladores para sistemas a gran escala, que se presentan principalmente en casos industriales como los procesos existentes en la industria petrolera [23].

Las técnicas de diseño que actualmente se han explorado e implementado para estos SEDs a gran escala, incluyen la descomposición modular y jerárquica de los sistemas de control de acuerdo a los criterios de operación y la arquitectura del equipo, así como la síntesis de controladores utilizando cálculos de tipo explícito incremental y de tipo simbólico. Esto ha permitido resolver problemas de complejidad industrial para plantas completas en donde un controlador modular se sintetiza partiendo de espacios de estados de hasta 10^{12}, utilizando una PC estándar como herramienta de cálculo. La aplicación de estas técnicas antes mencionadas fue llevada a cabo por Douriet [7] en un ejemplo relacionado con un un sistema de administración y control de mezclado de petróleo crudo.

Sin embargo, en muchas ocasiones no es posible realizar una descomposición del sistema en cuestión que dé como resultado subsistemas de estos órdenes de magnitud. Se pueden dar casos industriales con espacios de estados demasiado grandes para los cuales, utilizando técnicas de descomposición, se llegan a obtener subsistemas con espacios de tamaños menores que no son suficientes para efectuar el cálculo de la síntesis del controlador.

En este trabajo de tesis se exploraron e implementaron técnicas incrementales de cálculo de tipo simbólico basadas en BDDs. También se incluyeron algoritmos para la optimización en el ordenamiento de las variables de los BDDs. Con estas nuevas técnicas se logró tener algoritmos que resuelven problemas de espacios de estados potenciales de hasta 10^{27} para controles supervisores y 10^{6} para controladores de procedimientos. Estas técnicas se incorporaron a la herramienta de síntesis de controladores SSPC [7, 20], la cual se toma como punto de partida para el desarrollo de este trabajo de tesis.

Una vez que se planteó un panorama general del trabajo a realizar, se presentan a continuación los principales objetivos:

1. La representación simbólica de las MEFs mediante BDDs, las cuales puedan ser expresadas mediante estructuras más compactas y eficientes, para lo cual se buscará la optimización en el ordenamiento de las variables de los BDDs de manera dinámica mediante el uso de las librerías de Buddy [18], el cual es el paquete que se utilizó para el manejo de los BDDs en la herramienta SSPC.

2. Utilizar un algoritmo incremental para el cálculo de la síntesis del controlador, basado en la TCS, que no necesite del producto síncrono entre la planta y la especificación, evitando la generación de estructuras de datos (BDDs) que ocupan gran espacio de memoria en la computadora para su procesamiento.

3. Usar un nuevo algoritmo incremental basado en predicados para mejorar el cálculo de la superestructura maximal para el controlador de procedimientos. Además de utilizar las técnicas de optimización antes mencionadas para el ordenamiento de variables en los BDDs con el fin de mejorar el cálculo del controlador de procedimientos.

4. Llevar a cabo una reestructuración de la herramienta SSPC que consista, principalmente, en la facilitación de la escritura de los archivos de entrada y la mejor interpretación de los archivos de salida, donde éstos describen el comportamiento de MEFs, las cuales pueden representar el modelo del sistema, el producto síncrono, el controlador, etc. Para esto se realizaron los siguientes cambios en la herramienta:

   - Generación de forma interna de los autolazos de las MEFs para facilitar la utilización de la herramienta a los usuarios. Esto debido a que en la versión 2, para introducir cada autolazo era necesario la escritura de un renglón en el archivo de entrada por cada transición y el estado al que se llega, algo que se repite para cada estado de partida y que puede ser tedioso y provocar la generación de errores al momento de la escritura de los archivos de entrada.

   - Que la herramienta etiquete todos los estados marcados en los archivos de salida, ya que hasta la versión 2 del SSPC el programa sólo puede identificar un solo estado marcado al momento de llevar a cabo la impresión de la MEF de salida.

1.3.2. Contenido de la tesis

El contenido de este trabajo de tesis está organizado de la siguiente manera.

En este capítulo se presentó una introducción de lo que es la síntesis de controladores para SEDs utilizando autómatas. Se habló sobre las diferentes alternativas que se han utilizado para el cálculo de los controladores, como lo son las técnicas explícitas y las simbólicas, estas últimas basadas en su mayoría en el manejo de BDDs. También se presentaron las principales herramientas que existen dedicadas a la síntesis de controladores para SEDs, y se dieron a conocer los avances que se han logrado hasta el momento en cuanto a los espacios de estados potenciales alcanzados para el cálculo de este tipo de controladores. Además, se presentaron las principales características de los trabajos de investigación realizados anteriormente por Reza y Douriet, los cuales se toman como el punto de partida para la realización de este trabajo. Finalmente, se establecieron los objetivos y las técnicas a implementar para conseguirlos.

En el capítulo 2 se presenta una breve introducción a la teoría de control supervisor (TCS) y a la teoría de control de procedimientos (TCP). Se incluyen algunas definiciones generales de las MEFs y sus principales características y propiedades. Se da un pequeño recordatorio de lo que son los BDDs y de la importancia que representa el ordenamiento que se esté usando para las variables del BDD, con el fin de obtener una estructura más compacta y eficiente para la representación simbólica de la información.

En el capítulo 3 se presenta el algoritmo de síntesis para SEDs utilizado para calcular el control supervisor. También se incluye el algoritmo usado para encontrar la superestructura maximal capaz de producir el supremo sublenguaje controlable.

En el capítulo 4 se describen todos los algoritmos y técnicas implementadas en la herramienta computacional SSPC para la optimización de la síntesis de controladores de SEDs. Aquí se presentan tanto el algoritmo para el cálculo incremental por especificaciones como el algoritmo de reordenamiento dinámico de variables de BDDs para su reducción de tamaño. Además se dan a conocer las funciones de Buddy usadas para llevar a cabo dicho reordenamiento dinámico.

En el capítulo 5 se demuestra la capacidad de cálculo alcanzada por la herramienta SSPC versión 3 para la obtención del control supervisor, la superestructura maximal y para el controlador de procedimientos. Se hace una comparación entre el desempeño de esta herramienta contra STCT [30] y la versión anterior de SSPC [7].

Finalmente, en el capítulo 6, se presentan los resultados finales obtenidos y las conclusiones en cuanto al alcance de la herramienta SSPC versión 3 y las comparaciones realizadas con otras herramientas para la síntesis de controladores. También se muestran las ventajas y desventajas de los nuevos algoritmos del SSPC contra los anteriormente implementados en esta misma herramienta, y finalmente se da a conocer el trabajo a realizar en el futuro para este tema de investigación.

Capítulo 2

Fundamentos teóricos

En este capítulo se mencionan las preliminares teóricas y las bases para la implementación de las técnicas involucradas en este trabajo.

Primeramente se presenta un poco de teoría relacionada con máquinas de estados finitos (MEFs) y la teoría de control supervisor, así como el supremo sublenguaje controlable y la superestructura del controlador de procedimientos.

Debido a que en la herramienta desarrollada se utilizan diagramas de decisión binarios (BDDs) para codificar los predicados que representan las MEFs, en este capítulo se muestra una breve introducción de lo que son los BDDs, así como la importancia del reordenamiento dinámico en BDDs y algunos trabajos y métodos existentes acerca del reordenamiento dinámico de BDDs. También se presenta una breve introducción referente a la teoría del cálculo incremental.

2.1. Máquinas de estados finitos (MEFs)

Una máquina de estados finitos (MEF) es una entidad matemática que consiste en un conjunto de estados y un conjunto de transiciones de estado a estado, que se dan sobre símbolos de entrada tomados de un alfabeto. La máquina comienza en un estado inicial y además contiene estados designados como marcados o de aceptación. A continuación se presentan algunas definiciones relacionadas con las MEFs.

Definición 2.1. Máquina de estados finitos (MEF)

Una máquina de estados finitos (MEF) es una tupla $\{Q, \Sigma, \delta, q_0, Q_m\}$ donde Q es un conjunto de estados, Σ es un alfabeto de símbolos de entrada, $\delta : Q \times \Sigma \rightarrow Q$ es una función de transición parcial (a cada elemento de esta función de transición se le llamará "evento"), q_0 es el estado inicial y $Q_m \subseteq Q$ es el conjunto de estados marcados o de aceptación.

Definición 2.2. Lenguaje de comportamiento cerrado

Se dice que una cadena s es generada por una MEF $M = \{Q, \Sigma, \delta, q_0, Q_m\}$ si $\delta(q_0, s) = p$ para algún $p \in Q$. El lenguaje de comportamiento cerrado de M denominado $L(M)$ es el conjunto $\{s \in \Sigma^* | \delta(q_0, s)!\}$.

El símbolo ! se refiere a que la función δ está definida para el estado y transición indicados.

Definición 2.3. Lenguaje marcado

Se dice que una cadena s es marcada si $\delta(q_0, s) = p$ para algún $p \in Q_m$. El lenguaje marcado de M denominado $L_m(M)$ es el conjunto $\{s \in \Sigma^* | \delta(q_0, s) \in Q_m\}$.

Con base en lo anterior, el comportamiento de una planta P está definido por los lenguajes $(L(P), L_m(P))$ y es lo que se considera como comportamiento a lazo abierto. Si se tiene un controlador C, el comportamiento a lazo cerrado está dado por $L(C/P) = L(P) \cap L(C)$ [28].

Finalmente, el comportamiento marcado a lazo cerrado está dado por $L_m(C/P) = L_m(P) \cap L(C)$ [28]. Es decir, el comportamiento marcado en lazo cerrado consiste en todas aquellas cadenas $s \in L_m(P)$ que sobreviven bajo el control de C.

En este trabajo se utilizará un formalismo de MEFs para el modelado del comportamiento de la planta y la especificación. El uso de las MEFs y los lenguajes generados por las mismas proveen un marco teórico adecuado para el problema de la síntesis de

controladores para SEDs.

Los eventos ocurren de manera instantánea y llevan al proceso de un estado a otro. Un evento se representa mediante la tripleta (q, σ, q') donde q es el estado actual del que se parte, σ es la transición y q' es el estado siguiente al que se llega. El estado q_0 es único, y el conjunto de estados marcados Q_m distingue a los estados que tienen un especial significado para el sistema, tal como la finalización de la tarea.

Definición 2.4. Alcanzabilidad

El subconjunto de estados alcanzables Q_{al} es el conjunto de estados que puede ser alcanzados desde el estado inicial. Esto es:

$$Q_{al} : \{q \in Q | \exists s \in \Sigma^*, \delta(q_0, s) = q\} \tag{2.1}$$

Definición 2.5. Coalcanzabilidad

El subconjunto de estados coalcanzables Q_{co} es el conjunto de estados desde los cuales se puede llegar a un estado marcado. Esto es:

$$Q_{co} : \{q \in Q | \exists s \in \Sigma^*, \delta(q, s) \in Q_m\} \tag{2.2}$$

Una MEF es alcanzable si todos los estados pueden ser alcanzados desde su estado inicial; es coalcanzable si todos sus estados pueden alcanzar al menos uno de los estados marcados. Si una MEF cumple con estas dos condiciones, se dice que es aseada [28].

Si en una MEF M el lenguaje generado por $L(M)$ se puede extender de tal forma que cada trayectoria del lenguaje se puede completar para alcanzar un estado marcado Q_M, es decir, que M sea alcanzable y coalcanzable, entonces se dice que M es no bloqueante. Esto es:

$$L(M) = \overline{L_m(M)} \tag{2.3}$$

2.2. Teoría de control supervisor

Considerando un SED modelado por un par de lenguajes, L y L_m, donde L es el conjunto de todas las cadenas que el SED puede generar y $L_m \subseteq L$ es el lenguaje de las cadenas de aceptación o cadenas que llevan a estados marcados, los cuales se usan para representar la culminación de operaciones o tareas. Sin perder generalidad, se asume que L y L_m son los lenguajes generados y marcados del autómata.

$$G = (Q, \Sigma, \delta, q_0, Q_m)$$

esto es,

$$\mathcal{L}(G) = L \text{ y } \mathcal{L}_m(G) = L_m$$

Entonces se debe adjuntar un supervisor, denotado por S, para interactuar con G a manera de retroalimentación, como se muestra en la figura 2.1.

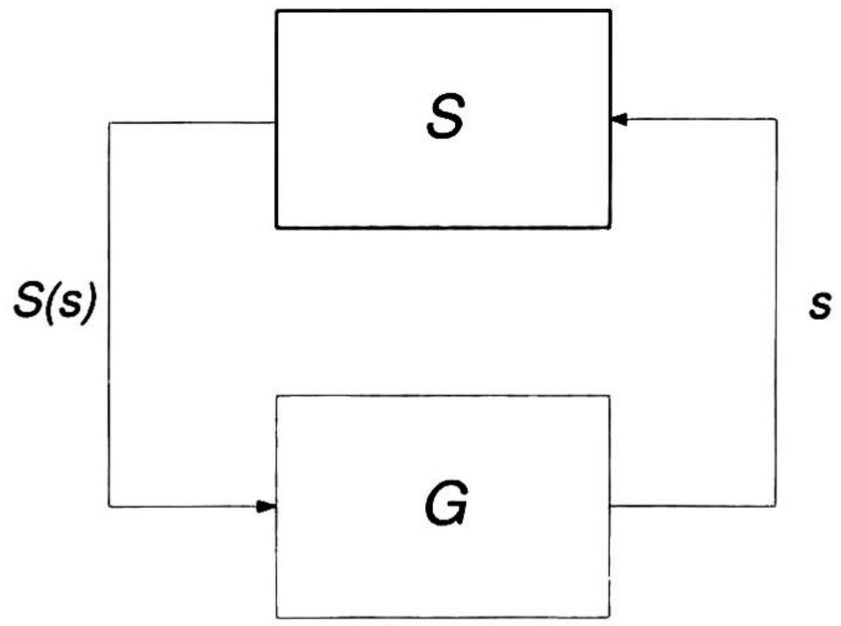

Figura 2.1: Control supervisor a lazo cerrado, con G representando el sistema incontrolable y S el supervisor

El conjunto de eventos o transiciones, representadas por Σ se divide en dos subconjuntos disjuntos: eventos controlables e incontrolables.

$$\Sigma = \Sigma_c \cup \Sigma_u$$

donde

- Σ_c es el conjunto de eventos controlables. Un evento controlable es aquél que puede ser inhabilitado por el supervisor S antes de que ocurra.

- Σ_u es el conjunto de eventos incontrolables. Este tipo de eventos no pueden ser inhabilitados por el supervisor S.

Hay muchas razones por las cuales un evento puede ser modelado como incontrolable: debido a que es inherentemente imprevisible (por ejemplo, una falla); un cambio en la lectura de sensores que no sea debido a un comando; o por elección, como por ejemplo cuando el evento tiene alta prioridad y no se desea inhabilitarlo.

El paradigma de control nos dice que la función de transición de G puede ser controlada por S en el sentido de que los eventos controlables de G pueden ser dinámicamente inhabilitados por S.

Teorema 2.1. ***Teorema de controlabilidad (CT)***

Dado el SED $G = (Q, \Sigma, \delta, q_0)$ *donde* $\Sigma_u \subseteq \Sigma$ *es el conjunto de eventos incontrolables. Sea* $K \subseteq \mathcal{L}(G)$, *donde* $K \neq \emptyset$. *Entonces existe un supervisor* S *tal que* $\mathcal{L}(S/G) = \overline{K}$ *si y sólo si*

$$\overline{K}\Sigma_u \cap \mathcal{L}(G) \subseteq \overline{K}$$

La prueba de este teorema se encuentra en [5].

Definición 2.6. Controlabilidad

Sean K y $L(G) = \overline{L(G)}$ lenguajes sobre el conjunto de eventos de Σ. Y sea Σ_u un subconjunto de Σ. Se dice que K es *controlable* con respecto a $L(G)$ y Σ_u si

$$\overline{K}\Sigma_u \cap L(G) \subseteq \overline{K} \tag{2.4}$$

Esto significa que la ejecución de un prefijo arbitrario de K, digamos s, seguido de un evento incontrolable σ, de manera que si $s\sigma$ es posible en la planta, entonces $s\sigma$ está contenido también dentro de los prefijos de K. Si K es controlable y cerrado con respecto a $L(G)$, entonces el autómata generador de K puede utilizarse como supervisor.

Teorema 2.2. ***Teorema de existencia***

Dado $K \subseteq \mathcal{L}_m(G), K \neq 0$. Existe un control supervisor no bloqueante S para G tal que $\mathcal{L}_m(S/G) = K$ si y sólo si

1. *K es controlable con respecto a G, y*

2. *K es $L_m(G)-$cerrado.*

La prueba de este teorema se encuentra en [28].

2.2.1. Supremo sublenguaje controlable

Si un lenguaje dado K no es controlable, entonces sería bueno encontrar el sublenguaje más grande de K, el cual es controlable. Sea $\mathcal{C}_{in}(K)$ una clase de sublenguaje controlable de K el cual es cerrado bajo la unión y que se define como sigue:

$$\mathcal{C}_{in}(K) := \{L \subseteq K : \overline{L}\Sigma_u \cap L(G) \subseteq \overline{L}\}$$

Tomando la union de todos los elementos de la clase $\mathcal{C}_{in}(K)$, y denotando el resultado como $K^{\uparrow}_{sup}$, se tiene que

$$K^{\uparrow}_{sup} := \bigcup_{L \in \mathcal{C}_{in}(K)} L.$$

Este resultado es un sublenguaje bien definido de K, por lo que se denota con el superíndice $\uparrow$, mientras que el superíndice C se refiere al hecho de que es controlable.

Que la clase $\mathcal{C}_{in}(K)$ sea cerrada bajo la unión significa que si K_1 y K_2 son dos elementos de $\mathcal{C}_{in}(K)$, entonces $K_1 \cup K_2$ también está en $\mathcal{C}_{in}(K)$. Por inducción se tiene que $\mathcal{C}_{in}(K)$ es cerrado en uniones finitas. Sin embargo, ya que la clase $\mathcal{C}_{in}(K)$ puede tener cardinalidad infinita se verifica si esto se mantiene para uniones infinitas. El paso clave que permite probar la cerradura de $\mathcal{C}_{in}(K)$ bajo la unión es la igualdad $\overline{K_1 \cup K_2} = \overline{K_1} \cup \overline{K_2}$. El cual se mantiene para uniones infinitas también, ya que

$$\overline{\cup_{i \in I} K_i} = \cup_{i \in I} \overline{K_i}$$

cuando I es un conjunto de índice infinito. Concluyendo que la controlabilidad se preserva bajo una cantidad de uniones arbitrarias y consecuentemente

$$K^{\uparrow}_{sup} \in C_{in}(K)$$

Ya que por definición $L \subseteq K^{\uparrow}_{sup}$ para cualquier $L \in C_{in}(K)$ se le llama a $K^{\uparrow}_{sup}$, el *supremo sublenguaje controlable* de K.

- En el peor caso, $K^{\uparrow}_{sup} = \emptyset$, ya que $\emptyset \in C_{in}(K)$.
- Si K es controlable, entonces $K^{\uparrow}_{sup} = K$.

2.3. Teoría de control de procedimientos

Los fundamentos de la teoría de control de procedimientos se basan en la teoría de control supervisor, con la diferencia de que en control de procedimientos se puede forzar la ejecución de ciertas transiciones. Esto es, el control de procedimientos establece en forma determinística las acciones de control a ejecutar.

A continuación se presentan algunas definiciones relacionadas con la teoría de control de procedimientos.

Definición 2.7. Controlador de procedimientos

Un controlador de procedimientos es una MEF $C = \{X, \Sigma, \gamma, x_0, X_m\}$, en la cual para cada $x \in X, \sigma \in \Sigma$ tal que si $\gamma(x, \sigma)!$, una de las siguientes condiciones es verdadera:

1. $\sigma \in \Sigma_u \wedge (\forall \sigma_c \in \Sigma_c, \gamma(x, \sigma_c)$ está indefinida).
2. $\sigma \in \Sigma_c \wedge (\forall \sigma' | (\sigma' \neq \sigma) \in \Sigma, \gamma(x, \sigma')$ está indefinida).

Definición 2.8. Completud

Una MEF $M = \{X, \Sigma, \gamma, x_0, X_m\}$ es completa con respecto a un proceso dado $P = \{Q, \Sigma, \delta, q_0, Q_m\}$ si para cada $s \in \Sigma^*$ y $\sigma_u \in \Sigma_u$, las condiciones $\gamma(x_0, s)!$ y $\delta(q_0, s\sigma_u)!$ implican que una de las siguientes condiciones es verdadera:

1. $\gamma(x_0, s\sigma_u)!$

2. $\exists \sigma_c \in \Sigma_c$ tal que $\gamma(x_0, s\sigma_c)! \wedge \gamma(x_0, s\Sigma_u)$ está indefinida.

Definición 2.9. Controlabilidad

Se dice que un lenguaje $K \subseteq L$ es controlable con respecto al lenguaje L si para toda $s \in K$ se cumple alguna de las siguientes afirmaciones:

1. $(s\Sigma_u \in L) \wedge (s\Sigma_u \in \overline{K})$

2. $\exists \sigma_c \in \Sigma_c$ tal que $(s\sigma_c \in \overline{K}) \wedge (\forall \sigma_u \in \Sigma_u \mid s\sigma_u \notin L \cap \overline{K})$

Definición 2.10. Supremo sublenguaje controlable

Dado un lenguaje $K \subseteq L$ existe un supremo sublenguaje controlable $K^{\uparrow}_{ss}$:

$$K^{\uparrow}_{ss} = \cup\{K' : K' \subseteq K \text{ y } K' \text{ es controlable con respecto a } L\}$$

Definición 2.11. Superestructura de controladores de procedimientos

Una máquina de estados finitos R es una superestructura de controlador de procedimientos si $L(R) \cap L(P)$ es controlable con respecto a $L(P)$.

Una superestructura de controladores de procedimientos es una MEF que conjunta todas las posibles estructuras candidatas a ser controlador de procedimientos.

Definición 2.12. Superestructura maximal de controladores de procedimientos

Una superestructura de un controladores de procedimientos R es maximal para un lenguaje de especificación K tal que $L(R) \subseteq K$, si se cumple que:

$$L(R) \cap L(P) = K^{\uparrow}_{ss}$$

Definición 2.13. Controlador completo no bloqueante

A partir de la superestructura maximal se obtiene el controlador de procedimientos, el algoritmo para obtenerlo se encuentra en [25] y está basado en el siguiente teorema.

Teorema 2.3. *Dado un lenguaje de especificación* $K(M) \subseteq L(P)$ *y* $K_m = L_m(P) \cap K$ *con* $K_m \neq \emptyset$. *Si* M *es aseada y* K_m *es cerrado y controlable, entonces existe* $C = \{X, \Sigma, \gamma, x_0, X_m\}$ *no bloqueante y completo con* $L(C/P) \subseteq \overline{K_m}$.

La prueba de este teorema se encuentra en [25] y se realiza por construcción.

2.4. Ordenamiento dinámico de variables para BDDs

En esta sección se da una breve introducción de lo que son los diagramas de decisión binarios (BDDs por sus siglas en inglés) y su codificación en la herramienta SSPC, mostrando la importancia del ordenamiento de las variables en cuanto al tamaño de la estructura del BDD. También se muestran los órdenes de variables anteriormente usados en SSPC. Finalmente, se explica lo que es el ordenamiento dinámico de variables, técnica utilizada en este trabajo.

2.4.1. Diagramas de Decisión Binarios (BDDs)

Un diagrama de decisión binario (BDD) es una eficiente representación para la manipulación de funciones booleanas. Un BDD es un grafo directo acíclico (DAG)

con un nodo raíz para cada función de salida y nodos finales que representan el valor de cada función para cada mintérmino de entrada. Cada nodo intermedio representa una variable de entrada, y cada una de las salidas del nodo representa el valor de la variable (0 o 1) a lo largo de esa rama. La representación se hace compacta al compartir subgrafos que sean comunes. Además de que para una función f y un *orden* de variables de entrada, el BDD es una forma canónica de f.

La manipulación de funciones booleanas es una parte importante de muchos algoritmos lógicos de síntesis, incluyendo optimización lógica y verificación lógica de circuitos combinacionales y secuenciales. Los BDDs han probado su gran utilidad en estas aplicaciones como una eficiente estructura de datos para la representación y manipulación de funciones booleanas.

Ya que el tamaño de un BDD puede ser exponencial en el número de entradas para muchos ordenamientos, en muchos casos prácticos un buen ordenamiento puede ser encontrado para obtener un BDD pequeño. Las operaciones funcionales en BDDs toman a lo más n^2 en espacio y tiempo (n es el número de nodos en el BDD); la comprobación entre la equivalencia entre dos BDDs requiere sólo de la comprobación de que los dos grafos son isomorfos. La propiedad de canonicidad de BDDs, implementada en paquetes eficientes para el manejo de BDDs, y recientes mejoras en las estrategias del ordenamiento de variables, han hecho a los algoritmos basados en BDDs eficientes y efectivos para una variedad de problemas que envuelven la manipulación de funciones booleanas.

En los trabajos realizados anteriormente por Reza [20] en el 2002, y por Douriet [7] en el 2003, los cuales se tomaron como base para este trabajo de tesis, se explica la teoría básica sobre la representación de fórmulas booleanas por medio de BDDs [3] y las funciones utilizadas para la manipulación de BDDs mediante el uso de la librería para C llamada Buddy, que fue desarrollada por Nielsen [18], y sobre la cual se explicarán más a adelante las funciones utilizadas en este trabajo para la aplicación del reordenamiento dinámico de BDDs en el software SSPC.

2.4.2. Orden de variables

El tamaño de un BDD se determina mediante el número de nodos que éste contiene. Entre más nodos tenga un BDD se necesita más espacio en memoria para almacenarlo. El tener BDDs de gran tamaño también afecta el tiempo que se requiere para realizar operaciones entre ellos; así que entre más pequeñas sean las estructuras de los BDDs manejados, menor tiempo le tomará al programa realizar las operaciones. El orden de las variables en un BDD es crucial para determinar el número de nodos del mismo.

Para mostrar esto se presenta un ejemplo tomado de [7], donde también se explica cómo se codifica una MEF como BDD.

Codificación de MEF a BDD

Para convertir una MEF a BDD, primero se debe expresar la información de la MEF como una expresión booleana. Después esta expresión se codifica como BDD. Para nuestro se toma la MEF de la figura 2.2.

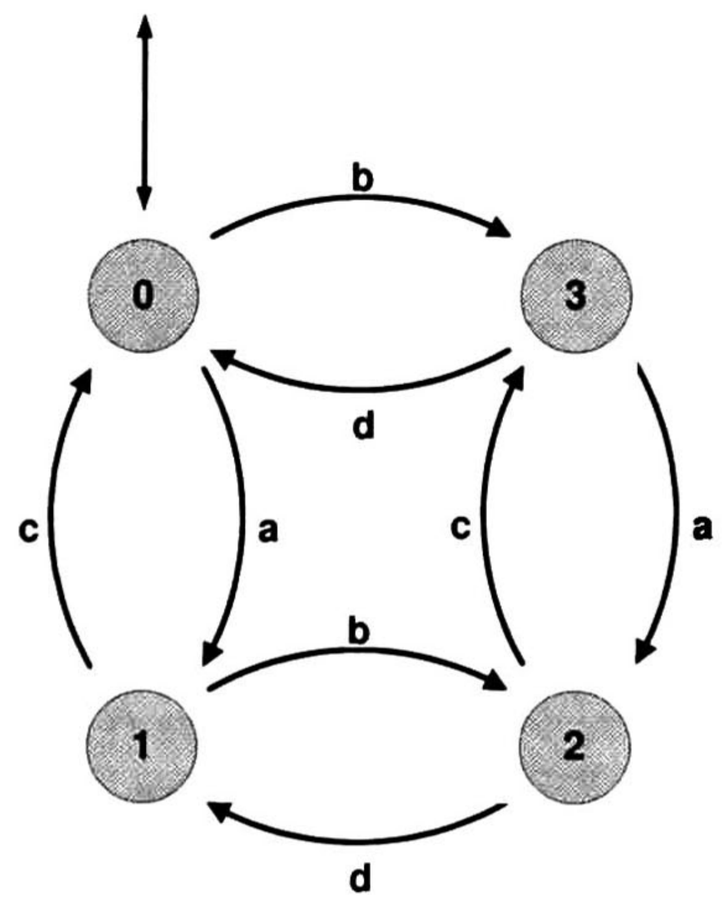

Figura 2.2: MEF a codificar como BDD

Para convertir esta MEF a una expresión booleana se asignan las siguientes variables booleanas:

- Codificación de estados actuales (q): x_1, x_2

- Codificación de estados siguientes (q'): y_1, y_2
- Codificación de transiciones (σ): s_1, s_2

En las tablas 2.1 y 2.2 se muestra cómo se codifica la información de la MEF en expresiones booleanas.

Estado	Código binario	q	q'
0	00	$\neg x_1 \wedge \neg x_2$	$\neg y_1 \wedge \neg y_2$
1	01	$\neg x_1 \wedge x_2$	$\neg y_1 \wedge y_2$
2	10	$x_1 \wedge \neg x_2$	$y_1 \wedge \neg y_2$
3	11	$x_1 \wedge x_2$	$y_1 \wedge y_2$

Tabla 2.1: Codificación de estados como expresiones booleanas

Estado	Código binario	σ
a	00	$\neg s_1 \wedge \neg s_2$
b	01	$\neg s_1 \wedge s_2$
c	10	$s_1 \wedge \neg s_2$
d	11	$s_1 \wedge s_2$

Tabla 2.2: Codificación de transiciones como expresiones booleanas

Para codificar un evento se hace la conjunción de las expresiones: estado actual, estado siguiente y transición. Para codificar la relación de transición de la MEF se hace la disyunción de todos sus eventos. De esta manera, la MEF mostrada en la figura 2.2 se representa por la siguiente expresión booleana:

$$
\begin{aligned}
&(\neg x_1 \wedge \neg x_2 \wedge \neg y_1 \wedge y_2 \wedge \neg s_1 \wedge \neg s_2) \vee \\
&(\neg x_1 \wedge x_2 \wedge \neg y_1 \wedge \neg y_2 \wedge s_1 \wedge \neg s_2) \vee \\
&(\neg x_1 \wedge x_2 \wedge y_1 \wedge \neg y_2 \wedge \neg s_1 \wedge s_2) \vee \\
&(x_1 \wedge \neg x_2 \wedge \neg y_1 \wedge y_2 \wedge s_1 \wedge s_2) \vee \\
&(x_1 \wedge \neg x_2 \wedge y_1 \wedge y_2 \wedge s_1 \wedge \neg s_2) \vee \\
&(x_1 \wedge x_2 \wedge y_1 \wedge \neg y_2 \wedge \neg s_1 \wedge \neg s_2) \vee \\
&(x_1 \wedge x_2 \wedge \neg y_1 \wedge \neg y_2 \wedge s_1 \wedge s_2) \vee \\
&(\neg x_1 \wedge \neg x_2 \wedge y_1 \wedge y_2 \wedge \neg s_1 \wedge \ s_2)
\end{aligned}
$$

En las figuras 2.3 y 2.4 se muestran BDDs que representan a la MEF de la figura 2.2 con diferentes órdenes de variables.

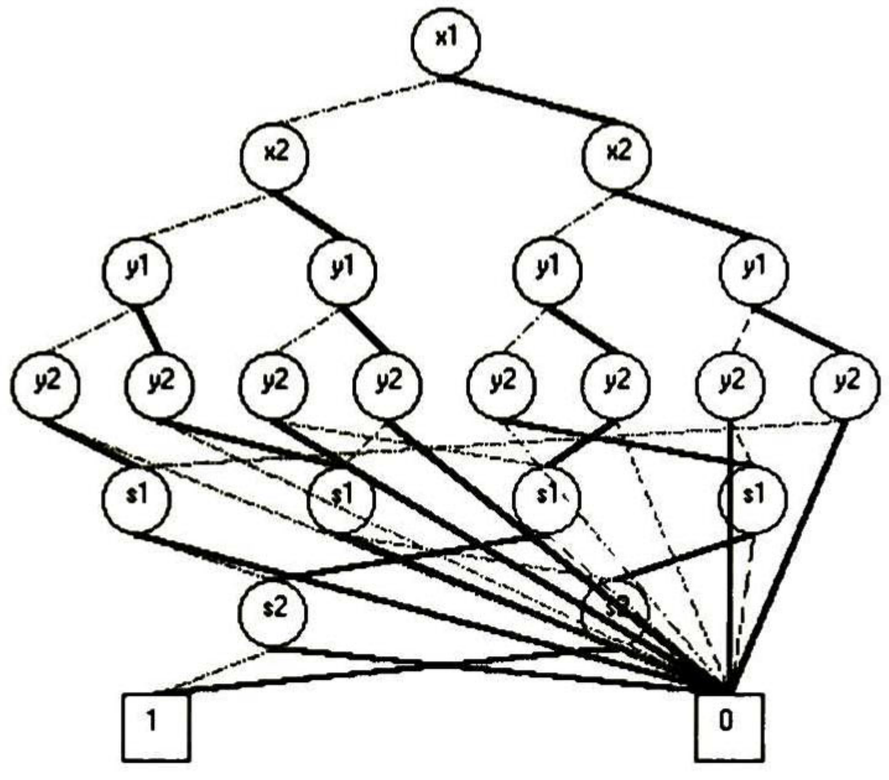

Figura 2.3: BDD correspondiente a la MEF de la figura 2.2 utilizando el orden de variables $x_1 < x_2 < y_1 < y_2 < s_1 < s_2$

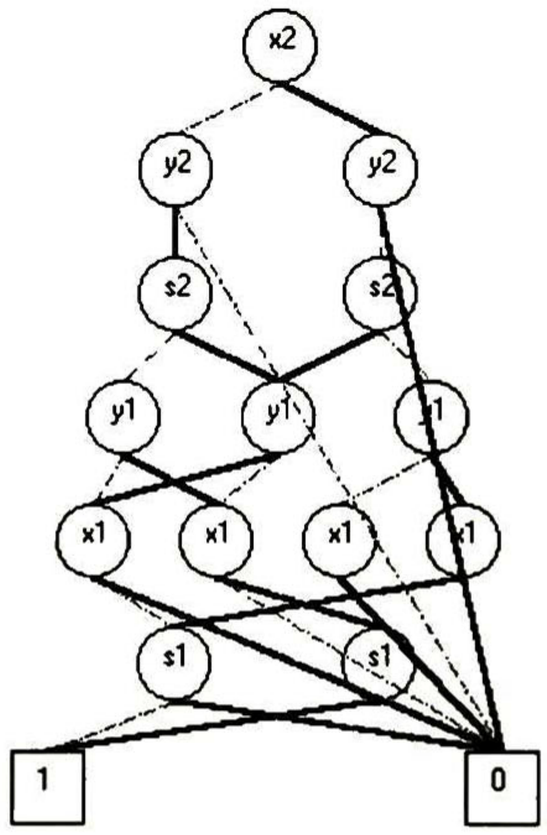

Figura 2.4: BDD correspondiente a la MEF de la figura 2.2 utilizando el orden de variables $x_2 < y_2 < s_2 < y_1 < x_1 < s_1$

Este ejemplo muestra que, con diferentes órdenes de variables, una misma expresión puede tener diferentes BDDs para representarla.

Orden de variables utilizado en las versiones anteriores del SSPC

El orden óptimo de variables para tener un BDD con una estructura lo más pequeña posible no se puede establecer con seguridad, pero existen heurísticas para lograr construir BDDs de manera ordenada, lo cual ayuda a tener estructuras pequeñas, mas no óptimas.

Inicialmente, en el trabajo realizado por Reza [20], debido a que el ordenamiento de las variables de los BDDs no era el objetivo principal de su trabajo de tesis, estableció un orden que facilitara la programación y la realización de operaciones mediante el uso de los BDDs en el SSPC. Para ello, las variables se agruparon en los siguientes bloques: variables para representar los estados actuales del sistema, variables para los estados siguientes del sistema, variables para los estados actuales de la especificación, variables para los estados siguientes de la especificación y variables para las transiciones.

El orden de variables utilizado en la versión 1 del SSPC fue: bits del estado actual del sistema, bits del estado siguiente del sistema, bits del estado actual de la especificación, bits del estado siguiente de la especificación, y bits de las transiciones. Esto se muestra en la figura 2.5.

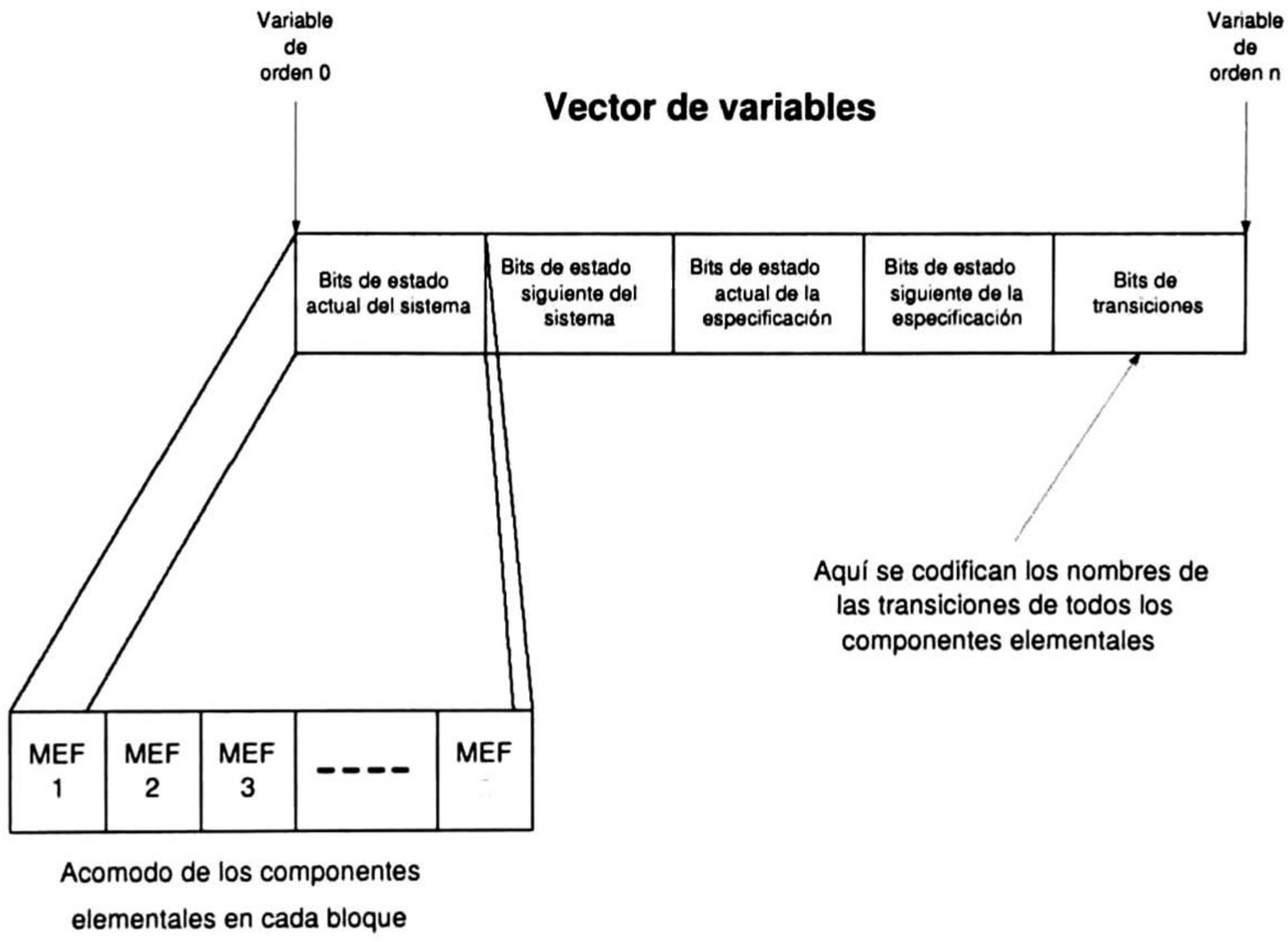

Figura 2.5: Orden de variables utilizado en SSPC versión 1

Ahora bien, si dos variables codifican información relacionada entre sí, entre más cercanas estén estas variables una de la otra, más pequeña es la estructura del BDD que se genera [9].

Así pues, Douriet [7] buscó diferentes arreglos para las variables de los BDDs tratando de acercar aquellas que codifican información relacionada; en este caso es obvio que las variables que codifican las transiciones son las que más interactúan con otras variables, ya que relacionan las variables de estado actual con las de estado siguiente tanto del sistema como de la especificación. Sin embargo, para facilitar la programación y la realización de las operaciones entre BDDs, era conveniente mantener las variables agrupadas en los bloques descritos anteriormente: variables para estados actuales del sistema, estados siguientes del sistema, estados actuales de la especificación, estados siguientes de la especificación y transiciones.

Entonces se buscó un punto intermedio entre estos dos aspectos, es decir, buscar un orden de variables que minimice el tamaño de los BDDs a utilizar, pero que al mismo tiempo mantenga los bloques de variables de manera ordenada.

Después de intentar con varios arreglos entre los bloques de variables, encontró un orden que mostró mejoras significativas en el desempeño del programa.

Douriet utilizó dos bloques principales, el de estados actuales-siguientes y el de transiciones. Y el de estados actuales-siguientes a su vez se divide en dos sub-bloques, uno para el sistema y otro para la especificación. Utilizando este arreglo de variables, cada estado actual está junto a su respectivo estado siguiente. Esto se muestra gráficamente en la figura 2.6.

2.4.3. Ordenamiento dinámico de variables

En los BDDs un orden total es impuesto a las variables, es decir, un lugar es asignado para cada variable en el BDD, y las variables deben aparecer en orden ascendente a lo largo de cada ruta o camino del BDD. Debido a que el BDD está ordenado, el grafo directo acíclico (DAG) tiene varios niveles con todos los nodos de una variable en particular en cada nivel, donde el nivel i se refiere a todos los nodos con la variable x_i.

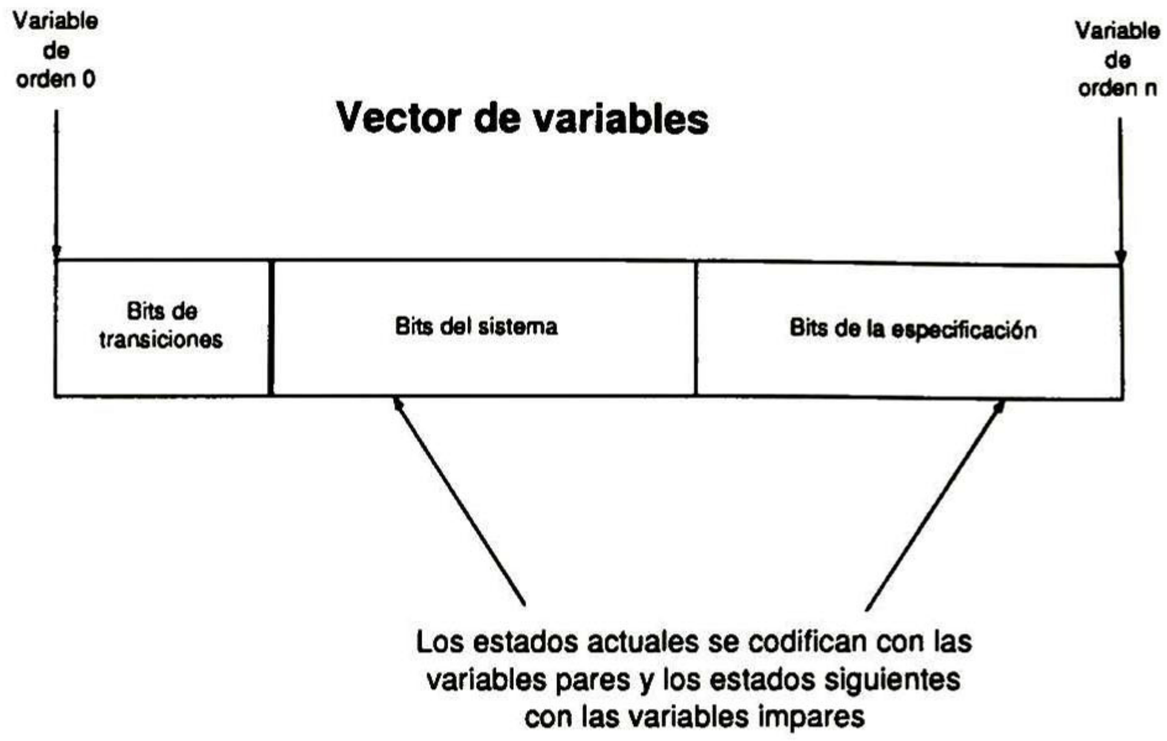

Figura 2.6: Orden de variables utilizado en SSPC versión 2

Mediante el ordenamiento dinámico de variables de los BDDs el orden de las variables es cambiado automáticamente por el programa usado para el manejo de BDDs de forma transparente para el usuario; el orden de las variables dentro del BDD ya no es estático. Como se ve, el ordenamiento dinámico de variables difiere del uso típico de los BDDs donde las variables son ordenadas cuando el BDD es creado y mantienen ese orden a lo largo de todo el proceso subsecuente. Así pues, este reordenamiento de variables se da cuando el programador considera que es requerido.

Al usar el ordenamiento dinámico de variables, un orden completo es definido para todas las variables después de cada operación de reordenamiento; el orden de variables es periódicamente ajustado por el programa como consecuencia de las operaciones de reordenamiento realizadas, con el fin de encontrar un mejor orden, manteniéndose todas las ventajas provistas por este tipo de estructuras de datos, tales como la canonicidad y algoritmos recursivos eficientes.

La única desventaja es que, si por alguna razón se requiere mantener un orden fijo en las variables de los BDDs para la obtención de cierta información, el ordenamiento dinámico de variables no es conveniente.

2.4.4. Diferentes tipos de reordenamiento de variables

Con el fin de conseguir un buen orden de variables en los BDD que permitan trabajar con estructuras más compactas y que ocupen menos espacio en memoria, han surgido varias heurísticas para el reordenamiento de variables. Algunos ejemplos de las principales heurísticas existentes son la de profundidad (depth-first) y la de anchura (breadth-first) [19]. También se ha buscado la reducción en el tamaño de BDDs a través de la identificación de variables dependientes dentro del BDD, las cuales pueden ser expresadas como función de otras variables [10]. Además, se han buscado otras técnicas para la realización de operaciones con BDDs como la paralelización mediante el uso de varias estaciones de trabajo al mismo tiempo [17].

Sin embargo, el tipo de reordenamiento de variables más sencillo, y a la vez más utilizado, es el que consiste en cambiar de posición dos variables que se encuentren en niveles adyacentes (swapping). Dicha heurística se llama permutación de ventana (sliding window o window permutation). Lo malo de este tipo de reordenamiento es que sólo va afectando a los nodos correspondientes a las dos variables que intercambian posiciones. Basado en esta heurística surge el reordenamiento sifting [21] introducido por Rudell, y utilizado en este trabajo, por lo cual se explicará más a fondo en el capítulo 4.

La idea principal de este algoritmo llamado sifting consiste en encontrar la mejor posición para cada variable. Las variables individuales se mueven a todos los niveles del BDD una por una. La operación básica para la implementación de este movimiento es el cambio de lugar de dos variables adyacentes (operación swapping), el cual se puede realizar rápidamente si se usa una representación apropiada del BDD.

Este algoritmo es conocido como el estado del arte de las heurísticas de reordenamiento. Ya que no requiere ninguna información adicional de la función representada, puede ser aplicable en cualquier momento sobre los BDDs de forma exitosa. Sin embargo, este algoritmo tiene una limitación, mientras que un swapping puede ser muy rápido, el tiempo total del algoritmo crece rápidamente conforme aumenta el número de variables. Así pues, en aplicaciones con varias miles de variables, el sifting de una sola variable, sobre todo el ordenamiento, puede tomar mucho tiempo.

En años más recientes, Meinel y Slovodová [16] lograron hacer una preestimación de la posible reducción en tamaño del BDD usando este algoritmo, y propusieron un método llamado *sifting restringido a bloques.* La idea principal consiste en la restricción de la operación swap a ciertos bloques de variables que se determinan de acuerdo a algunas propiedades estructurales de los BDDs, las cuales son consideradas por complejas consideraciones teóricas. Esto mejoró considerablemente el tiempo de realización del algoritmo sifting original sin causar pérdidas substanciales en el tamaño final de los BDDs. Dicho trabajo no fue utilizado debido a que no había la suficiente información sobre las consideraciones y los complejos cálculos matemáticos que deben realizarse para obtener la preestimación de la reducción del tamaño de los BDDs al aplicar el algoritmo sifting. Así pues, habría que considerar este método para trabajos futuros.

Capítulo 3

Algoritmos de síntesis de controladores para SEDs

En este capítulo se presenta el algoritmo de síntesis utilizado para el cálculo del control supervisor, el cual se toma del trabajo realizado por Zhang [30] y sirve para extender el trabajo realizado por Reza [20] en cuanto a la síntesis de supervisores usando predicados estructurales. También se presenta un algoritmo propuesto para el cálculo de la superestructura del controlador de procedimientos el cual se agrega de igual forma al trabajo de Reza [20].

3.1. Algoritmo de síntesis del control supervisor para SEDs usando predicados estructurales

3.1.1. Algoritmo de síntesis para un SED simple

El algoritmo presentado en esta sección para la síntesis del control supervisor es tomado, como ya se comentó anteriormente, del trabajo realizado por Zhang [30]. Supongamos que se tiene un SED simple, cuya planta y especificación están dadas en forma de autómatas simples:

- Planta: $G_p = (X, \Sigma, x_0, \xi, X_m)$.
- Especificación: $G_s = (Y, \Sigma, y_0, \eta, Y_m)$.

La sincronización entre las máquinas G_p y G_s se expresa como

$$G_{sync}(G_p, G_s) = (X \times Y, \Sigma, (x_0, y_0), \xi \times \eta, X_m \times Y_m).$$

Para establecer condiciones en el estado (x, y) de G_{sync}, se usa un formalismo lógico. Un *predicado* P definido sobre $X \times Y$ es una función $P : X \times Y \longrightarrow 0, 1$. $\|P\|$ puede ser identificado por el correspondiente subconjunto de estados.

$$\|P\| := \{(x, y) \in X \times Y | P(x, y) = 1\}.$$

El predicado P está definido sobre un conjunto bidimensional y tiene la información acerca de las condiciones que hay en los estados, y también la información estructural acerca del sistema. Los operadores lógicos definidos para predicados de sistemas simples [20] pueden ser extendidos a este tipo de predicados. Para mayor claridad, si $P(x, y) = 1$ se utilizará $(x, y) \models P$, de otra forma $(x, y) \not\models P$.

Se define el lenguaje cerrado $L(G_{sync}, P)$ inducido por un predicado $P \in Pred(X \times Y)$ como

$$L(G_{sync}, P) := \{w \in L(G_{sync}) | (\forall v \leq w)(\xi(x_{p0}, v), \eta(y_{s0}, v)) \models P\}$$

y el lenguaje marcado $L_m(G_{sync}, P)$ inducido por un predicado $P \in Pred(X \times Y)$ como

$$L_m(G_{sync}, P) := \{w \in L(G_{sync}, P) | (\xi(x_{p0}, w), \eta(y_{s0}, w)) \in X_m \times Y_m\}.$$

Ahora se establecerán algunas definiciones para ayudar a introducir las propiedades de controlabilidad y de no bloqueo relacionadas al predicado estructural P

Definición 3.1. Alcanzabilidad de un estado

Para un P dado, un estado (x, y) es *alcanzable* ssi

$$(\exists w \in L(G_{sync}, P))(\xi(x_0, w) = x \text{ y } \eta(y_0, w) = y)$$

Definición 3.2. Coalcanzabilidad de un estado

Para un P dado, un estado (x, y) es *coalcanzable* ssi

$$(\exists w \in \Sigma^*)(\xi(x, w) \in X_m \text{ y } \eta(y, w) \in Y_m \text{ y } (\forall v \leq w)(\xi(x, v), \eta(y, v)) \models P)$$

Definición 3.3. Controlabilidad de un predicado estructural P

Para una G_p y una G_s dadas, P es *controlable* ssi $L(G_{sync}, P)$ es controlable con respecto a $L(G)$.

Definición 3.4. Propiedad de no bloqueo de un predicado estructural P

Para una G_p y una G_s dadas, P es *no bloqueante* ssi $\overline{L_m(G_{sync}, P)} = L(G_{sync}, P)$.

Ahora se pueden establecer dos lemas para predicados estructurales:

Lema 3.1. *Para una G_p y una G_s dadas, P es controlable ssi*

$$(\forall (x, y) \models P, \forall \sigma \in \Sigma_u)(\xi(x, \sigma)!) \Longrightarrow (\eta(y, \sigma)!\ y\ (\xi(x, \sigma), \eta(y, \sigma)) \models P)$$

La prueba de este lema se encuentra en [30].

Lema 3.2. *Para una G_p y una G_s dadas, P es no bloqueante ssi*

$$(\forall (x, y) \models P)((x, y)\ es\ alcanzable) \Longrightarrow ((x, y) es\ coalcanzable)$$

La prueba de este lema también se encuentra en [30].

Ahora se presenta el siguiente teorema como resultado como resultado de los lemas y las definiciones anteriores:

Teorema 3.1. *Dadas G_p y G_s, existe un predicado estructural P_{sup} tal que*

1. $L_m(G_{sync}, P_{sup}) = sup\mathcal{C}(L(G_p), L_m(G_s))$.

2. P_{sup} *es no bloqueante.*

La prueba se realiza constructivamente como sigue:

Dadas G_p y G_s, se puede construir P_{sup} como sigue:

1. Se inicializa $P_{good} = 1$.

2. $\|P_{bad1}\| = \{(x,y) \models P_{bad1}\}$ ssi $((x,y) \not\models P_{good})$
   o $(\exists \sigma \in \Sigma_u)(\xi(x,\sigma)!$ y $\neg\eta(y,\sigma)!)$
   o $(\exists \sigma \in \Sigma_u)((\xi(x,\sigma)!, \eta(y,\sigma)!)$ y $(\xi(x,\sigma), \eta(y,\sigma)) \not\models P_{good})$

3. $\|P_{bad2}\| = \{(x,y) \models P_{bad2}\}$ ssi $(\exists w \in \Sigma_u^*)((\xi(x,w), \eta(y,w)) \models P_{bad1})$

4. $\|P_{al}\| = \{(x,y) \models P_{al}\}$ ssi
   $(\exists w \in \Sigma^*)$
   $(\xi(x_0,w) = x, \eta(y_0,w) = y)$ y $(\forall v < w)$ $((\xi(x_0,v), \eta(y_0,v)) \not\models (P_{bad1} \cup P_{bad2}))$

5. $\|P_{co}\| = \{(x,y) \models P_{co}\}$ ssi
   $(\exists w \in \Sigma^*)$
   $(\xi(x,w) \in X_m, \eta(y,w) \in Y_m)$ y $(\forall v < w)((\xi(x,v), \eta(y,v)) \not\models (P_{bad1} \cup P_{bad2}))$

6. $\|P_{newgood}\| = \|P_{al}\| \cap \|P_{co}\|$

7. Si $\|P_{newgood}\| \subset \|P_{good}\|$ se repiten los pasos del 2 al 7 con $\|P_{good}\| = \|P_{newgood}\|$, sino $\|P_{sup}\| = \|P_{newgood}\|$ y termina el algoritmo.

La prueba de que P_{sup} es lo que se busca se encuentra en [30].

Este algoritmo termina a lo más en $|X| \times |Y|$ iteraciones y logra la obtención de supervisores para sistemas grandes debido al manejo de la información de los SEDs mediante el uso de predicados estructurales.

3.1.2. Algoritmo de síntesis para SEDs complejos

El algoritmo descrito en la sección anterior puede ser fácilmente extendido a SEDs complejos.

Supongamos que tenemos un SED complejo, cuya planta y especificación están dadas en forma del producto síncrono entre más de un autómata:

- Planta: $G_p = sync(G_{pi})$. $G_{pi} = (X_i, \Sigma, x_{0i}, \xi_i, X_{mi})$. $i \in 1, \ldots, N_p$.

- Especificación: $G_s = sync(G_{sj})$. $G_{sj} = (Y_j, \Sigma, y_{0j}, \eta_j, Y_{mj})$. $j \in 1, \ldots, N_s$.

Entonces se puede mostrar, como en la sección anterior, que la síntesis e implementación del controlador se puede hacer mediante un predicado estructural de $N_p + N_s$ variables:

Teorema 3.2. *Dadas G_{pi} y G_{sj}, existe un predicado estructural P_{sup} tal que*

1. $L_m(G_{sync}, P_{sup}) = supC(L(sync(G_{pi})), L_m(sync(G_{sj})))$.

2. P_{sup} *es no bloqueante.*

La prueba es idéntica a la del teorema anterior, excepto que es para una descripción de MEFs con mayor cantidad de estados. Esta prueba también se encuentra en [30].

Como se demostró en el teorema anterior se puede llevar a cabo la síntesis de supervisores para SEDs complejos de la misma manera que para SEDs simples tomando el producto asíncrono de los componentes elementales de la planta como G_p, y el producto síncrono de los componentes de la especificación como G_s. Esto hace pensar en la posibilidad llevar a cabo la síntesis de manera incremental con respecto a las especificaciones.

Una de las opciones para llevar a cabo la síntesis de forma incremental es con las especificaciones por separado, es decir, realizar la síntesis para el sistema completo con cada una de las j especificaciones por separado, con lo cual obtendremos j conjuntos de estados controlables, alcanzables y coalcanzables correspondientes a cada $\|P_{sup-j}\|$ calculado. Una vez que se tienen estos j conjuntos de estados realizar su intersección para encontrar un $\|P_{sup}\|$ que contenga a los estados que sean controlables en todos los $\|P_{sup-j}\|$. El problema aquí es que la intersección de MEFs aseadas no garantiza que la MEF resultante sea aseada, por lo que habría que revisar si todos los estados de $\|P_{sup}\|$ son alcanzables y coalcanzables. En caso de que así fuera ya tenemos el supervisor que buscamos, de lo contrario habría que aplicar el algoritmo con G_s como el producto síncrono de todas las especificaciones y tomando a $\|P_{sup}\|$ como el conjunto de estados

$\|P_{good}\|$, ya que son estados controlables con respecto a todas las especificaciones y así se aprovechan los cálculos hechos anteriormente.

Otra opción para realizar la síntesis de manera incremental es con especificaciones sucesivas, la cual consiste en realizar la síntesis de igual manera para la primera especificación, y después realizar la síntesis para la segunda especificación tomando como $\|P_{good}\|$ a los estados obtenidos en $\|P_{sup-1}\|$, y así sucesivmente con todas las especificaciones hasta obtener $\|P_{sup}\|$ en la iteración j, donde todos los estados serán controlables, alcanzables y coalcanzables.

De las anteriores opciones propuestas para realizar la síntesis del supervisor la que obtiene mejores resultados es la forma incremental con las especificaciones por separado. Su descripción y los resultados obtenidos se mostrarán en los siguientes capítulos.

3.2. Algoritmo para la obtención de los estados de la superestructura usando predicados estructurales

3.2.1. Algoritmo para un SED simple

De acuerdo con la teoría anteriormente presentada acerca del supremo sublenguaje controlable y de la superestructura maximal del controlador de procedimientos, y tomando como base el teorema desarrollado por Zhang para la síntesis de un control supervisor para un SED simple se propone el siguiente teorema:

Teorema 3.3. *Dadas G_p y G_s, existe un predicado estructural P_{struc} tal que contiene todos los estados de la superestructura del control de procedimientos.*

La prueba se realiza constructivamente y se basa en la prueba realizada para el teorema presentado para la síntesis del control supervisor:

Dadas G_p y G_s, se puede construir P_{struc} como sigue:

1. Se inicializa $P_{good} = 1$.

2. $\|P_{bad}\| = \{(x,y) \models P_{bad}\}$ ssi $((x,y) \not\models P_{good})$
   o $(\exists \sigma \in \Sigma_u)(\xi(x,\sigma)!$ y $\neg\eta(y,\sigma)!)$
   o $(\exists \sigma \in \Sigma_u)((\xi(x,\sigma)!, \eta(y,\sigma)!)$ y $(\xi(x,\sigma), \eta(y,\sigma)) \not\models P_{good})$

3. $\|P_{bad-cont}\| = \{(x,y) \models P_{bad} | (((x,y) \models P_{good})$ y $(\exists \sigma \in \Sigma_c)(\xi(x,\sigma)!$ y $\eta(y,\sigma)!))\}$

4. $\|P_{erase1}\| = \|P_{bad}\| - \|P_{bad-cont}\|$

5. $\|P_{erase2}\| = \{(x,y) \models P_{erase2}\}$ ssi $(\exists w \in \Sigma_u^*)((\xi(x,w), \eta(y,w)) \models P_{erase1})$ y
   $(\nexists \overline{w}\sigma_c$ para cualquier $\sigma_c \in \Sigma_c)$ $((\xi(x,\overline{w}\sigma_c), \eta(y,\overline{w}\sigma_c)) \not\models P_{erase1})$

6. $\|P_{al}\| = \{(x,y) \models P_{al}\}$ ssi
   $(\exists w \in \Sigma^*)(\xi(x_0,w) = x, \eta(y_0,w) = y)$ y
   $(\forall v < w)$ $((\xi(x_0,v), \eta(y_0,v)) \not\models (P_{erase1} \cup P_{erase2}))$

7. $\|P_{co}\| = \{(x,y) \models P_{co}\}$ ssi
   $(\exists w \in \Sigma^*)(\xi(x,w) \in X_m, \eta(y,w) \in Y_m)$ y
   $(\forall v < w)((\xi(x,v), \eta(y,v)) \not\models (P_{erase1} \cup P_{erase2}))$

8. $\|P_{newgood}\| = \|P_{al}\| \cap \|P_{co}\|$

9. Si $\|P_{newgood}\| \subset \|P_{good}\|$ se repiten los pasos del 2 al 9 con $\|P_{good}\| = \|P_{newgood}\|$.
   Si $\|P_{newgood}\| = \|P_{good}\|$, entonces $\|P_{struc}\| = \|P_{newgood}\|$ y termina el algoritmo.

Ahora se necesita probar que $\|P_{struc}\|$ es lo que buscamos.

- El algoritmo termina en un número finito de iteraciones.
  Se define a $|P|$ como el número de estados asociados al predicado estructural P.
  La cantidad de estados totales que $|P_{struc}|$ podría llegar a tener como máximo es de $|X| \times |Y|$, por lo que en cada ciclo (pasos 2 al 9), si el algoritmo no termina, $|P_{erase1} \cup P_{erase2}|$ se incrementará al menos un estado. De ahí que el algoritmo puede repetirse a lo más $|X| \times |Y|$ veces, lo cual lo hace finito.

- $L_m(G_{sync}, P_{struc}) \subseteq K^{\uparrow}_{ss}(L(G_p), L_m(G_s))$.

$\|P_{struc}\|$ es no bloqueante.

Esto debido a que cuando el algoritmo termina siempre se tendrá que $\|P_{good}\| = \|P_{newgood}\| = \|P_{al}\| = \|P_{co}\|$, es decir, cada estado que satisface P_{struc} es alcanzable y coalcanzable.

P_{struc} es controlable de acuerdo a la teoría de control de procedimientos.

Supongase que $(\exists(x', y') \models P_{struc})((\exists\sigma_u \in \Sigma_u)(\xi(x', \sigma_u)!$ y $\neg(\eta(y', \sigma_u)!)))$ y $((\nexists\sigma_c \in \Sigma_c)((\xi(x', \sigma_c)!$ y $\eta(y', \sigma_c)!)))$, entonces de acuerdo a los pasos 2, 3 y 4 $(x', y') \models P_{erase1}$, lo cual contradice que $(x', y') \models P_{struc}$. Así tenemos que $((\forall(x, y) \models P_{struc}), (\forall\sigma_u \in \Sigma_u)(\xi(x, \sigma_u)!) \longrightarrow (\eta(y, \sigma)!))$.

Ahora suponiendo que $(\exists(x', y') \models P_{struc})(((\exists\sigma_u \in \Sigma_u)(\xi(x', \sigma_u)!, \eta(y', \sigma_u)!))$, $((\nexists\sigma_c \in \Sigma_c)(\xi(x', \sigma_c), \eta(y', \sigma_c)) \models P_{good})$ y $((\xi(x', \sigma_u), \eta(y', \sigma_u)) \not\models P_{good}))$ entonces $(x', y') \models P_{erase2}$, lo cual contradice que $(x', y') \models P_{struc}$.
Así tenemos que $((\forall(x, y) \models P_{struc})((\exists\sigma_c \in \Sigma_c)(\xi(x, \sigma_c)!, \eta(y, \sigma_c)!)$ y $((\forall\sigma_u \in \Sigma_u)(\xi(x, \sigma_u)!, \neg(\eta(y, \sigma_u)!))))$

Entonces $L_m(G_{sync}, P_{struc}) \subseteq K^{\uparrow}_{ss}(L(G_p), L_m(G_s))$.

- $L_m(G_{sync}, P_{struc}) \supseteq K^{\uparrow}_{ss}(L(G_p), L_m(G_s))$.

Ahora se debe probar que $(\forall K' \subseteq L(G_{sync})(K'$ es controlable de acuerdo a la teoría de control de procedimientos y es no bloqueante) $K' \subseteq L_m(G_{sync}, P_{struc}))$

Suponiendo lo contrario, entonces podemos encontrar una K' la cual sea controlable y no bloqueante, pero $K' \nsubseteq L_m(G_{sync}, P_{struc})$.
Entonces $(\exists s \in K')(s \notin L_m(G_{sync}, P_{struc}))$, donde s es una cadena de eventos.
Haciendo $x = \xi(x_0, s)$ y $y = \eta(y_0, s)$. Entonces $(x, y) \not\models P_{struc}$.

El algoritmo es iterativo y existe un ciclo, pasos de 2 al 9. Para mayor claridad se puede escribir $P_{good(i)}$, $P_{newgood(i)}$, $P_{bad1(i)}$, y $P_{bad2(i)}$ para los valores del ciclo

i. Por ejemplo, se escribe $P_{bad1(2)}$ para el valor de P_{bad1} en el segundo ciclo.

Ahora se mostrará que si $(\exists s \in K')((\xi(x_0, s), \eta(y_0, s)) \not\models P_{newgood(n)})$ entonces $(\exists s' \in K')((\xi(x_0, s'), \eta(y_0, s')) \not\models P_{good(n)})$

Haciendo $(x_1, y_1) = (\xi(x_0, s), \eta(y_0, s))$. Si $(x_1, y_1) \not\models P_{good(n)}$, entonces $s' = s$ que es lo que necesitamos. De otra forma, es decir, si $(x_1, y_1) \models P_{good(n)}$ tendremos que $(x_1, y_1) \not\models P_{al(n)}$ o $(x_1, y_1) \not\models P_{co(n)}$ o ambas. Suponiendo que $(x_1, y_1) \not\models P_{al(n)}$. Entonces también $(x_1, y_1) \models (P_{erase1(n)} \cup P_{erase2(n)})$ o $(\exists w \leq s)((\xi(x_0, w), \eta(y_0, w)) \models (P_{erase1(n)} \cup P_{erase2(n)}))$. Si esto es verdadero, se hace $(x_2, y_2) = (x_1, y_1)$ y $s_1 = s$; o de otra forma $(x_2, y_2) = (\xi(x_0, w), \eta(y_0, w))$ y $s_1 = w$. En este último caso tendríamos que $w \in K'$ también. Un razonamiento similar se puede aplicar si $(x_1, y_1) \not\models P_{co(n)}$. Ahora tenemos que $(\exists s_1 \in K')((x_2, y_2) = (\xi(x_0, s_1), \eta(y_0, s_1)), (x_2, y_2) \models (P_{erase1(n)} \cup P_{erase2(n)}))$

Si $(x_2, y_2) \not\models P_{erase1(n)}$, entonces se tiene que $(x_2, y_2) \models P_{erase2(n)}$. Así entonces $(\exists s_2 \in K^*)((\xi(x_2, s_2), \eta(y_2, s_2)) \models P_{erase1(n)})$

Ahora haciendo $(x_3, y_3) = (\xi(x_2, s_2), \eta(y_2, s_2))$ y $s_3 = s_1 s_2$; de otra forma $((x_2, y_2) \models P_{erase1(n)})$, se hace $(x_3, y_3) = (x_2, y_2)$ y $s_3 = s_1$. Y ahora tenemos $(\exists s_3 \in K')((x_3, y_3) = (\xi(x_0, s_3), \eta(y_0, s_3)), (x_3, y_3) \models P_{erase1(n)})$

Ahora se tienen los siguientes casos posibles:

- $(x_3, y_3) \not\models P_{good(n)}$. Entonces s_3 es lo que necesitamos.

  $(\exists \sigma_u \in \Sigma_u)((\xi(x_3, \sigma_u)!)(\neg(\eta(y_3, \sigma_u)!)))$ y $(\nexists \sigma_c \in \Sigma_c)(\xi(x_3, \sigma_c)!$ y $\eta(y_3, \sigma_c)!)$. Esto es imposible porque esto haría K' incontrolable.

  $(\exists \sigma \in \Sigma)(\xi(x_3, \sigma)!, \eta(y_3, \sigma)!$ y $(\xi(x_3, \sigma), \eta(y_3, \sigma)) \not\models P_{good(n)})$. Así debemos tener que $s_3\sigma \in K'$. También tenemos que $(\xi(x_0, s_3\sigma), \eta(y_0, s_3\sigma)) \not\models P_{good(n)}$. Así $s_3\sigma$ es lo que necesitamos.

Debido a que $P_{good(n)} \equiv P_{newgood(n-1)}$, este proceso puede continuar inductivamente hasta que finalmente se llegue a $n = 1$, es decir, debe ser verdad que $(\exists s' \in K')((\xi(x_0, s'), \eta(y_0, s')) \not\models P_{good(1)})$

Pero esto es imposible, porque $P_{good(1)} \equiv 1$. Con lo que concluye la prueba.

□

3.2.2. Algoritmo para SEDs complejos

De igual forma que en la síntesis del supervisor, el algoritmo descrito en la sección anterior para la síntesis de la superestructura puede ser fácilmente extendido a SEDs complejos.

Supongamos que tenemos un SED complejo, cuya planta y especificación están dadas en forma del producto síncrono entre más de un autómata:

- Planta: $G_p = sync(G_{pi})$. $G_{pi} = (X_i, \Sigma, x_{0i}, \xi_i, X_{mi})$. $i \in 1, \ldots, N_p$.
- Especificación: $G_s = sync(G_{sj})$. $G_{sj} = (Y_j, \Sigma, y_{0j}, \eta_j, Y_{mj})$. $j \in 1, \ldots, N_s$.

Entonces la síntesis e implementación de la superestructura se puede hacer mediante un predicado estructural de $N_p + N_s$ variables.

Teorema 3.4. *Dadas G_{pi} y G_{sj}, existe un predicado estructural P_{struc} tal que contenga todos los candidatos a controlador de procedimientos.*

La prueba es idéntica a la hecha para el teorema anterior, excepto que es para una descripción de MEFs con mayor cantidad de estados. Se pueden usar los mismos pasos para calcular P_{struc}.

De igual manera que para la síntesis de supervisores, para el cálculo de la superestructura del controlador de procedimientos se experimento con los algoritmos incrementales con las especificaciones por separado y con especificaciones sucesivas. Los mejores resultados se obtuvieron con la forma incremental con especificaciones por separado. Su descripción y resultados se muestran en los siguientes capítulos.

Capítulo 4

Algoritmos y técnicas implementadas para la síntesis de controladores en la herramienta SSPC versión 0.03

Una vez presentadas las bases teóricas en los capítulos anteriores, ahora en este capítulo se presentan los algoritmos implementados para el cálculo del controlador supervisor y de la superestructura, así como las funciones de Buddy para el manejo de BDDs que se usaron para el reordenamiento dinámico de las variables.

Además, se presentan otras modificaciones realizadas a la herramienta SSPC en cuanto al cálculo del controlador de procedimientos a partir de la superestructura maximal, en la escritura de los estados marcados en los archivos de salida y en la generación de autolazos para las especificaciones de manera interna.

4.1. Algoritmo usado para la síntesis del control supervisor

El procedimiento computacional utilizado para el cálculo del control supervisor es el mostrado en la prueba del teorema 2.3. Dicho algoritmo fue presentado por Zhang y Wonham en el 2001, y utilizado en el programa SmartTCT [30]. A continuación se presenta el algoritmo:

1. Se inicializa $P_{good} = 1$.

2. $\|P_{bad1}\| = \{(x,y) \models P_{bad1}\}$ ssi $((x,y) \not\models P_{good})$
   o $(\exists \sigma \in \Sigma_u)(\xi(x,\sigma)!$ y $\neg\eta(y,\sigma)!)$
   o $(\exists \sigma \in \Sigma_u)((\xi(x,\sigma)!, \eta(y,\sigma)!)$ y $(\xi(x,\sigma), \eta(y,\sigma)) \not\models P_{good})$

3. $\|P_{bad2}\| = \{(x,y) \models P_{bad2}\}$ ssi $(\exists w \in \Sigma_u^*)((\xi(x,w), \eta(y,w)) \models P_{bad1})$

4. $\|P_{al}\| = \{(x,y) \models P_{al}\}$ ssi
   $(\exists w \in \Sigma^*)$
   $(\xi(x_0,w) = x, \eta(y_0,w) = y)$ y $(\forall v < w)$ $((\xi(x_0,v), \eta(y_0,v)) \not\models (P_{bad1} \cup P_{bad2}))$

5. $\|P_{co}\| = \{(x,y) \models P_{co}\}$ ssi
   $(\exists w \in \Sigma^*)$
   $(\xi(x,w) \in X_m, \eta(y,w) \in Y_m)$ y $(\forall v < w)((\xi(x,v), \eta(y,v)) \not\models (P_{bad1} \cup P_{bad2}))$

6. $\|P_{newgood}\| = \|P_{al}\| \cap \|P_{co}\|$

7. Si $\|P_{newgood}\| \subset \|P_{good}\|$ se repiten los pasos del 2 al 7 con $\|P_{good}\| = \|P_{newgood}\|$, sino $\|P_{sup}\| = \|P_{newgood}\|$ y termina el algoritmo.

Los pasos 1, 2, 6 y 7 son fácilmente realizables mediante operaciones lógicas, tales como AND, OR, y la NEGACIÓN.

Los pasos 4 y 5 se realizan utilizando los algoritmos de alcanzabilidad y coalcanzabilidad previamente programadas en el SSPC versiones 1 y 2.

Finalmente, para realizar el paso 3 se usa el algoritmo de coalcanzabilidad, considerando en vez de los estados marcados a los estados que se encuentran en $\|P_{bad1}\|$, y tomando como alfabeto sólo a los eventos incontrolables.

4.2. Algoritmo usado para el cálculo de la superestructura maximal

El procedimiento computacional utilizado para el cálculo del control supervisor es el mostrado en la prueba del teorema 2.5. Dicho procedimiento se presenta a continuación:

1. Se inicializa $P_{good} = 1$.

2. $\|P_{bad}\| = \{(x,y) \models P_{bad}\}$ ssi $((x,y) \not\models P_{good})$
   o $(\exists \sigma \in \Sigma_u)(\xi(x,\sigma)!$ y $\neg\eta(y,\sigma)!)$
   o $(\exists \sigma \in \Sigma_u)((\xi(x,\sigma)!, \eta(y,\sigma)!)$ y $(\xi(x,\sigma), \eta(y,\sigma)) \not\models P_{good})$

3. $\|P_{bad-cont}\| = \{(x,y) \models P_{bad} | (((x,y) \models P_{good})$ y $(\exists \sigma \in \Sigma_c)(\xi(x,\sigma)!$ y $\eta(y,\sigma)!))\}$

4. $\|P_{erase1}\| = \|P_{bad}\| - \|P_{bad-cont}\|$

5. $\|P_{erase2}\| = \{(x,y) \models P_{erase2}\}$ ssi $(\exists w \in \Sigma_u^*)((\xi(x,w), \eta(y,w)) \models P_{erase1})$ y
   $(\nexists \overline{w}\sigma_c$ para cualquier $\sigma_c \in \Sigma_c)$ $((\xi(x,\overline{w}\sigma_c), \eta(y,\overline{w}\sigma_c)) \not\models P_{erase1})$

6. $\|P_{al}\| = \{(x,y) \models P_{al}\}$ ssi
   $(\exists w \in \Sigma^*)(\xi(x_0,w) = x, \eta(y_0,w) = y)$ y
   $(\forall v < w)$ $((\xi(x_0,v), \eta(y_0,v)) \not\models (P_{erase1} \cup P_{erase2}))$

7. $\|P_{co}\| = \{(x,y) \models P_{co}\}$ ssi
   $(\exists w \in \Sigma^*)(\xi(x,w) \in X_m, \eta(y,w) \in Y_m)$ y
   $(\forall v < w)((\xi(x,v), \eta(y,v)) \not\models (P_{erase1} \cup P_{erase2}))$

8. $\|P_{newgood}\| = \|P_{al}\| \cap \|P_{co}\|$

9. Si $\|P_{newgood}\| \subset \|P_{good}\|$ se repiten los pasos del 2 al 9 con $\|P_{good}\| = \|P_{newgood}\|$.
   Si $\|P_{newgood}\| = \|P_{good}\|$, entonces $\|P_{struc}\| = \|P_{newgood}\|$ y termina el algoritmo.

Los pasos 1, 2, 3, 4, 8 y 9 se realizan mediante operaciones lógicas, tales como AND, OR, y la NEGACIÓN.

Y los pasos 6 y 7 se realizan con los algoritmos de alcanzabilidad y coalcanzabilidad programados anteriormente en el SSPC versiones 1 y 2. Y el paso 5, se realiza usando el algoritmo de coalcanzabilidad, considerando en vez de los estados marcados a los estados que se encuentran en $\|P_{erase1}\|$, tomando como alfabeto sólo a los eventos incontrolables, y teniendo en cuenta que no exista ningún evento controlable que pueda evitar llegar a estados de $\|P_{erase1}\|$.

Ahora, definiendo la relación de transición de la planta sincronizada con la especificación como sigue:

$$\| T_{ps} \| := \{((x, y), \sigma, (x', y')) : (x, \sigma, x') \in \| T_p \| \text{ y } (y, \sigma, y') \in \| T_s \|\}$$

Tenemos que la relación de transición de la superestructura $\|T_{cs}\|$ se obtiene a partir $\|T_{ps}\|$ y de los estados de $\|P_{struc}\|$ de la siguiente manera:

$\|T_{cs}\| := \{(q, \sigma, q') \in \|T_{ps}\| \,|(q, q' \in \|P_{struc}\|)$ y $((\forall(q, \sigma_u, q') \in \|T_{ps}\|), \nexists((q, \sigma_{u2}, q_2) \in \|T_{ps}\|)$ donde $(\sigma_u, \sigma_{u2} \in \Sigma_u)$ y $(q_2 \notin \|P_{struc}\|))\}$.

Es decir, se eliminan todos los eventos incontrolables de los estados en los que exista al menos un evento incontrolable que lleve a estados que no pertenezcan a la superestructura.

Para mostrar más claramente porque la relación de transición se obtiene así, se presenta el ejemplo de la figura 4.1. Los eventos incontrolables se presentan con líneas punteadas.

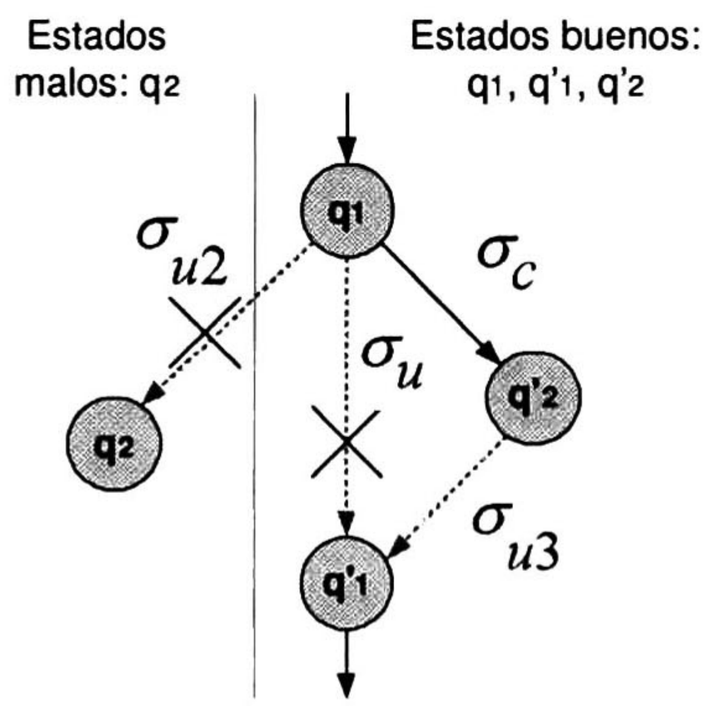

Figura 4.1: Ejemplo de la obtención de la relación de transición para la superestructura

Los estados q_1, q_1' y q_2' son parte de la superestructura mientras que el estado q_2 no forma parte de esta, es decir, $(q_1, q_1', q_2' \in \|P_{struc}\|)$ y $(q_2 \notin \|P_{struc}\|)$.

Con los estados $(q_1, q_1'$ y $q_2')$ se pueden formar las tripletas $(q_1, \sigma_u, q_1'), (q_1, \sigma_c, q_2')$ y $(q_2', \sigma_{u3}, q_1')$. De acuerdo con la TCP ya que el evento incontrolable formado por la tripleta (q_1, σ_{u2}, q_2) no forma parte de la superestructura, entonces todos los demás

eventos incontrolables que parten del mismo estados deben ser eliminados, en este caso la tripleta (q_1, σ_u, q_1'). Sin embargo, el estado q_1 sigue perteneciendo a la superestructura ya que existe un evento controlable σ_c que se puede anticipar a estos eventos incontrolables. Así la superestructura quedaría formada por los eventos mostrados en la parte derecha de la figura 4.1, exeptuando a la tripleta (q_1, σ_u, q_1').

4.3. Algoritmo sifting para reordenamiento dinámico de variables en BDDs

Como ya se mencionó anteriormente, Rudell introdujo en el 93 la heurística llamada sifting [21] para el reordenamiento de variables en BDDs. Este algoritmo tiene como idea principal encontrar la mejor posición para cada variable dentro del BDD. Las variables individuales se mueven a todos los niveles del BDD una por una. Este algoritmo está basado en una heurística llamada permutación de ventana (sliding window o window permutation), la cual es un tipo de reordenamiento de variables más sencillo y a la vez más utilizado que consiste en ir cambiando de posición dos variables que se encuentren en niveles adyacentes (swapping). A continuación se explicará el algoritmo sifting más a detalle.

El algoritmo sifting está basado en encontrar la posición óptima para una variable en el BDD, asumiendo que todas las otras variables permanecen fijas. Si hay n variables en el BDD, entonces hay n posiciones potenciales para una variable, incluyendo su posición actual. De entre estas n posiciones, el objetivo de emplear el algoritmo sifting es encontrar el orden que minimiza el tamaño del BDD.

La posición óptima para una variable es determinada por fuerza bruta como sigue. La variable se intercambia con su variable sucesora hasta que la variable llegue al lado de la última variable en el BDD; es decir, la variable es desplazada hacia al fondo del BDD. Entonces esta misma variable se intercambia con su predecesora hasta que llegue al principio del BDD; es decir, la variable se desplaza a la cima del BDD. El mejor tamaño del BDD obtenido durante esta búsqueda se recuerda y la posición de la variable se restaura moviendo la variable de la posición de la cima hacia su óptima

$x_1, x_2, x_3, x_4, x_5, x_6, x_7$	inicial
$x_1, x_2, x_3, x_5, x_4, x_6, x_7$	swap (x_4, x_5)
$x_1, x_2, x_3, x_5, x_6, x_4, x_7$	swap (x_4, x_6)
$x_1, x_2, x_3, x_5, x_6, x_7, x_4$	swap (x_4, x_7)
$x_1, x_2, x_3, x_5, x_6, x_4, x_7$	swap (x_7, x_4)
$x_1, x_2, x_3, x_5, x_4, x_6, x_7$	swap (x_6, x_4)
$x_1, x_2, x_3, x_4, x_5, x_6, x_7$	swap (x_5, x_4)
$x_1, x_2, x_4, x_3, x_5, x_6, x_7$	swap (x_3, x_4)
$x_1, x_4, x_2, x_3, x_5, x_6, x_7$	swap (x_2, x_4)
$x_4, x_1, x_2, x_3, x_5, x_6, x_7$	swap (x_1, x_4)
$x_1, x_4, x_2, x_3, x_5, x_6, x_7$	swap (x_4, x_1)
$x_1, x_2, x_4, x_3, x_5, x_6, x_7$	swap (x_4, x_2)
$x_1, x_2, x_3, x_4, x_5, x_6, x_7$	swap (x_4, x_3)
$x_1, x_2, x_3, x_3, x_4, x_6, x_7$	swap (x_4, x_5)
$x_1, x_2, x_3, x_3, x_6, x_4, x_7$	swap (x_4, x_6)
$x_1, x_2, x_3, x_3, x_6, x_7, x_4$	swap (x_4, x_7)

Tabla 4.1: Ejemplo del algoritmo sifting

posición.

La tabla 4.1 muestra las permutaciones de las variables, las cuales se obtienen al aplicar el algoritmo sifting a la variable x_4. Las 7 posiciones para la variable x_4 se exploran usando 9 intercambios adyacentes, y en la parte inferior de la tabla se muestra como la posición óptima se restaura con 6 intercambios adicionales, el cual sería el peor de los casos.

El algoritmo sifting procede como sigue. Las variables se ordenan en forma decreciente basándose en el número de nodos de cada nivel del BDD. Entonces, en ese orden, cada variable se mueve a su posición óptima asumiendo que todas las otras variables permanecen fijas. Cada variable es movida sólo una vez en este proceso, aunque el algoritmo podría ser iterativo hasta converger a un orden.

El algoritmo sifting tiene la ventaja de que una variable puede moverse una distancia larga en el ordenamiento. Nótese que el tamaño del BDD puede aumentar significativamente después de los primeros intercambios de algunas variables, y eventualmente reducirse debajo del punto de partida. Esto permite la aceptación de la secuencia completa de intercambios de variables adyacentes necesarios, basándose en

que la mejor posición se encontrara a pesar de que en cualquier momento se tendrá un aumento en el tamaño del BDD. Una limitación de la heurística permutación de ventana es que para mover una variable una distancia larga se requieren varios movimientos y éstos pueden ser bloqueados por un movimiento intermedio que incrementa el número de nodos.

El algoritmo sifting requiere $O(n^2)$ intercambios de niveles adyacentes en el BDD, y cada uno de éstos tiene una complejidad proporcional al tamaño del BDD. Para controlar esta complejidad, en el peor de los casos, la búsqueda en una dirección particular se termina si el tamaño del BDD aumenta a dos veces su tamaño original.

4.3.1. Funciones de la librería Buddy utilizadas para el reordenamiento dinámico en SSPC

Las funciones de la librería Buddy usadas para el reordenamiento dinámico de las variables en los BDDs en la herramienta SSPC son las siguientes:

```
int bdd_addvarblock(BDD var, int fixed)
```

Esta función crea un nuevo bloque de variables en el conjunto de variables `var`. Las variables de `var` deben ser contiguas.

Los bloques de variables se ordenan como un árbol, con los rangos más largos arriba y los más cortos abajo. A continuación se describe un ejemplo el cual se muestra en la figura 4.2: asumir que el bloque de las variables del 0 al 9 se agregan como primer bloque y después el bloque de variables del 0 al 6. Esto nos da al bloque 0-9 hasta arriba, con el bloque 0-6 como hijo. Si ahora se agrega el bloque de variables del 2 al 4, sería un hijo del bloque 0-6. Un bloque de variables del 0 al 8 sería hijo del bloque 0-9 y tendría al bloque 0-6 como hijo. Bloques parcialmente sobrepuestos no son permitidos.

El parámetro `fixed` establece al bloque como fijo (sin reorden permitido para los bloques hijos) o libre (con reorden permitido para los bloques hijos), usando las constantes `BDD_REORDER_FIXED` y `BDD_REORDER_FREE`. El reordenamiento siempre se hace primero en los bloques de más arriba y después recursivamente hacia abajo.

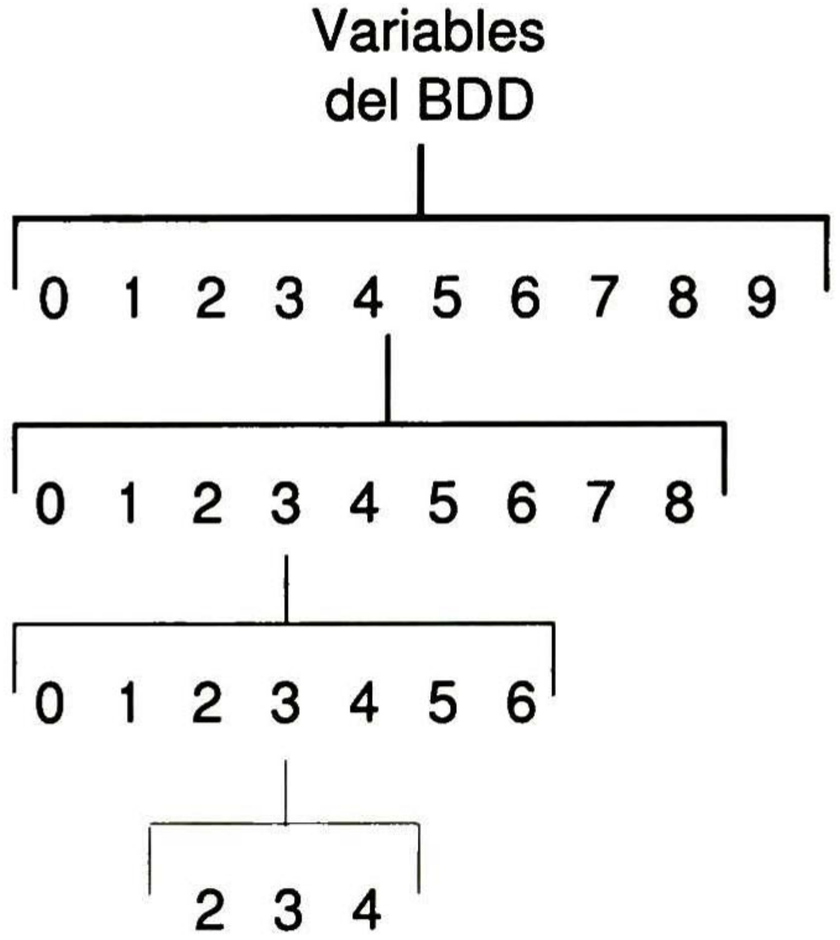

Figura 4.2: Ejemplo de la creación de bloques de variables de BDDs usando la función `bdd_addvarblock` de Buddy

El valor de salida es un entero que puede ser usado para identificar el bloque.

En este trabajo se creo un bloque con cada una de las variables utilizadas, de esta manera se puede encontrar la posición óptima en el BDD para cada variable.

```
void bdd_clrvarblock(void)
```

Elimina todos los bloques de variables que han sido definidos por llamados a la función `bdd_addvarblock`.

```
void bdd_reorder(int method)
```

Esta función inicia el reordenamiento dinámico usando la heurística definida por `method`, la cual puede ser una de las siguientes:

BDD_REORDER_WIN2 : Reordenamiento usando permutación de ventana de tamaño 2. Este algoritmo intercambia dos bloques de variables adyacentes, y si este resultado tiene más nodos entonces los dos bloques se intercambian de regreso a su posición original. De otra forma el resultado se mantiene en el orden de variables. Esto se repite para todos los bloques de variables.

BDD_REORDER_WIN2ITE : Lo mismo que el anterior, pero el proceso se repite hasta que no haya progreso en el resultado. Usualmente es un método rápido y eficiente.

BDD_REORDER_WIN3 : Lo mismo que el *BDD_REORDER_WIN2*, pero con una ventana de tamaño 3.

BDD_REORDER_WIN3ITE : Lo mismo que el *BDD_REORDER_WIN2ITE*, pero con una ventana de tamaño 3.

BDD_REORDER_SIFT : Reordenamiento sifting. Cada bloque es movido a todas las posibles posiciones. Potencialmente un método muy bueno pero muy lento. Esta opción es la que se uso para llevar a cabo el reordenamiento dinámico en el SSPC versión 3.

BDD_REORDER_SIFTITE : Lo mismo que el anterior pero el proceso se repite hasta que no haya ninguna mejora. Puede ser extremadamente lento.

BDD_REORDER_RANDOM : Selecciona una posición aleatoria para cada variable.

Como ya se ha mencionado en las secciones anteriores, el tipo de reordenamiento utilizado en este trabajo de tesis es el sifting. Este algoritmo se usa debido a las ventajas anteriormente descritas, a diferencia de las limitaciones ya mencionadas del algoritmo de permutación de ventana.

Además, el orden inicial al momento de la creación de los BDDs en el SSPC es el mismo que se utilizó en la versión 2. A partir de este orden, se realiza el reordenamiento dentro del algoritmo incremental, el cual se explica a continuación.

4.4. Algoritmo para el cálculo incremental

Como se había mencionado en el capítulo 3, para el cálculo del control supervisor y de la superestructura del control de procedimientos se utiliza un algoritmo incremental con las especificaciones por separadas con el cual se obtuvieron buenos resultados.

El algoritmo de cálculo incremental se realiza a partir del conjunto de especificaciones establecidas por el usuario. Este algoritmo se caracteriza por la forma incremental en que se va construyendo el controlador. El algoritmo usado trabaja con la representación del modelo completo de la planta, además del estado inicial, estados marcados y especificaciones.

Básicamente el algoritmo se compone de una función recursiva. El proceso incremental se realiza especificación por especificación de forma separada. La intersección de los estados alcanzables, coalcanzables y controlables que se obtengan del algoritmo de síntesis realizado para cada especificación formarán parte del supervisor o de la superestructura según sea el caso.

Como se comentó en el capítulo 3, debido a que la intersección de MEFs aseadas no garantiza que el resultado sea una MEF aseada, se revisa que después de llevar a cabo la intersección todos los estados son alcanzables y coalcanzables para asegurar que tenemos una MEF aseada. De no ser así se vuelve a aplicar el algoritmo de síntesis con la especificación completa, es decir, del producto síncrono de todas las especificaciones, partiendo de los estados obtenidos en la intersección.

4.4.1. Cálculo de control supervisor

Mediante la implementación conjunta del algoritmo de síntesis del control supervisor, el algoritmo incremental con especificaciones por separado y el algoritmo de reordenamiento dinámico de variables para BDDs, se realiza el cálculo del control supervisor. Primeramente se hace un reordenamiento de variables. Luego se aplica el algoritmo de cálculo incremental especificación por especificación de forma separada. Dentro del algoritmo incremental se utiliza el algoritmo para la síntesis del control supervisor usando la representación del modelo completo de la planta y de la especificación en turno. También, dentro del algoritmo incremental, se lleva a cabo un reordenamiento dinámico antes de terminar cada ciclo del algoritmo incremental. Luego, se realiza una intersección de todos los estados buenos obtenidos en cada ciclo del algoritmo incremental para formar el supervisor. Finalmente, se revisa que todos los estados del supervisor sean alcanzables y coalcanzables. De no ser así se realiza la

síntesis del supervisor tomando la especificación completa.

El cálculo del control supervisor se describe más claramente a continuación:

```
n = número de especificaciones;
reordenamiento de variables;
repetir desde j=0 hasta j<n (algoritmo incremental)
{
   síntesis para el control supervisor (entre la planta y la especificación j);
   reordenamiento de variables;
}
intersección de todos los estados buenos;
revisar si se tiene un MEF aseada
   * en caso de si ser aseada
       se tiene el supervisor;
   * en caso de no ser aseada
       síntesis del supervisor (entre la planta y la especificación completa);
```

4.4.2. Cálculo de la superestructura maximal

De manera similar al cálculo del control supervisor, para el cálculo de la superestructura del controlador de procedimientos se conjuntan el algoritmo incremental con especificaciones por separado, el reordenamiento dinámico de variables para BDDs y, en este caso, el algoritmo para el cálculo de la superestructura.

De igual forma, al principio se hace un reordenamiento de variables. Luego se aplica el algoritmo de cálculo incremental de forma recursiva especificación por especificación de forma separada. Dentro del algoritmo incremental se calcula la superestructura del modelo completo de la planta y de la especificación en turno. Y también dentro del algoritmo incremental se lleva a cabo un reordenamiento dinámico antes de terminar cada ciclo del algoritmo incremental.

Después de esto se realiza la intersección de todas las superestructuras parciales para obtener una sola superestructura final de todo el sistema. Luego se revisa el aseo de dicha superestructura final. De no ser aseada se realiza el cálculo de la superestructura tomando la especificación completa.

El cálculo de la superestructura para el controlador de procedimientos se describe más claramente a continuación:

```
n = número de especificaciones;
reordenamiento de variables;
repetir desde j=0 hasta j<n (algoritmo incremental)
{
   cálculo de la superestructura para la especificación j;
   reordenamiento de variables;
}
obtención de la superestructura final a partir de la intersección
de las superestructuras parciales;
revisar si la superestructura final es aseada
   * en caso de si ser aseada
       se tiene la superestructura;
   * en caso de no ser aseada
       cálculo de la superestructura (entre la planta y la especificación completa)
```

4.5. Síntesis del controlador de procedimientos a partir de la superestructura maximal

El controlador de procedimientos se calcula a partir de la superestructura de controladores. El algoritmo usado para el cálculo del controlador de procedimientos es prácticamente el mismo implementado por Reza en el SSPC versión 1 [20]. La única modificación que se le realizó fue la aplicación de un reordenamiento dinámico de variables de BDDs al principio del algoritmo, para así trabajar con estructuras más pequeñas.

A continuación se explica brevemente el algoritmo implementado por Reza:

Primero se determinan los estados en los que existan ambos tipos de transiciones (controlables e incontrolables). Se lleva a cabo un análisis de coalcanzabilidad para las transiciones incontrolables de dichos estados; en caso de que todas las transiciones incontrolables de un estado dado lleguen a algún estado marcado, se eliminan todas las transiciones controlables de ese estado; en caso contrario se eliminan todas las incontrolables.

Una vez analizados todos los estados en los que existen ambos tipos de transiciones, se procede a analizar los estados en los que existe más de una transición controlable. Se

analiza la coalcanzabilidad de estas transiciones y se elige sólo una de ellas, cualquiera que lleve al controlador a algún estado marcado.

Cuando se terminan de analizar todos los estados en los que existe más de una transición controlable se procede a asear el controlador, por medio de un análisis de alcanzabilidad y eliminando los estados que no son alcanzables.

Todos estos cálculos se realizan por medio de operaciones lógicas entre los BDD's almacenados en memoria.

4.6. Otras mejoras realizadas al software SSPC versión 3

4.6.1. Etiquetación completa de los estados marcados en los archivos de salida

Para especificar en los archivos de salida cuales son los estados iniciales y los marcados, se hace una lectura previa de los BDDs en donde se guarda la información de los estados iniciales y marcados.

Anteriormente sólo se guardaba el valor de un estado inicial y un estado marcado. Por lo que, cuando existía más de un estado marcado, no todos eran identificados como tales en los archivos de salida.

Para lograr guardar la información de todos los estados marcados, se creó un arreglo de variables. Una vez que se tiene la información de los estados iniciales y los marcados, al momento de imprimir el nombre de un estado se compara el nombre de este estado con los nombres de los estados guardados anteriormente, y en caso de ser iguales a alguno de ellos se coloca en el archivo de salida la leyenda INITIAL o MARKED, respetando la sintaxis establecida para los archivos de entrada del programa.

4.6.2. Creación de autolazos en las MEFs de las especificaciones de manera interna

Como es de esperarse, para sistemas grandes modelados mediante MEFs se tienen una gran cantidad de transiciones. Ahora bien, en el modelado de especificaciones en ocasiones no se encuentran presentes todas las transiciones del alfabeto en alguna especificación dada. Sin embargo, para la correcta sincronización de las MEFs, todas aquellas transiciones existentes en el alfabeto Σ que no se utilizan para describir el comportamiento deseado deben de estar presentes como autolazos en cada especificación.

Anteriormente esto se realizaba manualmente en la escritura del archivo de entrada de cada especificación. Esto podría llegar a ser un proceso muy tardado y tedioso e incluso se podrían llegar a tener errores al momento de estar capturando la información, sobre todo para sistemas con una gran cantidad de transiciones.

Por ello es que para la versión 3 del SSPC la generación de estos autolazos se realiza de manera interna por el programa y de forma transparente para el usuario. De esta forma se facilita la escritura de los archivos de entrada correspondientes a las especificaciones, sin modificar los archivos de salida de las especificaciones ni ningún otro resultado u operación.

Capítulo 5

Desempeño del SSPC versión 3

En este capítulo se muestra el desempeño de la herramienta SSPC versión 3 para el cálculo del control supervisor, la superestructura maximal y del controlador de procedimientos. Además, se compara con la versión 2 y en el caso del control supervisor también se compara con la herramienta STCT [30].

La comparación del cálculo del control supervisor se hace contra STCT debido a que es la herramienta que ha reportado haber obtenido mejores resultados en cuanto al cálculo de estados alcanzables y coalcanzables con los cuales se puede generar un supervisor. Esta herramienta reporta haber llegado a obtener un conjunto de estados de $2{,}59 \times 10^{23}$, con los cuales se puede formar un supervisor.

Las pruebas se realizaron en una computadora Compaq EVO modelo D510 con procesador Pentium IV a 2.4 Ghz con 1 Gb de memoria RAM. STCT se ejecutó en Windows 2000, mientras que SSPC se ejecutó en Mandrake Linux 9.0.

5.1. Funciones del SSPC

En esta sección se describen las opciones del SSPC utilizadas para los cálculos del control supervisor, la superestructura maximal y el controlador de procedimientos.

El menú principal del SSPC versión 3 se muestra en la figura 5.1. Las opciones que se agregaron a esta nueva versión son: (F) e (I), las cuales sirven para hacer los cálculos incrementales del control supervisor y de la superestructura, respectivamente.

Anteriormente, para calcular un control supervisor, se utilizaba la opción (A)

```
  Software for the Synthesis of Procedural Controllers (SSPC).
  Version 0.3
-------------------------------------------------------------------

      (A) Read FSM system and specification files
      (B) Add causal behavior to system model

      (C) Build the synchronous product from system & specification
      (D) Delete forbidden states from system & specification

      (E) Obtain the superstructure controller (full synchronous approach)
      (F) Obtain the superstructure controller (incremental approach)
      (G) Synthesize a procedural controller for system & specification

      (H) Synthesize a supervisory controller (full synchronous approach)
      (I) Synthesize a supervisory controller (incremental approach)

      (Q) Exit the program

  Choose an option:
```

Figura 5.1: Menú principal del SSPC

para introducir los archivos con los componentes de la planta y las especificaciones, realizándose a su vez el producto asíncrono para la planta y la sincronización de las especificaciones. Luego se obtenía la sincronización de la planta y la especificación con la opción (C). Y, finalmente, se calculaba el control supervisor a partir del producto síncrono entre la planta y la especificación con la opción (H).

Ahora, en la versión 3 del SSPC, se usa la opción (A); las opciones (B) y (D) si se requieren, y posteriormente la opción (I) para el cálculo incremental del control supervisor.

Para calcular un control de procedimientos se utilizaba, de igual forma que lo anteriormente explicado, la opción (A). La opción (B) para introducir el comportamiento causal, si es requerido. La opción (C) para realizar la sincronización entre la planta y la especificación. La opción (D) para eliminar los estados prohibidos, si se requiere. Luego la opción (E) para encontrar la superestructura. Y, finalmente, la opción (G) para obtener el controlador de procedimientos. Ahora en la versión 3, para calcular el control de procedimientos, tampoco se requiere el uso de la opción (C), y en vez de usar la opción (E) se utiliza la opción (F) para calcular la superestructura de forma incremental. Y, para el cálculo del controlador de procedimientos, se sigue utilizando la opción (G).

La información más detallada del uso de la herramienta SSPC se encuentra en el manual de usuario para SSPC [7], donde también se encuentra información sobre el formato de los archivos de entrada y salida.

5.2. Control supervisor

En esta sección se presentan los resultados obtenidos y las comparaciones hechas para el cálculo del control supervisor.

5.2.1. Línea de transferencia

El ejemplo utilizado para realizar las pruebas de desempeño del SSPC para el cálculo del control supervisor es el de una línea de transferencia la cual fue tomada del trabajo de Zhang [30], y se muestra en la figura 5.2.

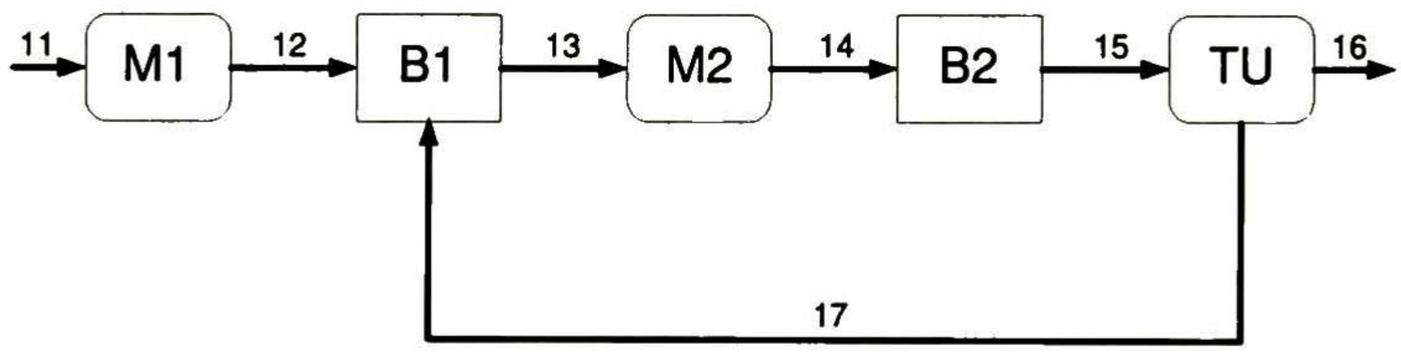

Figura 5.2: Línea de transferencia

La línea de transferencia consiste en dos máquinas M1 y M2, seguidas de una unidad de prueba llamada TU, unidos por los buffers B1 y B2. Una pieza probada por TU puede ser aceptada o rechazada. Si es aceptada, sale del sistema; si es rechazada, regresa a B1. Los eventos se definen como se muestra en la tabla 5.1. Cada vez que una pieza entra a una máquina o a la unidad de prueba es un evento controlable. Y cuando una pieza sale de una máquina o de la unidad de prueba es un evento incontrolable. Inicialmente, las máquinas no están trabajando ninguna pieza, la unidad de prueba está libre y los buffers vacíos. Se considera que el sistema concluye satisfactoriamente su labor cada vez que regresa a su estado inicial. Para este ejemplo se considera que la capacidad de los buffers es de una pieza.

Evento	Controlable	Definición
11	si	M1 mete una pieza al sistema
12	no	M1 manda una pieza a B1
13	si	M2 toma una pieza de B1
14	no	M2 manda una pieza a B2
15	si	TU toma una pieza de B2
16	no	TU saca una pieza del sistema
17	no	TU regresa una pieza a B1

Tabla 5.1: Definición de los eventos de la línea de transferencia

Los modelos de las MEFs de los componentes elementales de este sistema se muestran en la figura 5.3. A partir de ellos se construye el modelo completo de la planta mediante un producto asíncrono; este modelo se muestra en la figura 5.4. En la figura 5.5 se encuentra el modelo de la especificación en la cual B1 y B2 deben proteger de un sobreflujo, es decir, si entra una pieza a B1 con 12 ó 17, no puede entrar otra hasta que la pieza allá salido con 13 y, si entra una pieza a B2 con 14, no puede entrar otra hasta que la pieza allá salido con 15. Las líneas segmentadas indican transiciones incontrolables.

El resultado del control supervisor para una línea de transferencia se muestra en la figura 5.6.

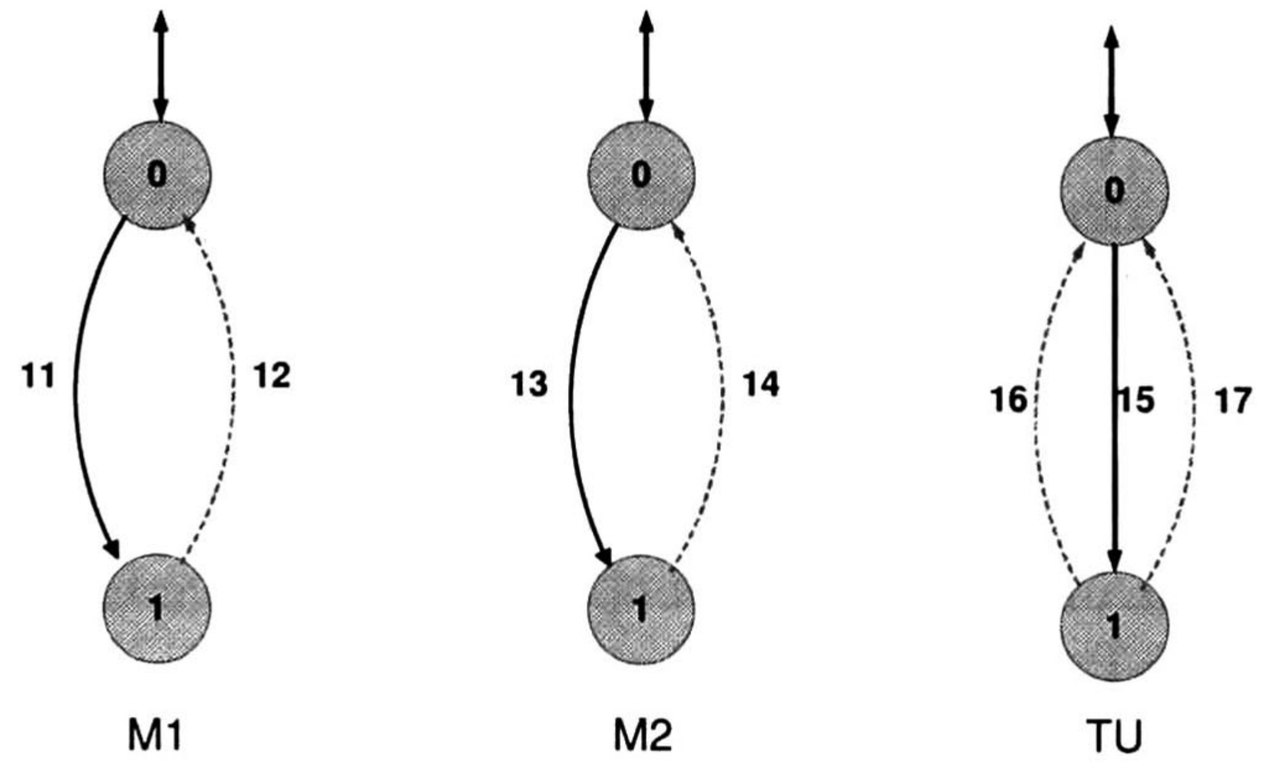

Figura 5.3: Componentes elementales de la planta para la línea de transferencia

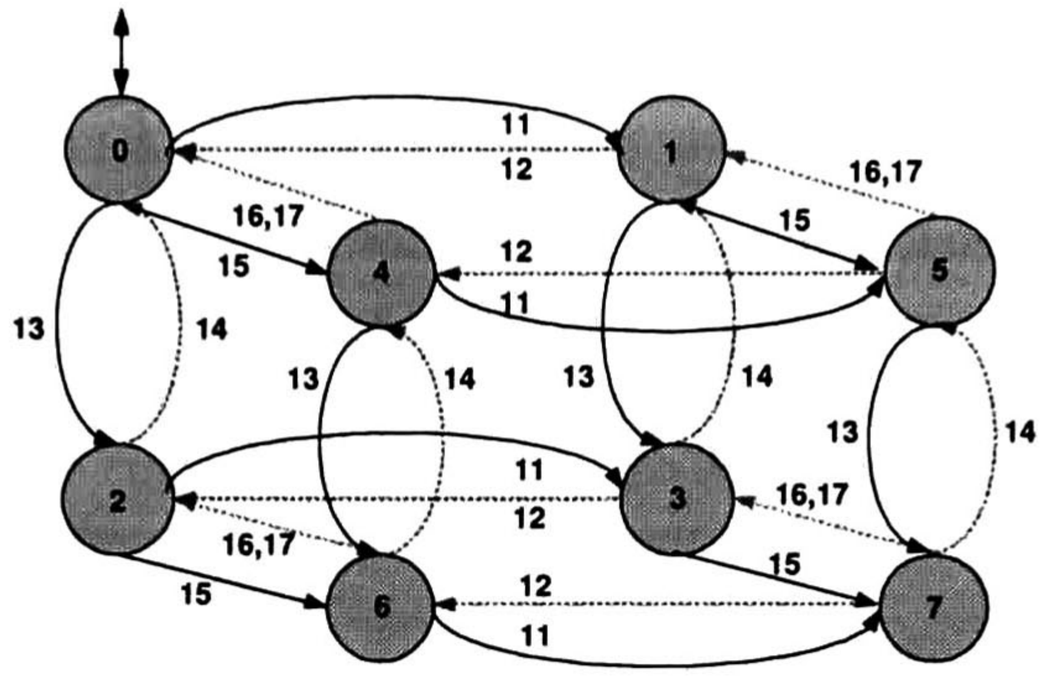

Figura 5.4: Modelo completo de la planta para la línea de transferencia

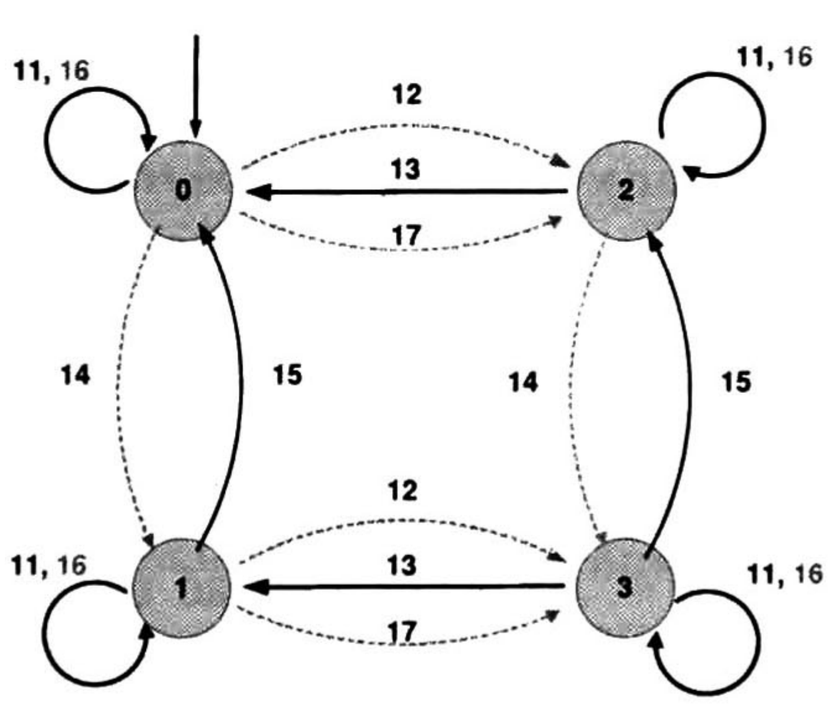

Figura 5.5: Especificación para la línea de transferencia

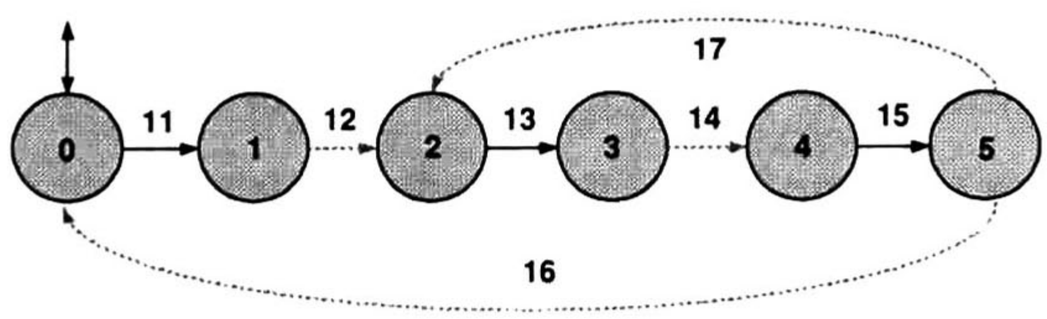

Figura 5.6: Control supervisor para la línea de transferencia

5.2.2. Crecimiento del sistema

La forma de crecer el sistema es agregando nuevas líneas de transferencia que se ejecutan de forma independiente de las ya existentes. Se agrega una línea a la vez y se pide como especificación que en cada línea los respectivos buffers no tengan sobreflujo. Aunque se puede implementar un controlador independiente para cada una de estas líneas de transferencia; en este caso se busca encontrar un controlador global para todas las líneas.

5.2.3. Resultados obtenidos

En las tablas 5.2, 5.3 y 5.4 se muestran los resultados obtenidos para el cálculo del control supervisor con SSPC v.2, STCT y SSPC v.3, respectivamente. Para el caso de la herramienta SSPC v.3 (desarrollada durante este trabajo), se logró obtener un controlador para un sistema de hasta 10^{27} estados potenciales.

La herramienta STCT, una vez que tiene los resultados del control supervisor arroja el orden óptimo encontrado de los componentes elementales mediante el cual se tiene una menor cantidad de nodos usados, de uso de memoria y de tiempo de cálculo. Los resultados que pueden llegar a obtenerse usando un orden aleatorio de los componentes elementales del sistema y las especificaciones para este ejemplo pueden llegar a ser muy lentos y ocupar mucho espacio de memoria, incluso no llegan a resolver este ejemplo para estados potenciales de 10^7 [7]; los resultados que se registraron con la herramienta STCT fueron tomando en cuenta el orden óptimo de los componentes elementales. Para ello se realizaba el ejercicio dos veces: una para obtener el orden óptimo y otra para registrar los resultados de tiempo de cálculo, uso de memoria y cantidad de nodos usados. Sin embargo, para la obtención de los órdenes óptimos de ejemplos muy grandes, se usaba primero el orden óptimo que se tenía del ejemplo anterior en cuanto al tamaño de los estados potenciales, para así partir de un orden subóptimo y lograr realizar el cálculo del supervisor para después obtener el orden óptimo correspondiente a los componentes elementales y repetir nuevamente el problema para obtener el tiempo de cálculo, uso de memoria y los nodos usados con el orden óptimo.

♯ Bloques	$\|Q_P\|$	$\|Q_E\|$	$\|Q_{SUP}\|$	Tiempo (s)	Memoria (Mb)	Nodos
1	8	4	6	0.01	13.30	62
2	64	16	36	0.02	13.80	372
3	512	64	216	0.18	14.30	1,262
4	4,096	256	1,296	1.62	14.80	3,480
5	32,768	1,024	7,776	12.94	21.50	8,719
6	262,144	4,096	46,656	117.29	55.00	20,690
7	2,097,152	16,384	279,936	1091.32	171.50	47,543
8	16,777,216	65,536	1,679,616	14007.00	702.50	106,968

Tabla 5.2: Resultados de tiempo de cómputo, uso de memoria y nodos existentes para el cálculo del control supervisor usando SSPC versión 2

♯ Bloques	$\|Q_P\|$	$\|Q_E\|$	$\|Q_{SUP}\|$	Caso óptimo		Nodos
				Tiempo(s)	Memoria (Mb)	
1	8	4	6	<1	<1	6
2	64	16	36	<1	<1	13
3	512	64	216	<1	1.2	20
4	4,096	256	1,296	1	1.7	27
5	32,768	1,024	7,776	2	2.3	34
6	262,144	4,096	46,656	6	2.9	41
7	2.09×10^6	16,384	279,936	12	3.8	48
8	1.67×10^7	65,536	1.67×10^6	22	4.5	55
9	1.34×10^8	262,144	1.01×10^7	53	5.6	62
10	1.07×10^9	1.04×10^6	6.04×10^7	94	6.7	69
15	3.51×10^{13}	1.07×10^9	4.70×10^{11}	1,520	14.6	112
20	1.15×10^{18}	1.09×10^{12}	3.65×10^{15}	10,232	31.0	170
25	3.77×10^{22}	1.12×10^{15}	2.84×10^{19}	45,688	36.8	208
30	1.23×10^{27}	1.15×10^{18}	2.21×10^{23}	193,389	53.3	285

Tabla 5.3: Resultados de tiempo de cómputo, uso de memoria y nodos existentes para el cálculo del control supervisor usando STCT

♯ Bloques	$\|Q_P\|$	$\|Q_E\|$	$\|Q_{SUP}\|$	Tiempo(s)	Memoria (Mb)	Nodos
1	8	4	6	0.23	13.47	55
2	64	16	36	0.59	13.93	290
3	512	64	216	1.39	14.40	1,032
4	4,096	256	1,296	2.65	14.84	3,010
5	32,768	1,024	7,776	4.83	15.35	7,741
6	262,144	4,096	46,656	8.09	15.84	18,836
7	2.09×10^6	16,384	279,936	13.15	16.36	44,602
8	1.67×10^7	65,536	1.67×10^6	20.40	16.91	102,010
9	1.34×10^8	262,144	1.01×10^7	33.47	21.33	231,967
10	1.07×10^9	1.04×10^6	6.04×10^7	52.03	30.99	396,321
15	3.51×10^{13}	1.07×10^9	4.70×10^{11}	424.40	64.63	1,326,426
20	1.15×10^{18}	1.09×10^{12}	3.65×10^{15}	5,048.00	138.96	2,529,240
25	3.77×10^{22}	1.12×10^{15}	2.84×10^{19}	16,699.00	156.21	3,863,796
30	1.23×10^{27}	1.15×10^{18}	2.21×10^{23}	270,920.00	230.53	6,135,484

Tabla 5.4: Resultados de tiempo de cómputo, uso de memoria y nodos existentes para el cálculo del control supervisor usando SSPC versión 3

En las figuras 5.7, 5.8 y 5.9 se muestran las comparaciones gráficas entre los resultados obtenidos entre el SSPC v.2, STCT y SSPC v.3 para el cálculo del control supervisor en cuanto al tiempo de cálculo, uso de memoria y cantidad de nodos utilizados. Como se puede observar, la herramienta desarrollada durante este trabajo (SSPC v.3) supera ampliamente a su predecesora (SSPC v.2). Se redujo la cantidad de nodos usados y la cantidad de memoria utilizada para el cálculo del supervisor, principalmente para los ejemplos de gran cantidad de estados, además de conseguir llevar a cabo los cálculos del control supervisor en tiempos menores a los requeridos por SSPC v.2 y competir en el tiempo de cálculo y uso de memoria con el desempeño mostrado por STCT.

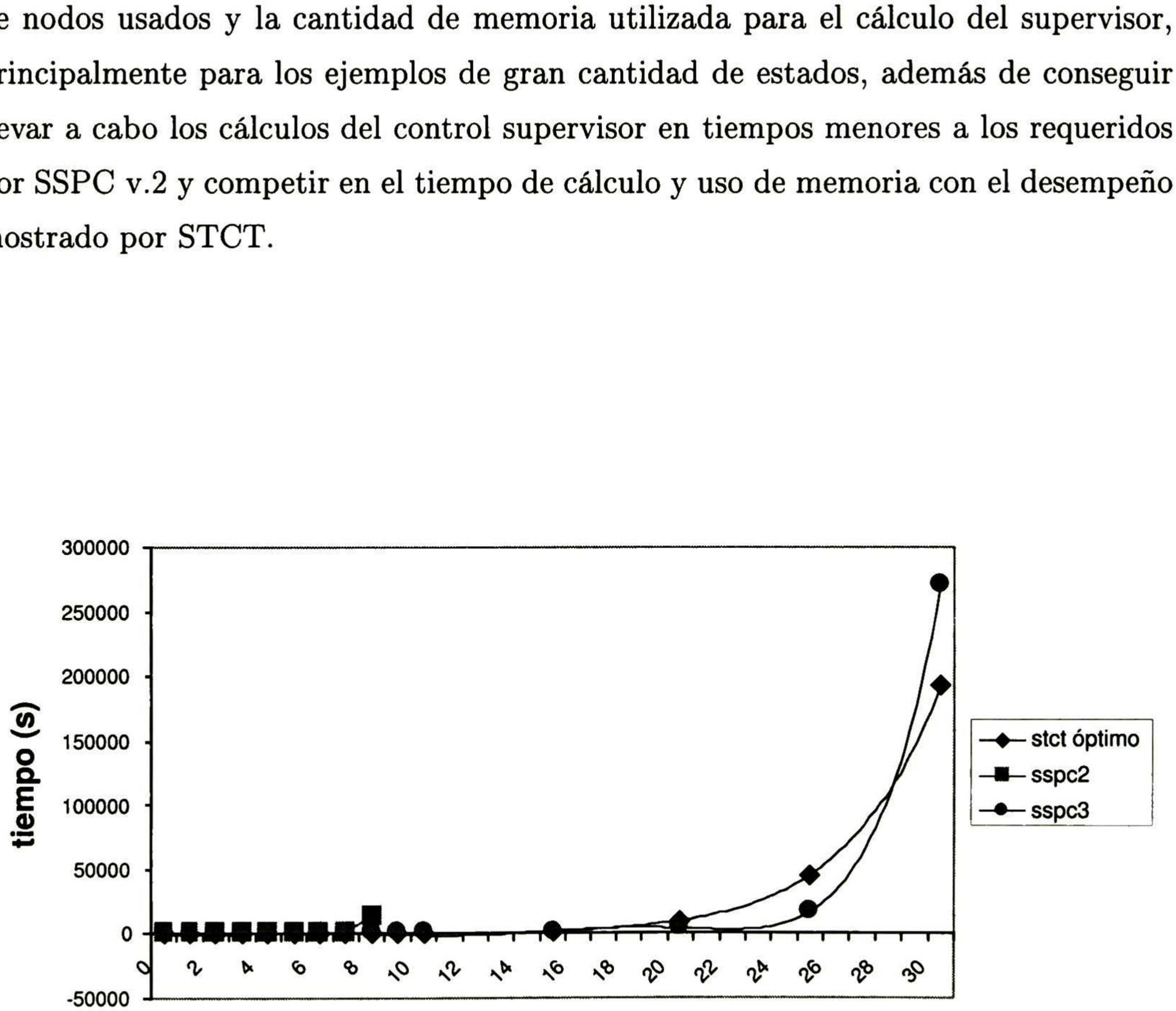

Figura 5.7: Comparación del tiempo de cálculo del control supervisor entre STCT, SSPC v.2 y SSPC v.3

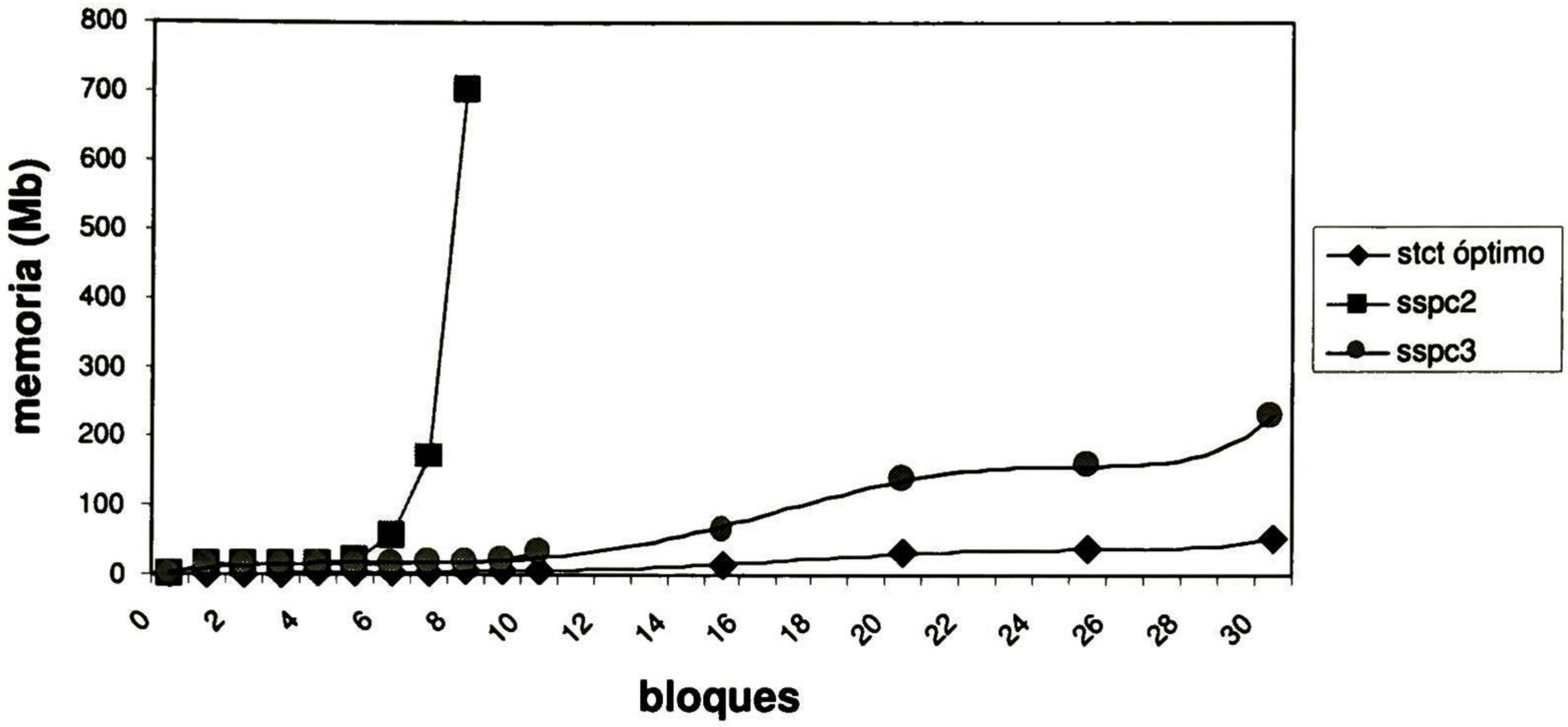

Figura 5.8: Comparación del uso de memoria para el cálculo del control supervisor entre STCT, SSPC v.2 y SSPC v.3

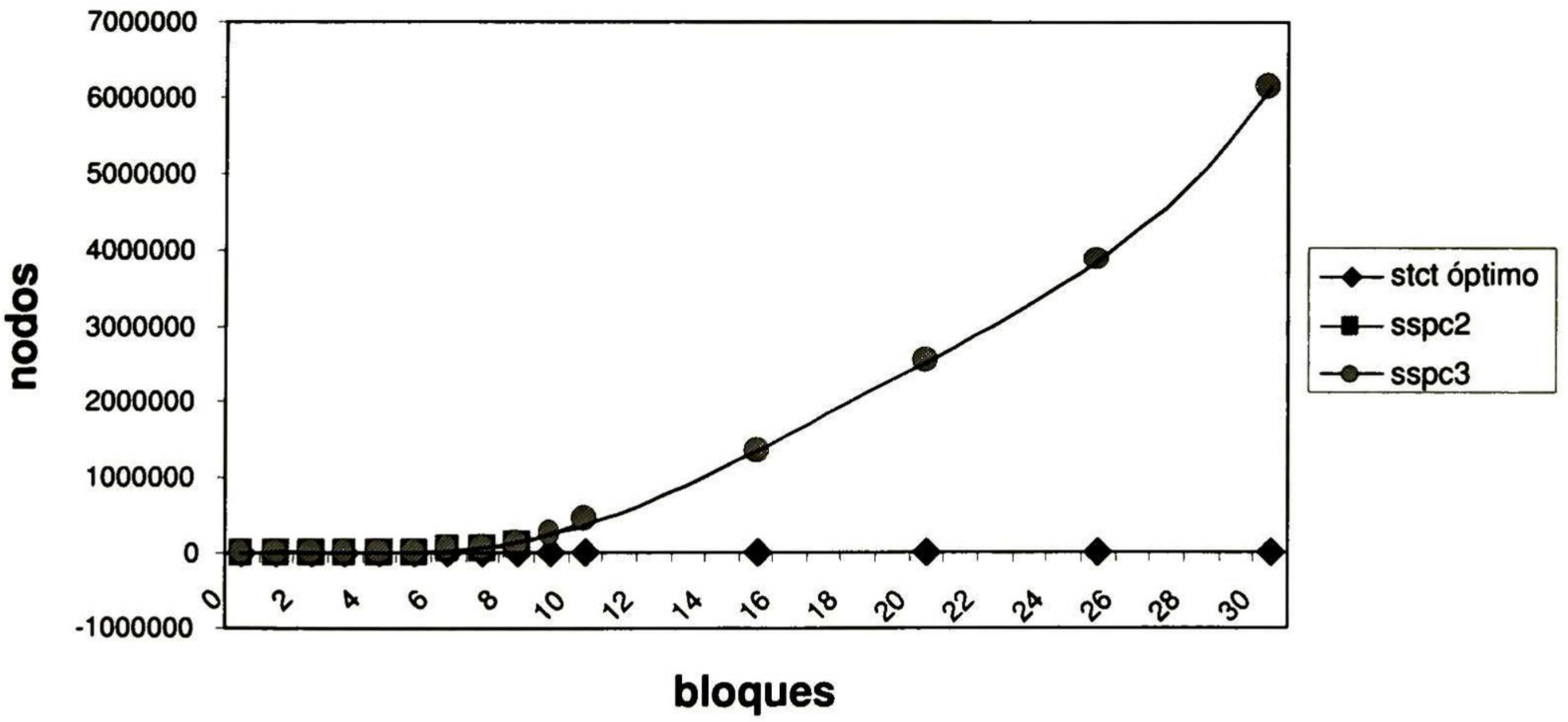

Figura 5.9: Comparación de la cantidad de nodos usados para el cálculo del control supervisor entre STCT, SSPC v.2 y SSPC v.3

5.2.4. Análisis de complejidad del tiempo de cálculo

Con el fin de dar una idea sobre el comportamiento de la herramienta SSPC al aumentar la cantidad de líneas de transferencia del sistema con el cual se trabaja, se muestra la tendencia del tiempo de cálculo para el control supervisor en la figura 5.10. Como se puede apreciar en la gráfica, se tiene un comportamiento exponencial.

Mediante un ajuste de curva se tiene que el comportamiento de la herramienta SSPC para el tiempo de cálculo del control supervisor está dado por la ecuación: $t = 3.6386M^{3,8615}$, donde M es el número de líneas de transferencia.

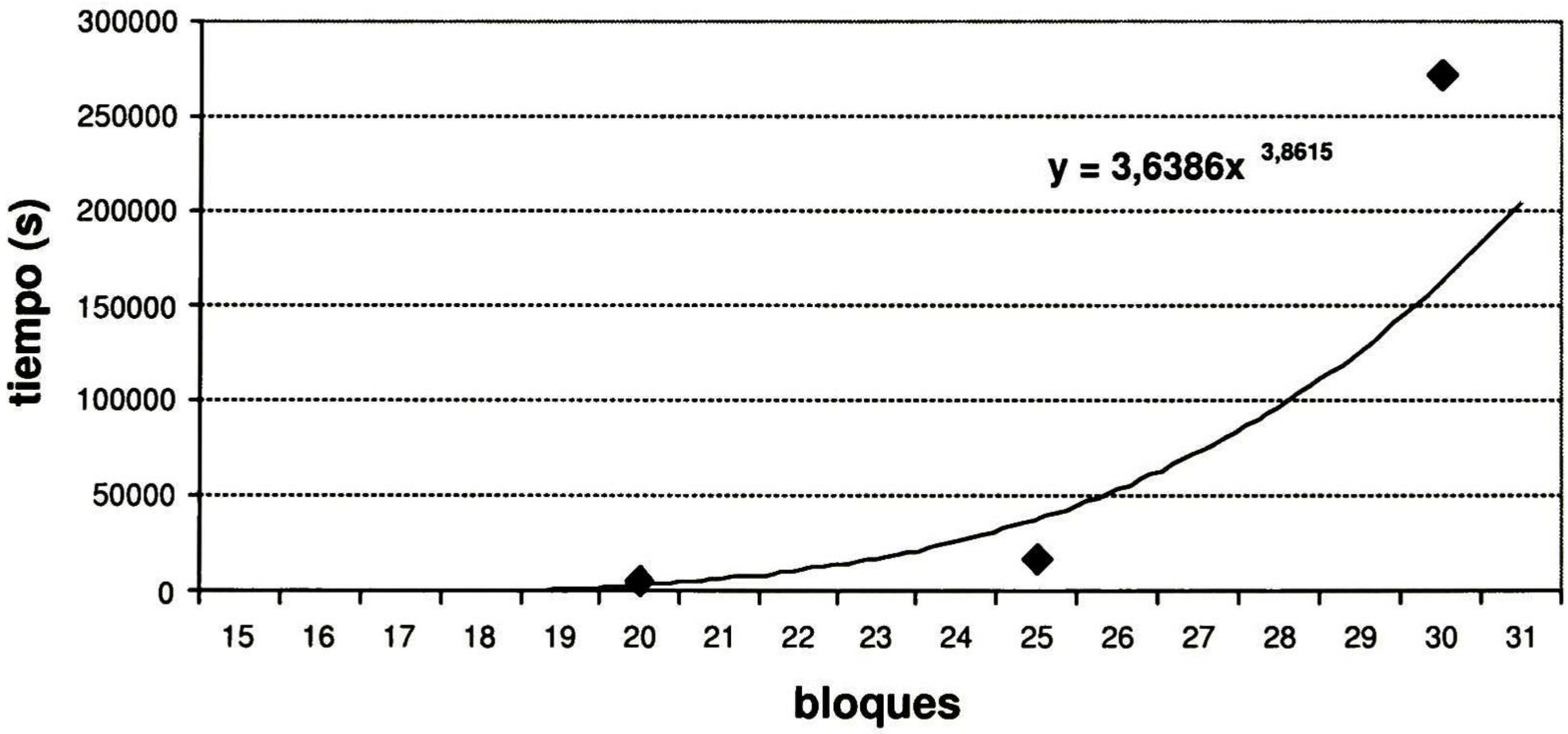

Figura 5.10: Comportamiento del tiempo de cálculo para el control supervisor

5.2.5. Comportamiento de los nodos utilizados con respecto al tamaño del sistema

Cuando se utilizan estructuras de datos simbólicas, una manera de medir su desempeño es con base en los nodos utilizados. En la tabla 5.4 se puede apreciar como la razón de crecimiento de los estados potenciales es mucho mayor que la de los nodos utilizados, esto es debido a que al utilizar una representación simbólica de los estados se pueden representar muchos estados con pocos nodos.

Zhang [30] y Douriet [7] realizaron un procedimiento de análisis del comportamiento de los nodos utilizados con respecto al tamaño del sistema mediante el cual se puede llegar a la siguiente expresion:

$$|nodos| = O(M^C) \tag{5.1}$$

En la figura 5.9 se mostró el comportamiento de los nodos utilizados con respecto al número de bloques M del sistema. De esta gráfica se puede observar que la relación es de tipo exponencial de la forma mostrada en la ecuación 5.1.

Douriet [7] calculó el valor de C para cada M de manera experimental despejando de la siguiente manera:

$$C = \log_M |nodos| \tag{5.2}$$

de esta forma obtuvo una aproximación de la tendencia de C cuando M tiende a infinito la cual fue de 3.37.

Sin embargo, mediante un ajuste de curva de los datos mostrados en la tabla 5.4 se tiene que el comportamiento de la herramienta SSPC versión 3 en cuanto a los nodos utilizados con respecto al tamaño del sistema está dado por la ecuación: $|\text{nodos}| = 1558.9M^{2,4455}$ Esto se muestra en la figura 5.11.

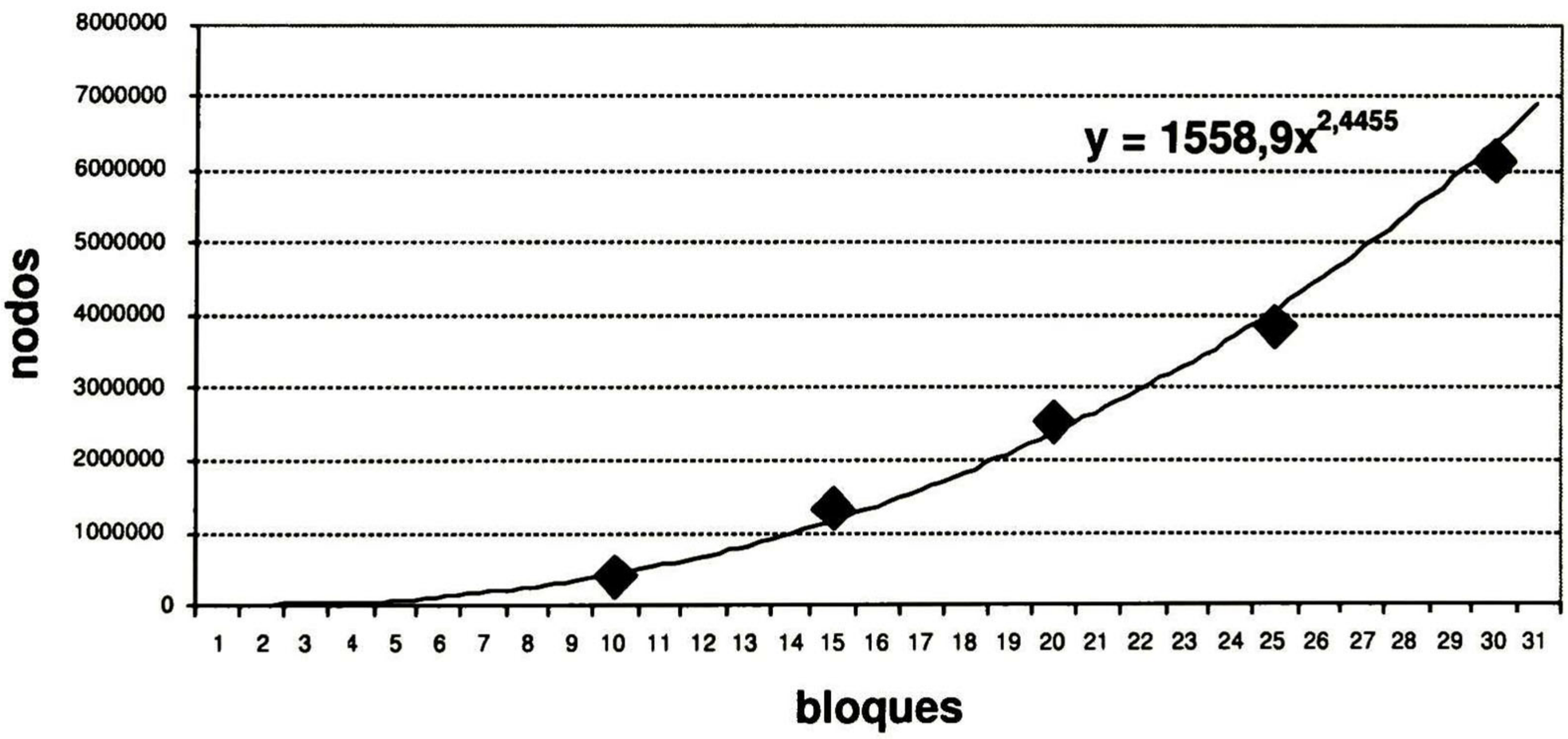

Figura 5.11: Comportamiento de los nodos en el cálculo del control supervisor

5.2.6. Tiempo usado en el reordenamiento de variables de los BDDs

Ahora, se presenta una comparación del tiempo usado para optimizar la realización de los cálculos para la síntesis de los supervisores en las herramientas SSPC versión 3 y STCT.

Para realizar esta comparación se toma el tiempo utilizado por la herramienta SSPC para el reordenamiento dinámico de variables de BDDs mediante el algoritmo sifting y el tiempo utilizado por STCT para el reordenamiento de los componentes elementales de la planta y la especificación. Los resultados se muestran en la tabla 5.5 y en la figura 5.12.

♯ Bloques	SSPC versión 3		STCT	
	Tiempo de reordenamiento (s)	Porcentaje del tiempo de reordenamiento (%)	Tiempo de reordenamiento (s)	Porcentaje del tiempo de reordenamiento (%)
1	0.2	95.65	< 1	100.00
5	4.5	93.16	1.5	75.00
10	44.2	84.96	79	84.04
15	331.7	78.15	1,426	93.81
20	1,410.0	27.93	9,887	96.62
25	4,057.3	24.29	42,115	92.17
30	10,774.0	3.97	178,731	92.42

Tabla 5.5: Resultados del tiempo usado en el cálculo del supervisor para el reordenamiento de variables de BDDs en SSCC versión 3, y para el reordenamiento de componentes elementales en STCT

Como se puede observar el tiempo usado por SSPC en la optimización mediante el reordenamiento de las variables de los BDDs es menor al usado para el reordenamiento de los componentes elementales en STCT sobre todo cuando los sistemas van aumentando de tamaño. También, se observa en la figura 5.13 que conforme aumenta el tamaño del sistema, SSPC dedica menos porcentaje de tiempo al reordenamiento de variables.

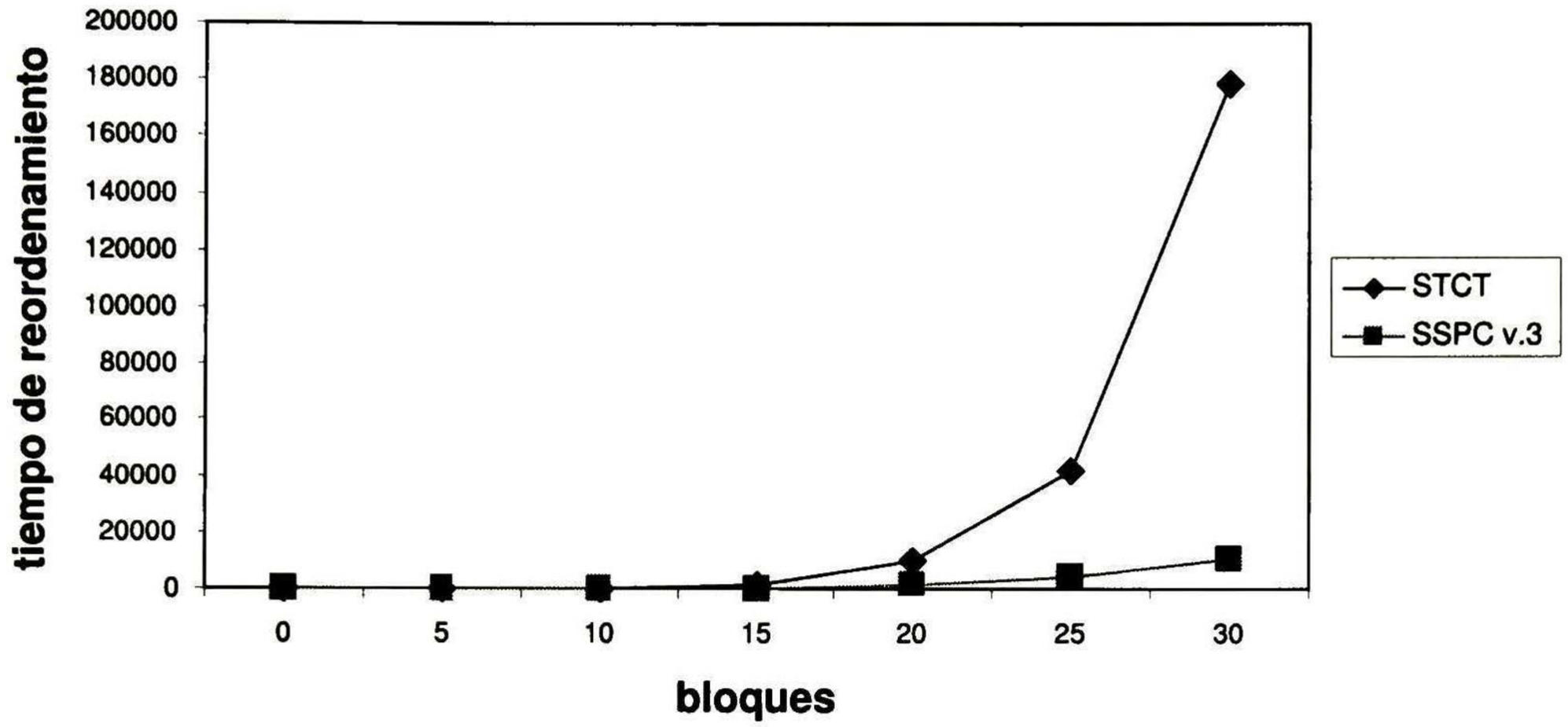

Figura 5.12: Tiempo usado en el cálculo del supervisor para el reordenamiento de variables en SSCC v.3 y de componentes elementales en STCT

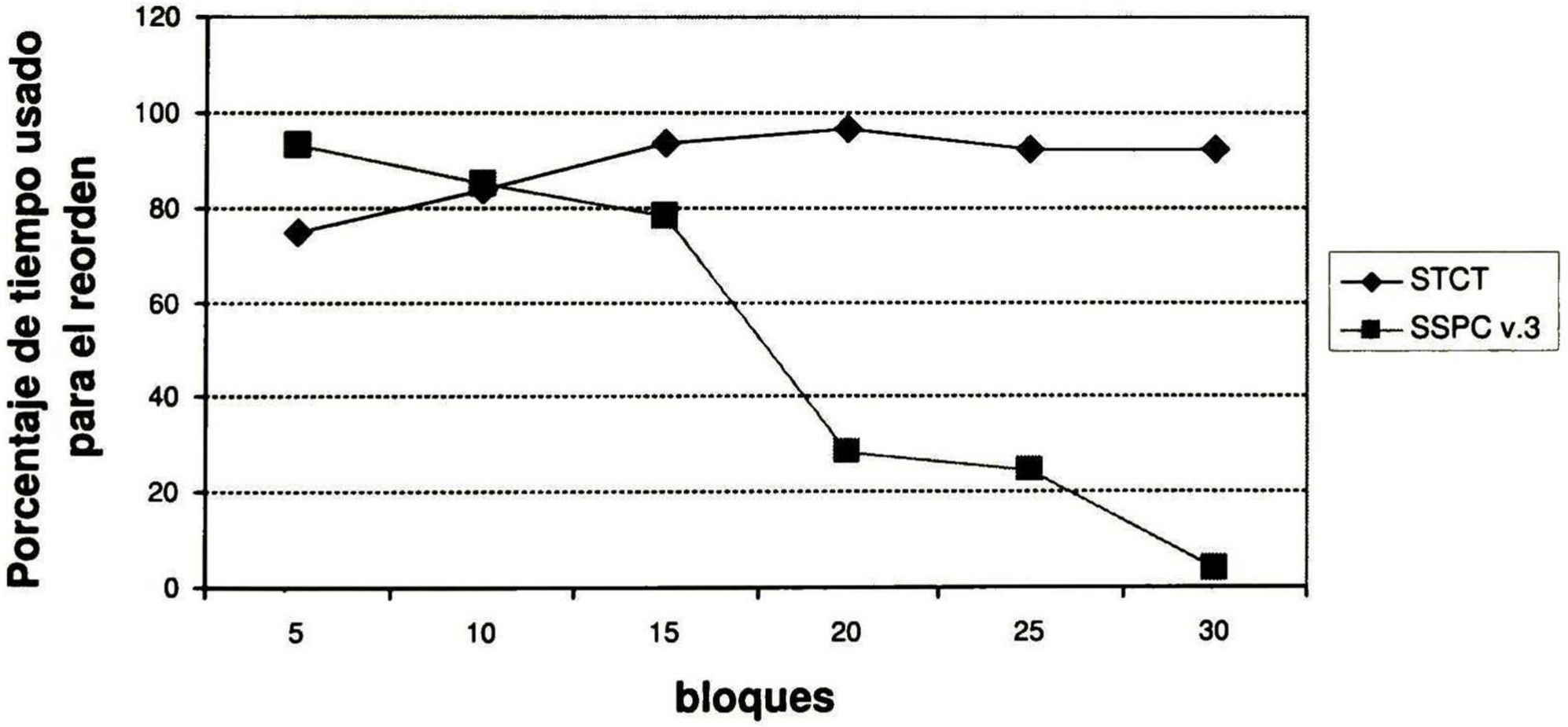

Figura 5.13: Porcentaje del tiempo usado para el reordenamiento con respecto al tiempo total de cálculo del supervisor en SSPC v.3 y STCT

5.3. Superestructura maximal para el controlador de procedimientos

En esta sección se presentan los resultados obtenidos y las comparaciones hechas para el cálculo de la superestructura maximal para el controlador de procedimientos, basada en la definición del supremo sublenguaje controlable.

5.3.1. Tanque presurizado

El ejemplo utilizado es el de una línea de gas que se encuentra conectada a un tanque, como se ilustra en la figura 5.14.

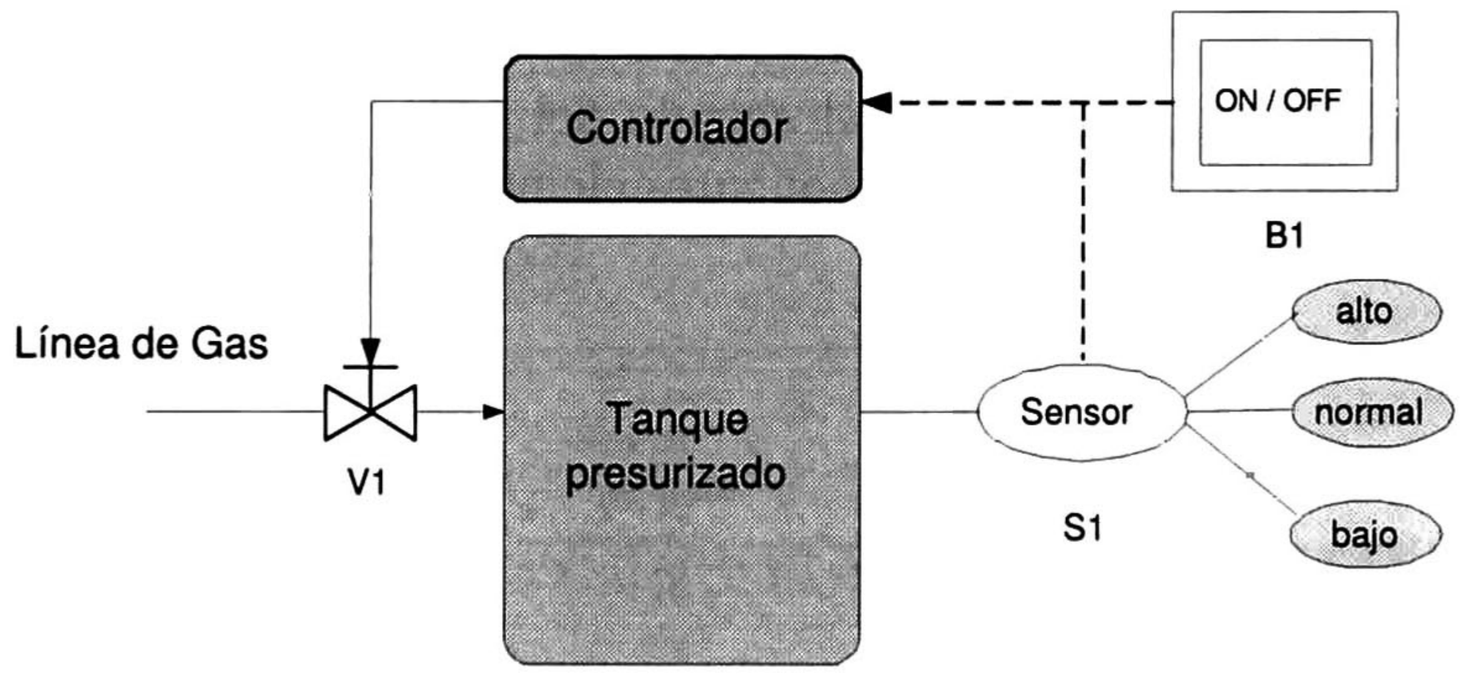

Figura 5.14: Tanque presurizado

El tanque se presuriza mediante el uso de una válvula de 2 posiciones *V1* hasta que se alcanza su valor de presión deseado. La presión en el tanque es medida por medio de un sensor (*S1*), el cual contiene tres valores de presión (bajo, normal y alto). Si la válvula está cerrada, la indicación de presión no aumenta pero sí puede disminuir debido a la operación intrínseca del tanque. Todo el sistema se activa y desactiva por medio del botón *B1*, el cual debe ser presionado por algún operador. La representación mediante MEFs de los componentes de la planta descritos se muestra en la figura 5.15.

Una vez que se abre la válvula, la presión en el tanque puede aumentar, pasando de baja a normal y de normal a alta. Si la válvula se cierra, la presión puede disminuir de

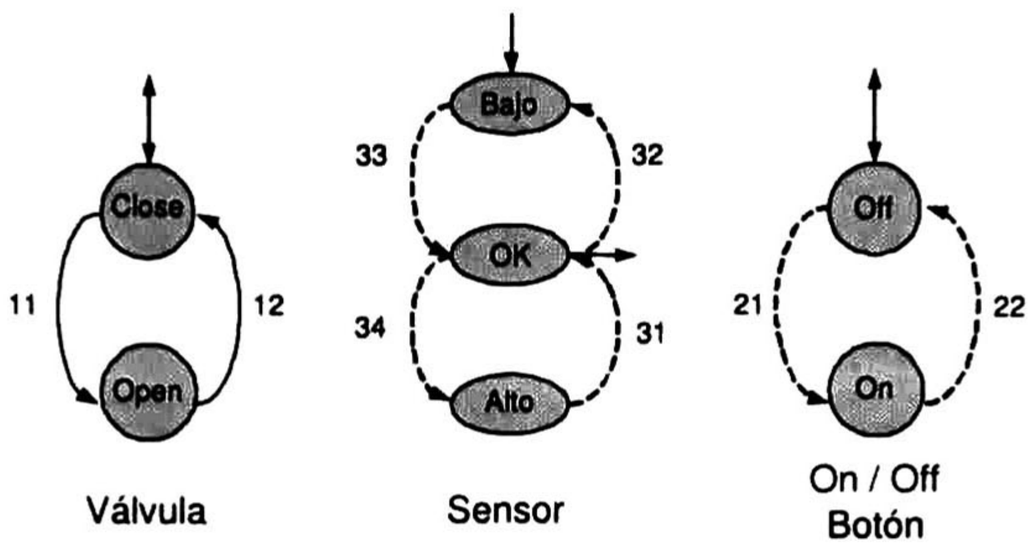

Figura 5.15: Componentes elementales de la planta para el tanque presurizado

alta a normal y de normal a baja. Mediante el modelado del comportamiento causal se eliminan del modelo del comportamiento de la planta aquellas transiciones en donde la presión aumente, partiendo de algún estado en el cual la válvula se encuentre cerrada.

El producto asíncrono de los componentes elementales, incluyendo el comportamiento causal se muestra en la figura 5.16.

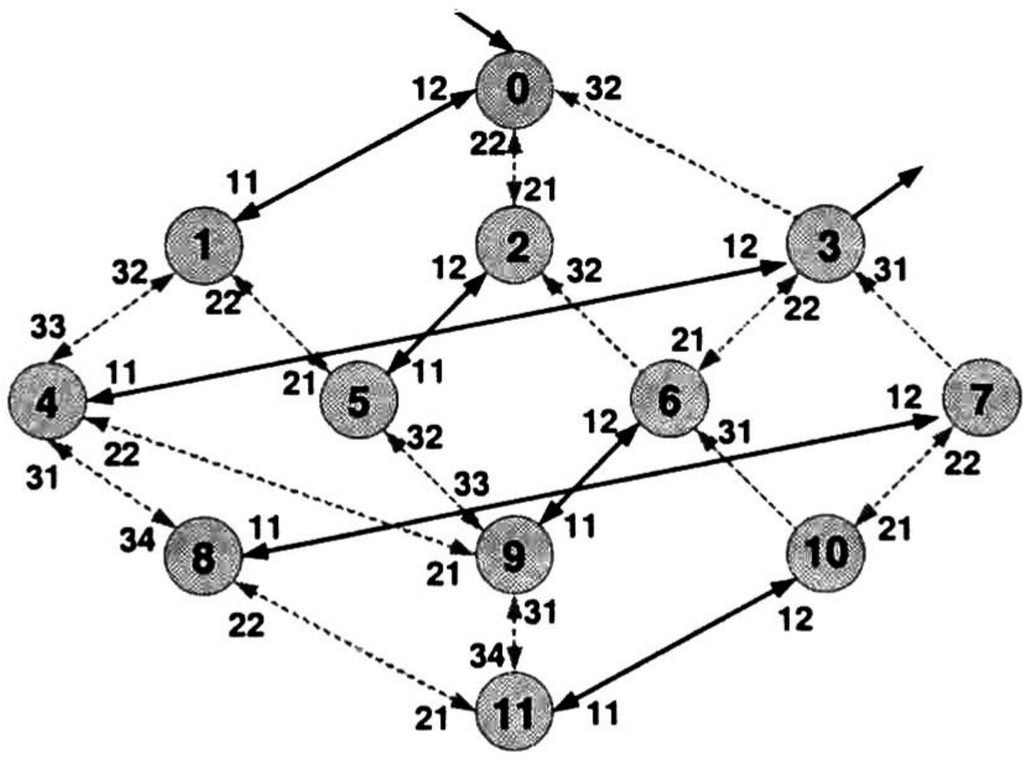

Figura 5.16: Modelo completo de la planta del tanque presurizado una vez aplicado el comportamiento causal

Para iniciar la operación, la válvula debe estar cerrada, el botón sin presionar y la presión de tanque en estado bajo. El proceso comienza cuando se presiona el botón, esto ocasiona que la válvula se abra. Si el operador suelta el botón, la válvula se debe cerrar sin importar el estado en el que se encuentre el dispositivo medidor de presión.

Una vez que se abre la válvula la presión en el tanque puede aumentar, pasando de baja a normal y de normal a alta. Si la válvula se cierra la presión puede disminuir de forma similar.

Cada vez que el tanque se despresuriza (el dispositivo de presión indica valor bajo) el tanque debe presurizarse de nuevo. Si la presión alcanza su valor normal, la válvula debería cerrar sin importar la posición del botón. Por seguridad, la presión nunca deberá estar en estado alto.

El objetivo es sintetizar un dispositivo de control que por medio de señales de control sobre la válvula conduzca la presión del tanque hacia el estado normal, siempre y cuando el botón se encuentre presionado por algún operador.

Para sintetizar el controlador se plantean las siguientes especificaciones:

- Especificación 1: A partir del estado inicial, cuando la presión es baja y la válvula está cerrada, si el operador presiona el botón inmediatamente se debe emitir un comando para abrir la válvula.

- Especificación 2: Si el operador deja de presionar el botón, entonces se debe emitir un comando para cerrar la válvula.

- Especificación 3: Si se detecta la señal de presión normal, entonces se debe emitir un comando para cerrar la válvula.

- Especificación 4: Nunca se debe alcanzar un nivel alto en la medición de la presión en el tanque (estados prohibidos).

Además, se considera que la operación del sistema ha concluido satisfactoriamente cuando la presión se encuentra en un nivel normal, la válvula cerrada y el botón desactivado.

Siguiendo con los mismos ejercicios realizados por Reza [20] y Douriet [7] para la realización de las pruebas de desempeño de la superestructura y del controlador de procedimientos en las versiones anteriores del SSPC, y con el fin de facilitar los cálculos antes mencionados, en las pruebas de desempeño del SSPC versión 3 se usa el ejemplo del tanque presurizado requiriendo que se cumpla únicamente con la segunda

especificación, la cual dice que si el operador deja de presionar el botón, entonces se debe emitir un comando para cerrar la válvula. Dicha especificación se muestra en la figura 5.17. Posteriormente, se realiza la eliminación de los estados prohibidos de acuerdo con la especificación 4, eliminando del sistema todos aquellos estados donde se encuentre la presión en estado alto.

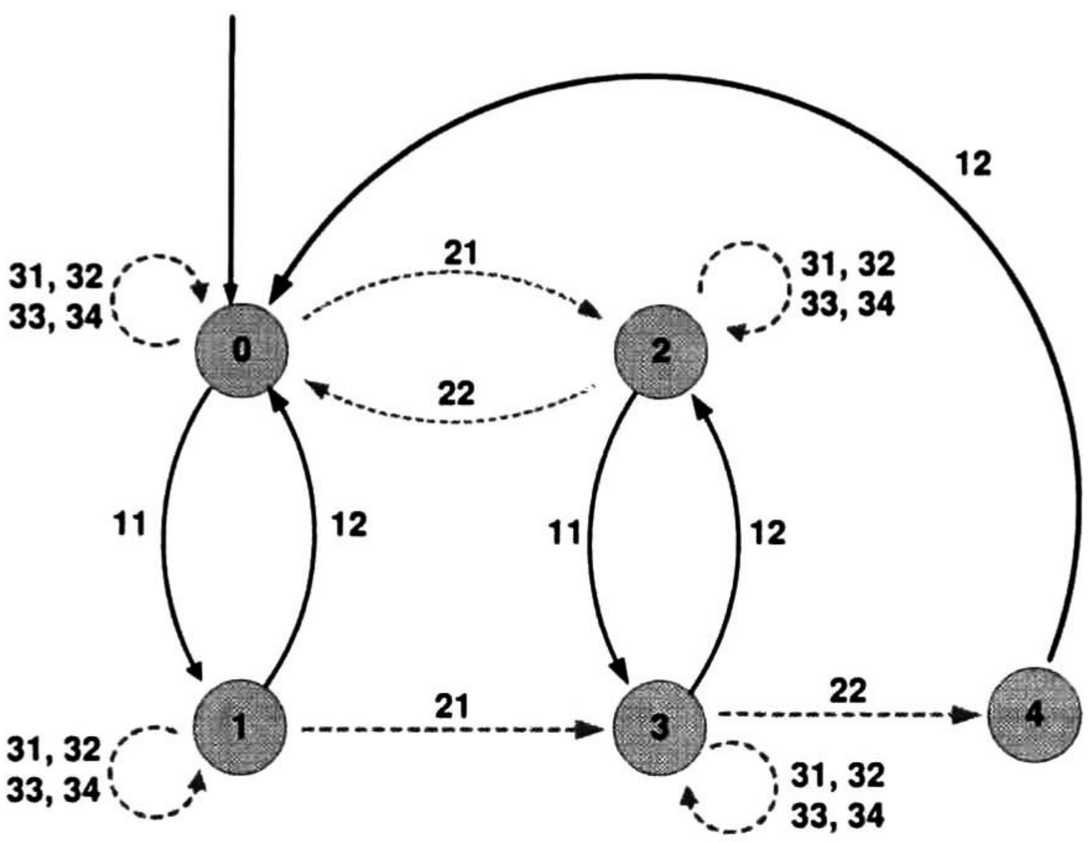

Figura 5.17: Especificación para el tanque presurizado

A partir de la MEF de la planta y de la especificación mostradas en las figuras 5.16 y 5.17 respectivamente, y de la eliminación de los estados prohibidos señalados en la especificación 4, se obtiene la superestructura maximal de controladores mostrada en la figura 5.18.

5.3.2. Crecimiento del sistema

El crecimiento por bloques se hace agregando sistemas de tanques presurizados, que funcionan en forma independiente. De manera similar a las pruebas de control supervisor, se desea encontrar un controlador global para todos los tanques que permita garantizar que en cada uno de ellos se cumpla la especificación de seguridad requerida. El sistema crece en potencias de 12 y la especificación en potencias de 5.

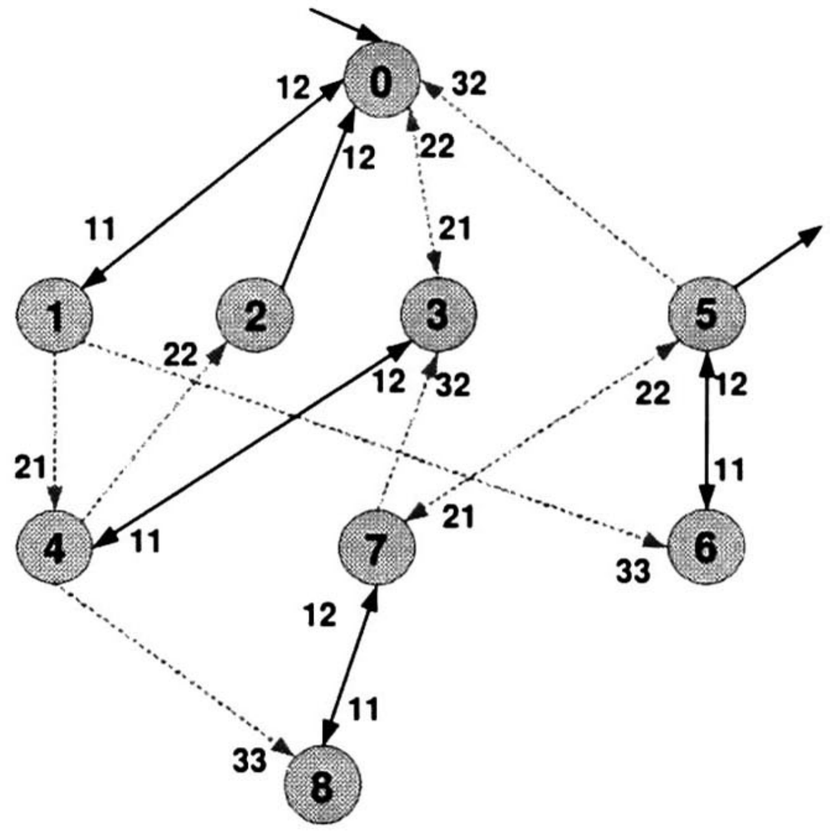

Figura 5.18: Superestructura del controlador de procedimientos

5.3.3. Resultados obtenidos

En la tabla 5.6 se muestran los resultados obtenidos para la síntesis de la superestructura maximal por SSPC versiones 2 y 3. En el SSPC versión 3, desarrollado durante este trabajo, se logró obtener una superestructura para un espacio de estados potencial de hasta 10^{12}.

♯ Bloques	$\|Q_P\|$	$\|Q_E\|$	$\|Q_{SE}\|$	SSPC v. 2			SSPC v. 3		
				T(s)	M(Mb)	Nodos	T(s)	M(Mb)	Nodos
1	12	5	9	<0.01	13.55	133	0.43	13.58	77
2	144	25	80	0.02	14.20	1,206	1.40	14.22	237
3	1,728	125	704	0.17	14.82	7,700	3.52	14.84	815
4	20,736	625	6,144	1.4	18.00	44,683	7.32	15.44	1,134
5	248,832	3,125	53,248	20.02	37.40	249,668	14.65	16.10	2,117
6	2,985,984	15,625	458,752	200.99	128.15	1,368,641	39.83	16.72	7,940
8	4.29×10^8	390,625	3.35×10^7	N/D	N/D	N/D	151.42	22.48	15,040
10	6.19×10^{10}	9.76×10^6	2.41×10^9	N/D	N/D	N/D	1,034.80	60.34	23,969
12	8.91×10^{12}	2.44×10^8	1.71×10^{11}	N/D	N/D	N/D	10,281.00	418.50	28,861

Tabla 5.6: Resultados de tiempo de cómputo, uso de memoria y nodos existentes en el cálculo de la superestructura maximal usando SSPC versiones 2 y 3

Como se observa, nuevamente la herramienta desarrollada durante este trabajo supera a su predecesora en todos los aspectos: tiempo de cálculo, uso de memoria y cantidad de nodos utilizados. Esto se puede ver más claramente en las comparaciones gráficas mostradas en las figuras 5.19, 5.20 y 5.21.

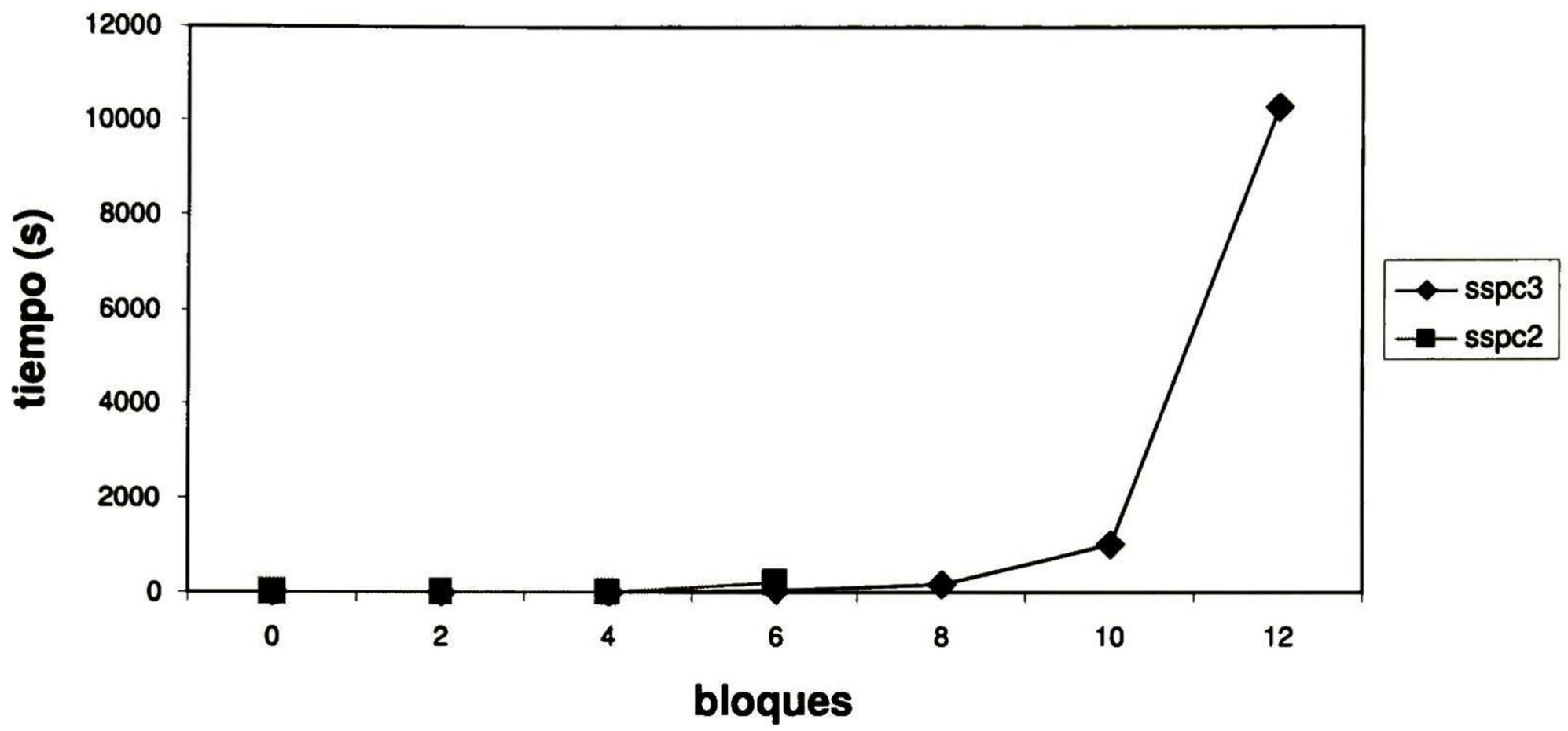

Figura 5.19: Comparación del tiempo de cálculo de la superestructura maximal entre SSPC v.2 y SSPC v.3

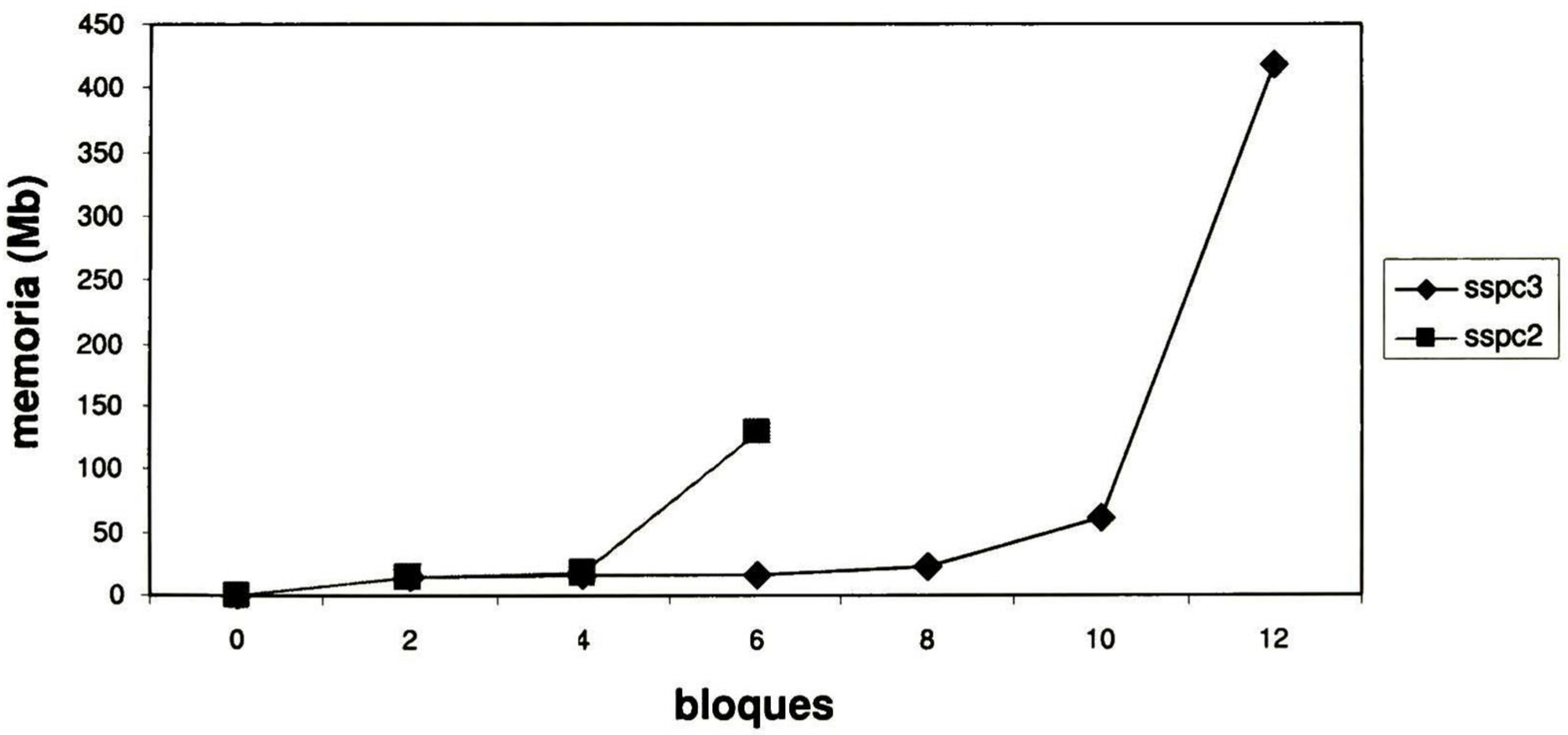

Figura 5.20: Comparación del uso de memoria para el cálculo de la superestructura maximal entre SSPC v.2 y SSPC v.3

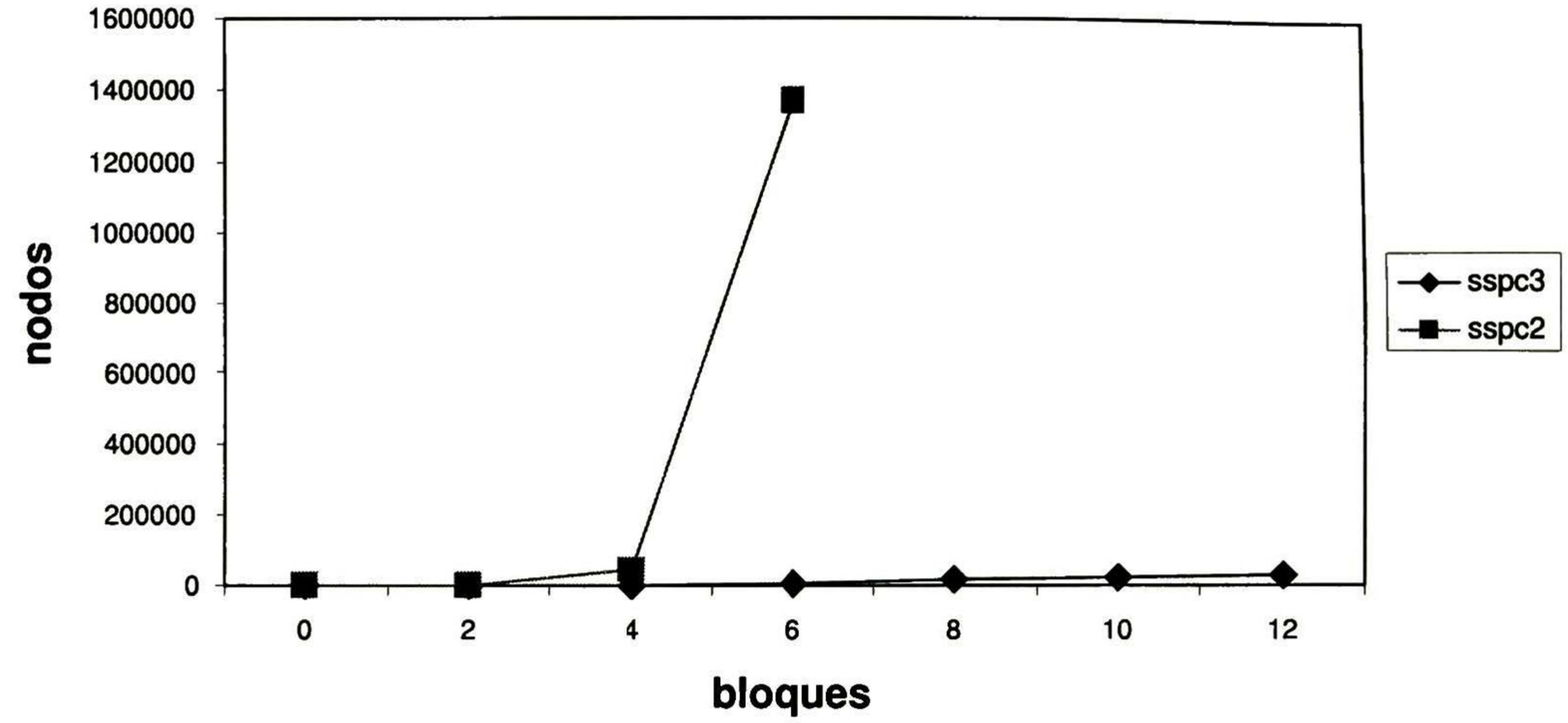

Figura 5.21: Comparación de la cantidad de nodos usados para el cálculo de la superestructura maximal entre SSPC v.2 y SSPC v.3

5.4. Controlador de procedimientos

En esta sección se presentan los resultados obtenidos para el cálculo del controlador de procedimientos. Aquí se comparan los resultados obtenidos con el SSPC versiones 2 y 3.

Los controladores de procedimientos se obtienen a partir de las superestructuras obtenidas en la sección anterior. En el caso de un sólo bloque, el controlador de procedimientos es el mostrado en la figura 5.22.

5.4.1. Resultados obtenidos

Los resultados obtenidos mediante el SSPC versiones 2 y 3 se muestran en la tabla 5.7. Como se puede observar, la mejora es poca. Esto porque la única modificación que se le hizo a este procedimiento fue la aplicación del algoritmo sifting para el reordenamiento dinámico de variables. Además, el algoritmo que se utiliza para obtener el controlador es muy tardado debido a que analiza cada estado individualmente.

En la tabla 5.7 se observa que se llegó a obtener un controlador para seis bloques, lo que corresponde a un espacio de estados potencial de 10^6 Además, también se puede

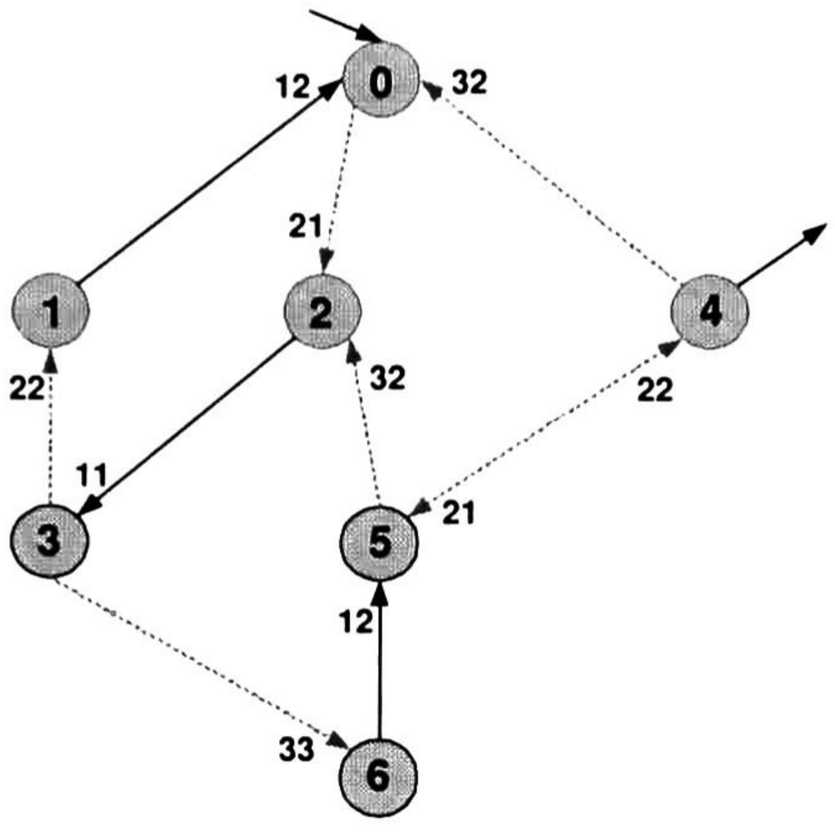

Figura 5.22: Controlador de procedimientos

ver que en algunos casos hay una diferente cantidad de estados en los controladores obtenidos a partir de una misma superestructura. Esto se puede explicar recordando que una superestructura contiene a todos los posibles controladores; entonces, como al momento del reordenamiento dinámico de variables de BDDs, existe un reacomodo de datos que provoca que en las diferentes versiones del SSPC el primer controlador que se encuentre en cada versión sea diferente; y, ya que el algoritmo usado siempre toma al primer controlador que se encuentre, entonces los resultados pueden ser diferentes.

Los resultados de las mejoras se muestran gráficamente en las figuras 5.23, 5.24 y 5.25, respectivamente. Se mejoró el tiempo de cálculo de los controladores, sin embargo después de varios días no se logró calcular el controlador a partir de la superestructura para 7 bloques. También se disminuyó el uso de memoria y sobre todo la cantidad de nodos utilizados.

♯ Bloques	$\|Q_{SE}\|$	SSPC v. 2				SSPC v. 3			
		$\|Q_C\|$	T(s)	M(Mb)	Nodos	$\|Q_C\|$	T(s)	M(Mb)	Nodos
1	9	7	<0.01	13.50	105	7	0.01	13.75	75
2	80	35	0.04	14.20	608	35	0.03	14.35	277
3	704	152	1.2	14.80	2,289	195	0.91	14.96	783
4	6,144	605	47.29	18.00	6,445	692	20.81	15.55	1,233
5	53,248	2,374	292,850	37.40	16,180	3,558	392.54	19.24	2,027
6	458,752	N/D	N/D	N/D	N/D	15,115	12,951	70.62	3,016

Tabla 5.7: Resultados de tiempo de cómputo, uso de memoria y nodos existentes en el cálculo del controlador de procedimientos usando SSPC versiones 2 y 3

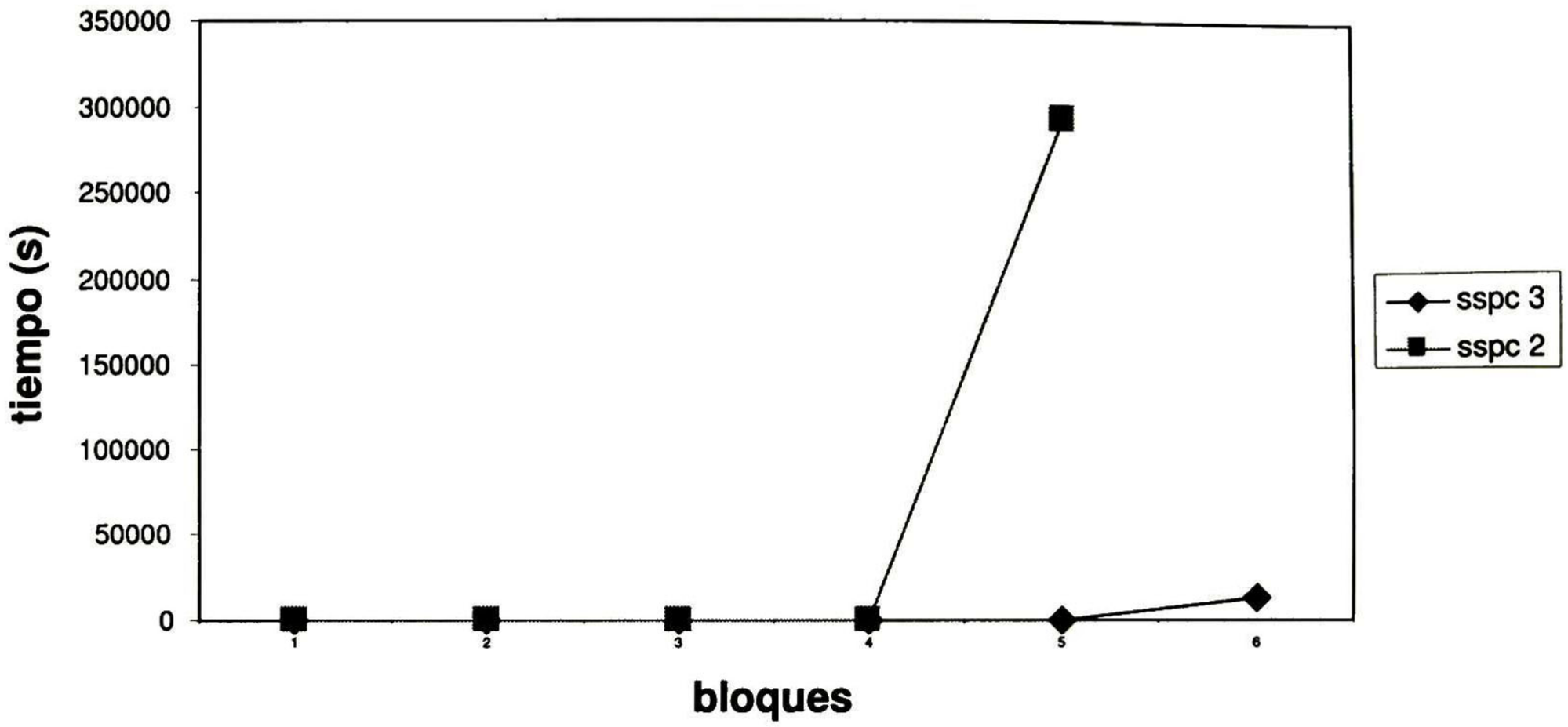

Figura 5.23: Comparación del tiempo de cálculo del control de procedimientos a partir de la superestructura entre SSPC v.2 y SSPC v.3

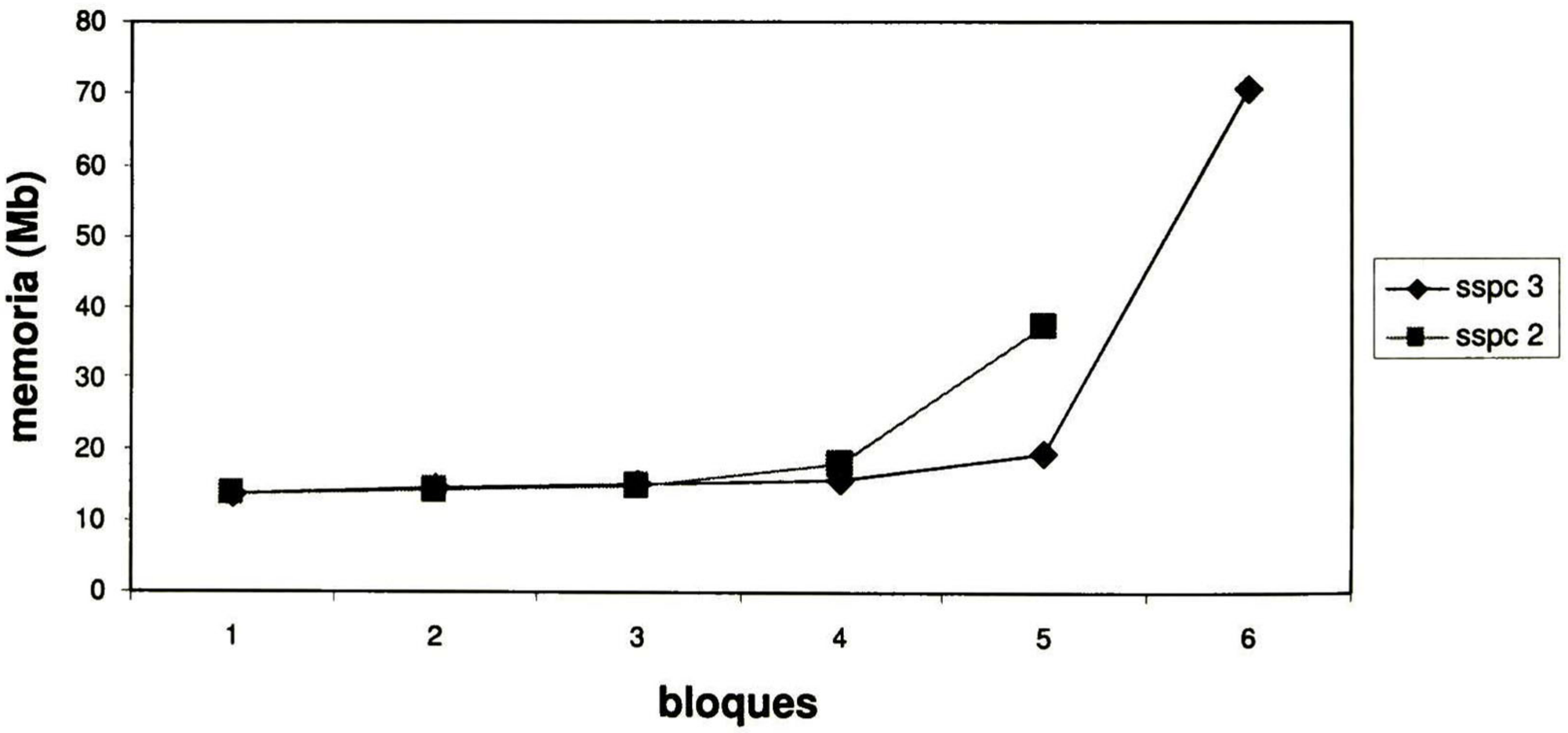

Figura 5.24: Comparación del uso de memoria para el cálculo del control de procedimientos a partir de la superestructura entre SSPC v.2 y SSPC v.3

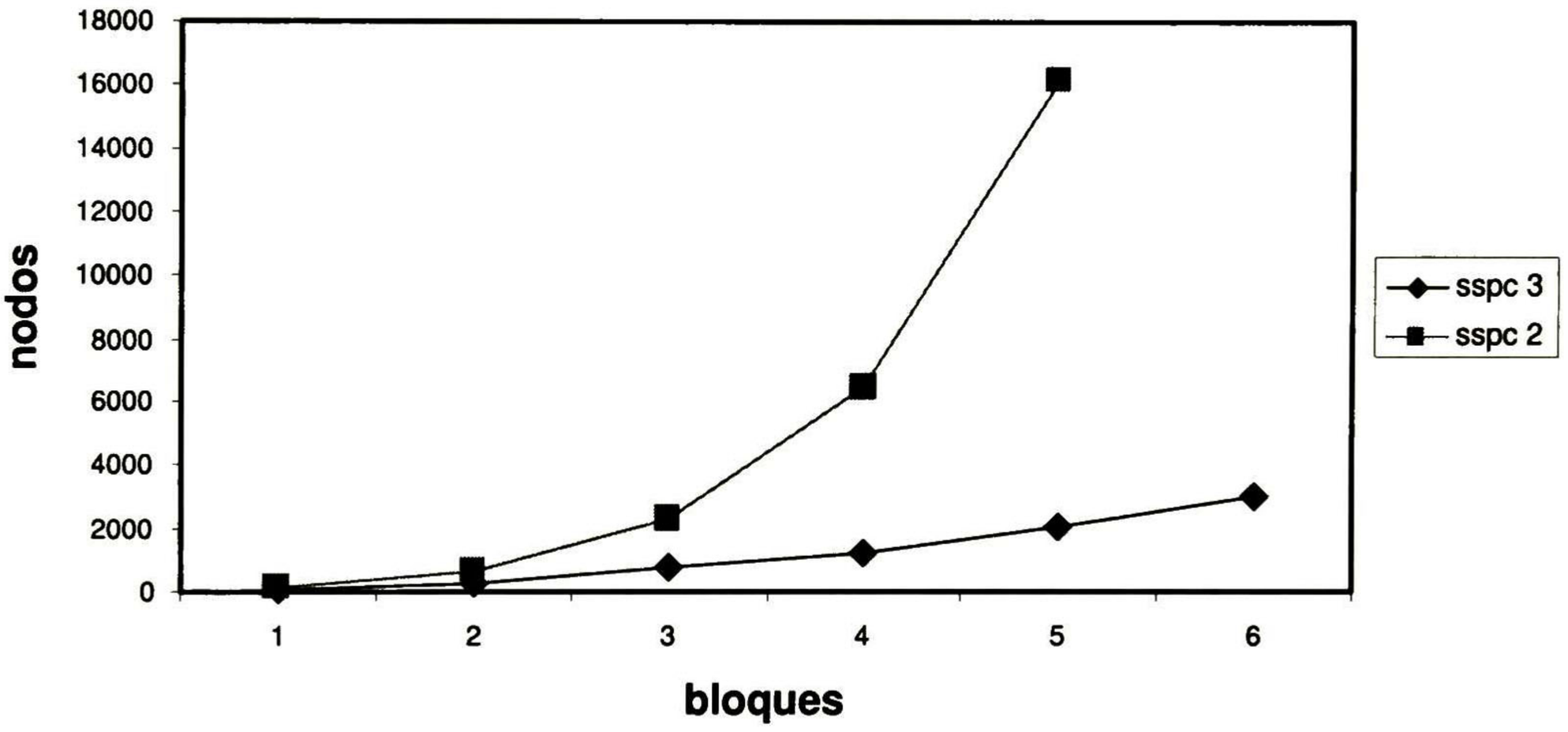

Figura 5.25: Comparación de la cantidad de nodos usados para el cálculo del control de procedimientos a partir de la superestructura entre SSPC v.2 y SSPC v.3

Capítulo 6

Conclusiones y trabajo futuro

En este capítulo se muestran los resultados finales alcanzados por la herramienta computacional SSPC versión 3, la cual logró mejorar a la versión anterior en la síntesis de controladores. Además, se mencionan las ventajas y desventajas de los nuevos algoritmos y técnicas implementadas, así como el trabajo a realizar en el futuro de esta investigación.

Finalmente, se presentan las conclusiones finales del trabajo de tesis desarrollado y de los algoritmos implementados en la herramienta SSPC.

6.1. Resultados obtenidos

Los resultados obtenidos con la nueva implementación del SSPC superan a las versiones anteriores en cuanto a la síntesis de controladores.

- Se logró realizar el cálculo de un control supervisor de $2{,}21 \times 10^{23}$ para un espacio de estados de $1{,}23 \times 10^{27}$ estados, superando también a otras herramientas para la síntesis del control supervisor.
- En cuanto al cálculo de la superestructura maximal, se logró encontrar una superestructura de $1{,}71 \times 10^{11}$ a partir de un espacio de estados de $8{,}91 \times 10^{12}$
- Finalmente, en cuanto al cálculo del controlador de procedimientos a partir de la superestructura maximal, se logró sintetizar un controlador de $15,115$ estados para un espacio de estados de $2{,}98 \times 10^{6}$

6.2. Ventajas y desventajas

En esta sección se presentan las ventajas y desventajas de los algoritmos implementados en este trabajo de tesis.

6.2.1. Ventajas

Las ventajas principales de los algoritmos de síntesis de controlador supervisor, y de la superestructura maximal aplicadas junto con el algoritmo incremental y el ordenamiento dinámica de variables, son:

- No se necesita realizar el producto síncrono planta-especificación. El cual es un proceso tardado y que limita cálculos posteriores debido a su gran uso de memoria.

- Los algoritmos de síntesis trabajan con puros estados, lo cual permite trabajar con estructuras de datos más pequeñas.

- Garantiza un supervisor completo y no bloqueante.

- Se trabaja con predicados, esto permite aprovechar las ventajas del cálculo simbólico.

- Los algoritmos usados favorecen la solución incremental especificación por especificación, gracias a esto se evita empezar con MEFs grandes e ir eliminando estados.

- Mediante el reordenamiento dinámico se reduce considerablemente el tamaño de las estructuras de datos utilizados.

6.2.2. Desventajas

A pesar de las grandes ventajas que tienen los algoritmos usados, también existen algunas desventajas:

- Para especificaciones que permitan tener un producto síncrono chico entre una planta grande y las especificaciones, tarda más la nueva implementación ya que trabaja con la planta y con cada especificación una por una, mientras que la anterior implementación parte de la MEF sincronizada cuyo tamaño ya fue reducido por las especificaciones. Estos casos no pueden ser identificados a simple vista y dependerán mucho de que tan restrictivas hayan sido hechas las especificaciones. Debido a esta desventaja la opción de utilizar la implementación anterior no fue removida de la herramienta SSPC.

- El algoritmo de síntesis del controlador de procedimientos a partir de la superestructura, está diseñado para trabajar con un método explícito de cálculo y por tanto su implementación con BDDs no aprovecha ventajas del cálculo simbólico como la representación de muchos estados mediante el uso de unas pocas variables, lo cual permitiría ahorrar operaciones. Esto debido a que en este algoritmo se analiza cada estado individualmente para obtener el controlador.

- Se tienen que hacer ciertos pasos para combinar la representación de solo estados y la representación mediante la relación de transición.

6.3. Trabajo a futuro

El trabajo a realizar en el futuro sobre este mismo tema de investigación, contempla los siguientes puntos:

- Investigación de nuevas técnicas de implementación de estructuras simbólicas, tales como los diagramas de decisión enteros (IDDs).

- Implementar el uso del algoritmo incremental mediante los componentes elementales de la planta y la especificación, es decir, evitar la generación del modelo

completo de la planta.

- Realizar una reestructuración del SSPC que permita efectuar procedimientos para operaciones independientes y procedimientos completos para síntesis de supervisores y controladores.
- Crear una versión del SSPC tipo librerías.
- Buscar un algoritmo de síntesis del controlador de procedimientos a partir de la superestructura, el cual esté diseñado para trabajar con métodos de cálculo simbólico e implementarlo para aprovechar las ventajas de los BDDs.

6.4. Conclusiones finales

Se mejoró la herramienta SSPC para la síntesis de controladores para sistemas a gran escala. Esta nueva implementación es capaz de trabajar con sistemas de tipo industrial reales.

La nueva implementación superó la capacidad de cálculo para supervisores, superestructuras y controladores de procedimientos de las versiones anteriores del SSPC y de otras herramientas existentes. Se mejoraron los tiempos de cálculo de la herramienta y disminuyó el uso de memoria.

Se lograron implementar algoritmos que trabajan con puros estados y que además necesitan sólo la información de la MEF del sistema y de las especificaciones por separado, lo cual permite el uso del cálculo incremental especificación por especificación. De esta forma, se consiguió evitar la necesidad del cálculo del producto síncrono entre la planta y la especificación. Además, se logró trabajar con estructuras de datos más compactas y eficientes mediante la aplicación del reordenamiento dinámico de BDDs.

Referencias

[1] E. Adl, M. K. Rodriguez, A. A. Tsakalis, and S. Kostas. Hierarchical modeling and control of re-entrant semiconductor facilities. In *Proc. of 35th IEEE Int. Conf. on Decision and Control*, Kobe, Japón, 1996.

[2] K. Akesson, H. Flordal, and M. Fabian. Exploiting modularity for synthesis and verification of supervisors. In *15th IFAC World Congress. Barcelona, Spain*, July 2002.

[3] H. R. Andersen. An introduction to binary decision diagrams. Dept. of Information Technology, Technical University of Denmark, 1997.

[4] S. Balemi, G. J. Hoffmann, P. Gyugyi, H. Wong-Toi, and G. F. Franklin. Supervisory control of a rapid thermal multiprocessor. *IEEE Transactions on Automatic Control (joint issue with Automatica)*, 38(7):1040–1059, 1993.

[5] C. G. Cassandras and S. Lafortune. *Introduction to Discrete Event Systems.* Kluwer Academic, 1999.

[6] V Chandra, B. Oruganti, and R. Kumar. Ukdes: A graphical software tool for the design, analysis and control of discrete event systems. *IEEE Transactions on Control Systems Technology, Submitted*, 2000.

[7] J. Douriet. Cálculo eficiente de controladores de procedimientos aplicados a sistemas de producción a gran escala. Tesis de Maestría, Dept. of Elect and Comp Eng., CINVESTAV, México, 2003.

[8] G. Hoffmann and H. Wong-Toi. Symbolic synthesis of supervisory controllers. In *Proc. of 1992 American Control Conference*, pages 2789–2793, Chicago, IL, USA, June 24-26 1992.

[9] A. Hu. Personal communication. 2002.

[10] A. J. Hu and D. L. Dill. Reducing BDD size by exploiting functional dependencies. In *Design Automation Conference*, pages 266–271, 1993.

[11] R. Kumar, V. Garg, and S. Marcus. Using predicate transformers for supervisory control. *In Proceedings of the 30th Conference on Decision and Control, Brighton, UK*, pages 98–103, December 1991.

[12] R. Kumar and L. E. Holloway. Supervisory control of petri net language. In *IEEE Conference on Decision and Control, Volume 3*, pages 1190–1195, December 1992.

[13] R. Kumar and L. E. Holloway. Supervisory control of deterministic petri nets with regular specification languages. *IEEE Transactions on Automatic Control*, 41(2):245–249, 1996.

[14] Y. Li and W. M. Wonham. Control of vector discrete–event systems. i.– The base model. *IEEE Transactions on Automatic Control*, 38(3):1215–1227, August 1993.

[15] Y. Li and W. M. Wonham. Control of vector discrete–event systems. ii.– Controller synthesis. *IEEE Transactions on Automatic Control*, 39(3):512–531, March 1994.

[16] C. Meinel and A. Slobodova. Speeding up variable reordering of OBDDs. In *International Conference on Computer Design*, pages 338–343, 1997.

[17] K. Milvang-Jensen and A. Hu. BDDNOW: A parallel BDD package. In *Formal Methods in Computer-Aided Design*, pages 501–507, 1998.

[18] J. L. Nielsen. Technical report, 2001. BuDDy: Binary Desision Diagram package, Release 2.0, IT University of Copenhagen (ITU).

[19] R. Ranjan, W. Gosti, R. K. Brayton, and A. L. Sangiovanni-Vincentelli. Dynamic reordering in a breadth-first manipulation based BDD package: Challenges and solutions. In *International Conference on Computer Design*, pages 344–351, 1997.

[20] J. Reza. Sintesis de controladores de procedimientos utilizando tecnicas de calculo simbolico. Tesis de Maestría, Dept. of Elect and Comp Eng., CINVESTAV, México, 2002.

[21] R. Rudell. Dynamic variable ordering for ordered binary decision diagrams. *International Conference on Computer-Aided Design, IEEE,*, pages 42–47, 1993.

[22] A. Sánchez. *Formal Specification and Synthesis of Procedural Controllers for Process Systems.* Lecture Notes on Control and Information Sciences, v. 212, Springer–Verlag, 1996.

[23] A. Sánchez and A. Morales. Technical report, 2003. Diseño de la Arquitectura ISA S88 del Sistema de Control de Procedimientos para el Proceso de Carga de Crudo Istmo de Exportación a Buque-tanques en TMDB. Programa de Matemáticas Aplicadas y Computación. Instituto Mexicano del Petróleo.

[24] A. Sánchez, J. Reza, J. Douriet, and R. E. González. A comparison of synthesis tools for supervisory controllers. In *Proc. of European Control Conference Cambridge, G.B.*, September 1-4 2003.

[25] A. Sánchez, G. Rotstein, N. Alsop, and S. Macchietto. Synthesis and implementation of procedural controllers for event–driven operations. *AIChE Journal*, 45(8):1753–1775, 1999.

[26] E. Tronci. Automatic synthesis of controllers from formal specifications. In *Procedings of 2nd IEEE Int. Conf. on Formal Engineering Methods*, pages 134–143, Brisbane, Australia, december 1998.

[27] J. J. Venegas. Controladores de procedimientos para especificaciones de seguridad utilizando cálculo explícito incremental. Tesis de Maestría, Dept. of Elect and Comp Eng., CINVESTAV, México, 2002.

[28] W. M. Wonham. Notes on control of discrete-event systems. Dept. of Electrical and Computer Engineering, University of Toronto, 1999.

[29] W. M. Wonham and P.J. Ramadge. On the supremal controllable sublanguage of a given language. *SIAM Journal of Control and Optimization*, 25(3):637–659, 1987.

[30] Z. Zhang. Stct: An efficient algorithm for supervisory control design. Master's thesis, Dept. of Elect and Comp Eng., University of Toronto, Canada, 2001.

[31] Z. Zhang. Personal communication. 2002.

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

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

Síntesis de Controladores basados en Autómatas para Sistemas Dinámicos de Eventos Discretos a Gran Escala

del (la) C.

Eduardo Isaac RAMÍREZ JIMÉNEZ

el día 13 de Diciembre de 2004.

Dr. Arturo del Sagrado Corazón
Sánchez Carmona
Investigador CINVESTAV 3B
CINVESTAV Unidad Guadalajara

Dr. José Javier Ruíz León
Investigador CINVESTAV 3A
CINVESTAV Unidad Guadalajara

Dr. Raúl Ernesto González Torres
Investigador CINVESTAV 2C
CINVESTAV Unidad Guadalajara