××(178613.1)

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

Generación y Uso de Abstracciones en la Comprobación de Modelos Simbólica

Tesis que presenta:

Carlos Dario Arenas Yerena

para obtener el grado de:
Maestro en Ciencias

en la especialidad de:
Ingeniería Eléctrica

Director de Tesis
Dr. Raúl Ernesto González Torres

Guadalajara, Jalisco, Febrero de 2009.

Generación y Uso de Abstracciones en la Comprobación de Modelos Simbólica

Tesis de Maestría en Ciencias
Ingeniería Eléctrica

Por:

Carlos Dario Arenas Yerena

Ingeniero Electrónico en Computación

Centro de Enceñanza Técnica Industrial 1999-2004

Becario de CONACYT, expediente no. 203121

Director de Tesis

Dr. Raúl Ernesto González Torres

CINVESTAV del IPN Unidad Guadalajara, Febrero de 2009.

Agradezco...

A mis padres María de la Luz y Calixto Jesús por todo el amor y la formación que me han dado.

A mis hermanos Priscila, Paulina y Jesús por su apoyo incondicional.

A mis profesores, especialmente mi asesor el Dr. Raúl Ernesto González por todo lo que he aprendido de ellos.

A todos mis amigos que han multiplicado mis alegrías y compartido mis tristezas.

A Pedro Arturo Cornejo Torres que tanto me hecho crecer profesionalmente.

Y a la interminable lista de personas que me han convertido en lo que soy.

Al CONACYT por el apoyo económico recibido.

A Dios.

Resumen

La comprobación de modelos simbólica es la técnica de verificación formal que ha obtenido mejores resultados en la verificación de sistemas industriales. Sin embargo, tiene una limitación importante: la explosión de estados, este problema se debe a que el tamaño de los modelos a verificar, es exponencial respecto al número de variables en él.

Es así que surge la necesidad de desarrollar nuevas técnicas que permitan, mediante la reducción del tamaño de los modelos a verificar, aliviar el problema de la explosión de estados; las técnicas a desarrollar deberán además ser automatizables, sólidas y completas, para fin de tener un campo de aplicación práctico.

En este documento presentamos una técnica automatizable que permite realizar la comprobación en modelos creados mediante abstracción existencial de variables, estos modelos preservan completud y típicamente son mucho menores que los modelos concretos; con el objetivo de asegurar la solidez del método, la comprobación se realiza en un marco de abstracción-refinamiento; en este se conjuntan las mejores características de la comprobación de modelos acotada con SAT Solvers y la comprobación de modelos simbólica con BDDs en un algoritmo del estado del arte. Con el propósito de mostrar la efectividad de nuestra técnica, presentamos los resultados obtenidos con su aplicación.

Abstract

Symbolic model checking is the formal verification technique that has best performance in the verification of industrial systems. However, it has a major limitation: the state explosion, the problem is that the size of the models to check is exponential to the number of variables in it.

Thus the need arises to develop new techniques that permit, by reducing the size of the models to check, alleviating the problem of the explosion of states. To be practical those techniques to be developed should be automatable, sound and complete.

We present a technique that allows verification of models created using existential abstraction of variables, these models preserve all behaviors of the original models and they are typically smaller. With the aim of ensuring the soundness of the method, the model checking is performed in an abstraction-refinement framework. Our technique joins the best features of bounded model checking based on SAT Solvers and symbolic model checking with BDDs in a state of the art algorithm. To show the effectiveness of our technique we present the results obtained with its application.

Contenido

Lista de Figuras

Lista de Tablas

Capítulo 1
Introducción

En este capítulo se hace una reseña de las técnicas de verificación que han sido utilizadas tradicionalmente, enfocándonos en las fortalezas y debilidades de éstas. También hablamos de las técnicas formales desarrolladas en las últimas décadas, del creciente éxito de las mismas y de su principal inconveniente: la explosión de estados, la cual tratamos de combatir en este estudio. Por último se hace mención de algunas de las técnicas más exitosas que preceden a esta investigación y se mencionan los objetivos a cumplir.

1.1 Ayer, hoy y mañana

El vertiginoso avance tecnológico en que estamos inmersos implica que los sistemas necesarios para satisfacer nuestras necesidades sean cada vez más y más complejos. Anteriormente una sola persona podía hacerse cargo del proceso

entero de producción de un sistema, pero esos días han quedado atrás, ahora son necesarios grandes grupos de trabajo especializados en las diferentes partes del proceso de producción y cada vez las interacciones entre estos grupos son más complicadas, tenemos así que todas las partes del proceso de producción han cambiado y el proceso de verificación no es la excepción. En el pasado la técnica utilizada para asegurar que un producto cumpliera con los requerimientos era la simulación, en ella se limitaban a realizar pruebas de la funcionalidad típica del producto, exponiéndolo a las condiciones de uso para las cuales fue diseñado, pero al paso de los años los sistemas han aumentado su tamaño, al aumentar su tamaño y complejidad también se incrementa el número de situaciones posibles de uso, cuando el sistema tiene tantos comportamientos esta técnica resulta no ser lo suficientemente buena para asegurar que el producto cumplirá las especificaciones. Por otra parte, tenemos que los errores tardíamente detectados en las líneas de producción de dichos sistemas son más costosos entre más temprana sea la fase del proceso donde se comete el error.

La conjugación de los factores anteriormente expuestos ha forzado a la industria a buscar alternativas que ofrezcan resultados más confiables en el proceso de verificación, y es en esta búsqueda que adquiere importancia una técnica de verificación desatendida anteriormente: la verificación formal de sistemas.

La verificación formal de sistemas ofrece la posibilidad de validar el comportamiento de los sistemas desde un punto de vista más completo y sólido, el punto de vista de la lógica; por otra parte, la verificación formal tiene la gran

ventaja de que algunas de sus principales vertientes son altamente automatizables, esto la convierte en la técnica casi ideal.

Desafortunadamente ha llegado el punto en que el tamaño de los sistemas es tal, que incluso la verificación formal llega a ser incosteable y en algunos casos insuficiente para explorar la totalidad de los comportamientos posibles, es así que surge un gran desafío para el futuro: diseñar técnicas que permitan reducir considerablemente el espacio de búsqueda; las nuevas técnicas deberán también conservar las ventajas antes mencionadas para fin de verificar eficientemente sistemas de tamaño colosal.

1.2 Verificación y validación

Verificación y validación (V&V) es el nombre dado al proceso de análisis y pruebas que nos permitirán asegurar que nuestro sistema cumple con las especificaciones requeridas. Para nuestros fines cabe hacer la distinción entre validación y verificación. La validación asegura que el sistema sea el adecuado para cubrir cierta funcionalidad, mientras que la verificación certifica que el sistema funcione correctamente. Nosotros estamos interesados en la parte correspondiente a la verificación, de esta fase existen muchas variantes las dos principales son las pruebas de simulación y la verificación formal.

1.2.1 Pruebas

La manera más común de realizar las pruebas es mediante la construcción de un prototipo, una vez construido se le coloca en el entorno en que para el cual fue diseñado y se procede a hacer una simulación de todas las entradas que se espera

tendrá el sistema; si en las simulaciones se detecta que el prototipo no cumple con algún comportamiento deseado, se procede a analizar la causa de la falla y si es posible, se corrige el error.

En esencia este enfoque no es malo, pero tiene algunos puntos débiles que vale la pena resaltar: en primer lugar es muy difícil someter al prototipo a todas las entradas posibles, puesto que habrá casos en que estas sean realmente demasiadas para contemplarlas; en segundo lugar, no siempre se podrá probar el prototipo en el entorno para el que fue diseñado, en estos casos será necesario simular también el entorno del sistema, por último, en muchos casos es necesaria la intervención directa del humano en el proceso de pruebas, lo cual es costoso e incrementa el margen de error.

1.2.2 Verificación formal

Mediante la verificación formal de sistemas se intenta demostrar matemáticamente que el diseño del sistema cumple con las especificaciones deseadas, en ella se somete el diseño a todas las entradas posibles y se verifica que en todas ellas el sistema cumpla con el comportamiento deseado. Tiene grandes ventajas respecto al método de pruebas de simulación, entre las que destacan que bajo esta técnica sí es posible simular absolutamente todas las entradas posibles, incluso en los casos en que estas serían demasiadas para las pruebas de simulación. Además se tiene que algunas de las principales técnicas de verificación formal son totalmente automatizables, con esto es posible reducir los tiempos de verificación, los costos y los márgenes de error asociados al factor humano. Por último, para realizar este tipo de pruebas no es necesaria la construcción de prototipos del

sistema que en algunos casos llegan a ser muy costosos y requieren una gran inversión de tiempo.

En este punto cabe pensar que mediante el uso de la verificación formal es posible prescindir del uso de las pruebas de simulación, pero no es ése el caso; los enfoques de estas técnicas no son contrarios sino complementarios y tienen aplicación en distintas partes del proceso de producción, la verificación formal está enfocada en detectar errores cometidos durante la fase de diseño. Por otra parte, las pruebas de simulación nos ayudarán a verificar que el diseño sea implementado de manera correcta teniendo con esto una certeza mayor de que el sistema cumple con las especificaciones deseadas.

Cuando se realiza verificación formal es necesario construir un modelo matemático que represente el sistema diseñado, también es necesario expresar en términos formales las propiedades que se desean verificar en el sistema. Estas propiedades se seleccionan de forma que su cumplimiento asegure en el sistema los comportamientos más esenciales, una vez que se tienen formalmente descritos el modelo de sistema y las propiedades, se procede a verificar que todas las propiedades se cumplan en presencia de todas las entradas posibles.

Como se mencionó anteriormente existen distintas técnicas que entran dentro de la clasificación de técnicas de verificación formal, a continuación se describirán brevemente las tres principales vertientes de esta área.

Prueba de teoremas

Es el método más general y más poderoso de los tres, mediante su uso se pueden verificar sistemas finitos o infinitos, discretos o continuos, lo que pueda ser probado matemáticamente puede ser probado por esta técnica, en ella se especifican tanto el modelo del sistema como las propiedades a verificar en alguna lógica. El sistema se describe de forma que las acciones básicas del mismo se convierten en axiomas y las interacciones básicas en reglas de inferencia, de esta forma tenemos que todos los comportamientos del sistema pueden ser derivados a partir de la aplicación de las reglas sobre los axiomas, si todos los comportamientos pueden ser generados entonces en particular tenemos que se pueden generar aquellos que aseguran que se cumple la propiedad deseada, el proceso se denomina prueba de teoremas porque al probar que la propiedad se cumple en el sistema ésta se convierte en un teorema de esa teoría axiomática. Lamentablemente esta técnica, por ser tan general, no es automatizable y por lo tanto en el mejor de los casos el proceso de prueba requerirá la guía de un experto.

Prueba de equivalencia

La prueba de equivalencia busca verificar la validez de un sistema, probando formalmente que el modelo del mismo es equivalente al modelo de algún sistema que ya se conoce cumple con el comportamiento deseado.

El gran inconveniente con esta técnica es que se necesita conocer algún sistema que cumpla con las especificaciones deseadas; nótese que si el sistema tomado como referencia no ha sido verificado en su totalidad es posible que el

sistema verificado tenga fallas aun cuando haya pasando con éxito la verificación por prueba de equivalencia.

Comprobación de modelos

Es probablemente la técnica de verificación formal de mayor éxito en la industria, esto se debe en gran parte a que es totalmente automatizable, además mediante su uso se ha logrado verificar sistemas que, por su tamaño, hasta hace algún tiempo se consideraban imposibles de verificar.

En esta técnica el sistema es representado mediante una máquina de estados y transiciones, algunas de las representaciones utilizadas son las estructuras de Kripke (EKs) y los autómatas de Buchi generalizados etiquetados (ABGEs), por otra parte la propiedad se especifica en alguna lógica temporal, las más utilizadas son la Lógica Temporal Lineal (LTL) y la Lógica de Árbol Computable (CTL).

Una vez especificados el modelo y la propiedad se procede a realizar una búsqueda exhaustiva por todo el espacio de estados del sistema en búsqueda de algún comportamiento que invalide el modelo del sistema, en caso de que se logre encontrar dicho comportamiento, el comprobador de modelos arroja como resultado la traza que no ha cumplido la propiedad, a esta traza se le denomina contraejemplo y puede ser útil en el proceso de reparación del diseño.

Cuando la propiedad a verificar es representada en CTL la relación de transición del modelo de sistema es representada mediante Diagramas de Decisión Binarios (BDD) y el proceso de búsqueda se realiza mediante una serie de cálculos

de punto fijo sobre estos, cuando se representa la propiedad en LTL es común que el problema se convierta en el de determinar la vacuidad de un lenguaje, para hacer esto primero se niega la propiedad, después se genera una máquina de estados y transiciones que represente la negación de la propiedad, entonces crea un nuevo modelo que es la composición del modelo de sistema y del modelo de la propiedad, finalmente se verifica el lenguaje del modelo compuesto, si este es vacío, el modelo cumple la propiedad.

A continuación se explican brevemente las dos grandes variantes dentro de la comprobación formal de modelos:

- **Comprobación de modelos explícita:** en ella se construye la máquina de estados y transiciones del modelo del sistema de forma explícita, para lo cual se necesita una gran cantidad de almacenamiento físico, su principal ventaja es que los algoritmos de búsqueda son relativamente eficientes, además se pueden utilizar fácilmente en ellos estrategias de búsqueda graduales.

- **Comprobación de modelos simbólica:** evita la construcción de la máquina de estados y transiciones mediante la representación de la misma en términos de fórmulas proposicionales, las cuales poseen la capacidad de representar conjuntos de estados y transiciones [MCM92], con esto se obtiene un ahorro de espacio considerable, sin embargo los algoritmos de búsqueda utilizados en esta técnica son significativamente más complicados.

Cada una de las vertientes antes mencionadas tienen a su vez dos variantes, éstas difieren en la profundidad de la búsqueda realizada.

- **Comprobación de modelos acotada:** limita la búsqueda a contraejemplos de longitud finita menores a una cota máxima predeterminada, para ello se representan tanto el sistema como la

propiedad a verificar en una fórmula proposicional, de forma que esta es satisfacible si y sólo si existe un contraejemplo de longitud menor a la cota definida[CLK01] [BIR03] [AML05].

- **Comprobación de modelos no acotada:** en esta técnica la búsqueda de contraejemplos se realiza en el espacio completo de estados del sistema y no es restringida por ninguna cota, hay que resaltar que aunque se verifiquen estados con trayectorias infinitas esta técnica sólo se aplica en máquinas de estados finitas, por lo tanto, por más que se extienda la búsqueda forzosamente llegará el momento en que se alcance un punto fijo [BAR07] [CLK94] [PRD98].

1.3 Explosión de estados: el enemigo a vencer

A pesar de los prometedores avances obtenidos en la comprobación de modelos seguimos eventualmente encontrándonos con el añejo problema de la explosión de estados; la razón es muy simple: la naturaleza intrínseca de la comprobación de modelos es no polinomial. Tomemos como ejemplo el algoritmo de verificación de propiedades especificadas en LTL que es "lineal con respecto al tamaño del modelo" y "exponencial respecto al tamaño de la fórmula a verificar" [HTH04], a primera vista no suena del todo mal esta complejidad puesto que podemos enfocarnos en verificar propiedades de tamaño reducido, pero si analizamos más a fondo las palabras "tamaño del modelo" podremos fácilmente notar que este es exponencial respecto al número de variables, lo que implica que por cada variable que se añada al modelo, éste duplicará su tamaño y por lo tanto, también será doblemente difícil verificarlo, por otra parte si se añade una variable a la propiedad, el autómata que la modela duplicará su tamaño y esto incidirá de forma exponencial en la verificación.

De lo anterior podemos deducir que el número de variables es en realidad el factor más determinante en el proceso de verificación, puesto que incide de forma exponencial en la complejidad.

1.4 Trabajo relacionado

Muchos esfuerzos se han realizado con el fin de reducir los tiempos de verificación, la mayor parte de ellos en dos vertientes: mejora de algoritmos y utilización de abstracciones, siendo ésta última el objeto de este documento.

La aplicación de técnicas de abstracción ha permitido obtener grandes reducciones en los tiempos de verificación, a continuación se explican brevemente algunas técnicas de abstracción.

1.4.1 Refinamiento de abstracciones basado en contraejemplos

El refinamiento de abstracciones basado en contraejemplos o CEGAR [CLK00][CLK03] por sus siglas en inglés, utiliza pequeños modelos sobre aproximados, es decir, que incluyen un superconjunto de los comportamientos que el modelo original, para realizar la verificación sobre ellos.

Sin embargo, la verificación en estos modelos no es del todo confiable, si se verifica que la propiedad se cumple en todos los comportamientos del modelo sobre aproximado, entonces tenemos que por simple contención de conjuntos también deberá cumplirse en el modelo concreto. Por otra parte, es posible encontrar comportamientos en el modelo sobre aproximado que no cumplan con la propiedad y que no existan en el modelo concreto, a estos comportamientos se les denomina contraejemplos espurios y su posible existencia nos obliga a validar el

resultado de la verificación cuando éste es negativo. Si en la validación se determina que hemos encontrado un contraejemplo espurio, entonces se analiza la estructura de éste en búsqueda de información que ayude a la construcción de un nuevo modelo sobre aproximado que refine al anterior en el sentido de que no incluya el contraejemplo espurio, este proceso se repite hasta encontrar una respuesta afirmativa o un contraejemplo no espurio.

Es importante destacar que aunque en un principio CEGAR fue una técnica de abstracción, con el tiempo se ha convertido en un marco de trabajo, puesto que es posible cambiar de forma relativamente simple como se realizan la verificación, el análisis de contraejemplos, y la forma en que los modelos sobre aproximados son creados; de hecho, en nuestra investigación también hemos recurrido a CEGAR como marco de trabajo.

1.4.2 Abstracción por predicados

En esta técnica se identifican las restricciones, también llamadas predicados, que describen las partes importantes del sistema, con ellas se construye un modelo que depende de los predicados y no de las variables en sí [CLK04], esta técnica resulta bastante útil cuando el número de restricciones del problema es mucho menor que el número de variables.

1.4.3 Reducción al cono de influencia

La reducción al cono de influencia es una técnica de abstracción exacta dependiente de la propiedad, por esta razón un modelo abstracto construido

mediante su uso conservará todos los comportamientos del modelo original que son relevantes a la propiedad [CLK01].

El principio de la reducción al cono de influencia es simple: eliminar los hechos que no influyan en el comportamiento de las variables de la propiedad, es posible implementar la reducción al cono de influencia mediante la eliminación de variables o la eliminación de predicados.

1.4.4 Abstracción por explotación de simetrías

Se basa en el hecho de que muchos sistemas tienen componentes que son estructuralmente idénticos o casi idénticos, por ello, se procede a identificar grupos de estados que presenten comportamientos similares para asociarlos en clases de equivalencia, una vez identificadas, se crea un nuevo modelo en el que se remplazan los miembros de cada clase de equivalencia por un único representante [ALL93], la efectividad de esta técnica depende de qué tanta simetría exhiba el modelo a verificar.

1.5 Objetivo de esta tesis

Con este trabajo se persiguen dos objetivos principales:

- Utilizar técnicas de abstracción y refinamientos basados en heurísticas a fin de permitir la verificación simbólica de modelos más grandes que los verificados actualmente.

- Reducir los tiempos de verificación en la comprobación simbólica de modelos usando propiedades en lógica temporal lineal.

En general, es sumamente difícil que una sola técnica pueda alcanzar ambos objetivos, por esta razón en el presente estudio se han combinado técnicas que persiguen reducir el tiempo total con técnicas que permiten verificar modelos de mayor tamaño.

1.6 Estructura de este documento

En el Capítulo 2 Sustento teórico presentamos estructuras de representación y la teoría sobre la cual apoyamos esta disertación. En el Capítulo 3 Herramientas actuales presentamos dos de las herramientas de verificación formal que mayor éxito han tenido en los últimos años: el comprobador de modelos simbólico NuSMV y el SAT Solver MiniSAT, también presentamos algunas de las técnicas y estructuras de representación utilizadas en estas herramientas. En el Capítulo 4 Comprobación de propiedades LTL usando técnicas de abstracción presentamos las técnicas de abstracción-refinamiento utilizadas en esta investigación. En el Capítulo 5 Trabajo experimental presentamos los resultados obtenidos al implementar las técnicas aquí descritas y someter dicha implementación a una batería de benchmarks. En el Capítulo 6 Conclusiones y trabajo futuro presentamos las conclusiones a las que hemos llegado el desarrollar esta investigación, además presentamos algunas ideas de cómo mejorar los resultados obtenidos, estas ideas no pudieron ser desarrolladas por limitaciones de tiempo. Por último e el Apéndice A presentamos una breve descripción de la herramienta desarrollada.

Capítulo 2
Sustento teórico

En este capítulo se incluyen como fundamento teórico de este trabajo un repaso de las lógicas proposicional y temporal lineal que nos serán de utilidad en la especificación de las propiedades que se desea cumpla un sistema, en este sentido, realizamos una caracterización de las propiedades que podemos verificar mediante el uso de los algoritmos de comprobación de modelos acotada y no acotada, para después terminar con la descripción de los mismos.

2.1 Lógica proposicional

En esta lógica se utilizan expresiones denominadas proposiciones atómicas o simplemente proposiciones, éstas se evalúan como verdadero o falso.

Es posible dar a las proposiciones un significado asociándolas con sentencias declarativas. No es necesario que las sentencias a las que se asocian las

proposiciones sean verdaderas, lo importante es que no tomen valores intermedios, sólo pueden ser verdaderas o falsas.

Es poco común que al expresar una idea ésta sea una sola sentencia declarativa, más frecuentemente se expresa con base en varias sentencias asociadas; también es posible realizar esto en la lógica proposicional mediante el uso de los denominados conectivos lógicos.

2.1.1 Fórmulas proposicionales

A las expresiones obtenidas mediante el uso de proposiciones y conectivos lógicos se les llama fórmulas proposicionales, se definen formalmente con la sintaxis y la semántica presentadas a continuación.

2.1.2 Sintaxis

Sea AP el conjunto de las proposiciones atómicas.

- Toda proposición atómica en AP es una fórmula proposicional.
- Si φ es una fórmula proposicional ($\neg \varphi$) también lo es.
- Si φ y ψ son fórmulas proposicionales, también lo son ($\varphi \vee \psi$) y ($\varphi \wedge \psi$).

2.1.3 Semántica

Al igual que las proposiciones atómicas, las fórmulas proposicionales son expresiones que se evalúan como verdaderas o falsas, para determinar el valor de verdad de una fórmula proposicional es necesario conocer los valores de verdad de las proposiciones en ella.

Definición 2-1 Constantes $\top$, $\bot$ y el Dominio $\mathbb{B}$

La constante $\top$ hace referencia al valor de verdad VERDADERO y $\bot$ referencia a FALSO. $\mathbb{B}:=\{ \top, \bot \}$ es el conjunto de valores de verdad para las fórmulas proposicionales.

Definición 2-2 Vocabulario de una fórmula proposicional

Sea φ una fórmula proposicional. El conjunto VOC(φ) es la colección de proposiciones atómicas que conforman φ y se le denomina vocabulario de φ.

Definición 2-3 Valuación

Sea φ una fórmula proposicional. Una valuación para φ es una función que asigna cada fórmula en φ un valor en $\mathbb{B}$.

Para poder dar interpretación a las fórmulas proposicionales y no sólo a las proposiciones que las conforman, es necesario conocer la interpretación de los conectivos lógicos.

Definición 2-4 Operador ¬

El operador ¬ niega el valor de verdad de la fórmula a la que precede, es decir, que si ψ tiene como valor de verdad $\bot$ entonces $\neg\psi$ tendrá como valor de verdad $\top$ y viceversa.

Definición 2-5 Operador ∨

El operador de disyunción ∨ asocia dos fórmulas proposicionales φ y ψ en una sola, de tal forma que φ ∨ ψ será falsa si y sólo si φ es falsa y ψ también es falsa. La fórmula φ ∨ ψ suele leerse "φ o ψ"

Definición 2-6 Operador ∧

El operador de conjunción ∧ asocia dos fórmulas proposicionales φ y ψ en una sola, de tal forma que φ ∧ ψ será verdadera si y sólo si φ es verdadera y ψ también es verdadera. La fórmula φ ∧ ψ suele leerse "φ y ψ"

2.1.4 Equivalencia de fórmulas proposicionales

Es posible que dos expresiones aparentemente distintas expresen la misma idea, cuando éste es el caso se dice que tenemos expresiones semánticamente equivalentes.

Definición 2-7 Equivalencia Proposicional

Sean φ y ψ dos fórmulas proposicionales. Se dice que φ y ψ son *equivalentes*, y se expresa como φ ≡ ψ, cuando para cualquier asignación de valores de verdad a las proposiciones en AP, φ será verdadera si y sólo si ψ lo es.

La equivalencia semántica nos permite definir operadores compuestos, definiremos dos con el fin de expresar de manera compacta dos tipos comunes de expresiones.

Definición 2-8 Operador →

El operador de implicación → nos permite asociar dos fórmulas proposicionales φ y ψ en una sola φ → ψ equivalente a ¬ φ ∨ ψ. La fórmula φ → ψ suele leerse "si φ entonces ψ".

Definición 2-9 Operador ↔

El operador de doble implicación ↔ nos permite asociar dos fórmulas proposicionales φ y ψ en una sola φ ↔ ψ equivalente a (¬φ ∨ ψ) ∧ (φ ∨ ¬ψ). La fórmula φ ↔ ψ suele leerse "φ si y sólo si ψ"

2.1.5 Satisfacibilidad

El problema de satisfacibilidad proposicional también llamado problema SAT consiste en determinar si dada una fórmula proposicional existe al menos una valuación a las variables presentes en ella que hace que la fórmula entera sea verdadera.

Definición 2-10 Satisfacibilidad

Sea φ una fórmula proposicional, se dice que φ es *satisfacible* si y sólo si existe al menos una valuación que hace que φ tome como valor de verdad T. Si no existe

dicha valuación se dice que φ es una fórmula *insatisfacible* o que φ es una *contradicción*.

Existe además un problema relevante que se relaciona a la satisfacibilidad proposicional: la validez, este cosiste en determinar si una fórmula es lógicamente equivalente a ⊤.

Definición 2-11 Validez de una fórmula proposicional

Sean φ una fórmula proposicional. Se dice que φ es *válida* o *tautológica*, y se expresa como ⊨φ, cuando para cualquier asignación de valores de verdad a las proposiciones en AP, φ toma como valor de verdad ⊤.

La validez y la satisfacibilidad proposicionales son problemas duales. Así tenemos que una fórmula proposicional es válida si y sólo si su negación es insatisfacible.

2.1.6 Forma normal conjuntiva

Una forma normal es un conjunto de normas para la representación de fórmulas proposicionales. Una de las formas normales más útiles para la representación de fórmulas proposicionales es la *Forma Normal Conjuntiva* (*CNF*).

Definición 2-12 Literal

Una *literal* es una proposición atómica (*p*) o la negación de una proposición atómica (¬*p*).

Definición 2-13 Cláusula

Una fórmula proposicional φ es una *cláusula* cuando sólo esta formada por literales en disyunción. A la cláusula que no contiene ninguna literal se le denomina *cláusula vacía* y su valor de verdad es ⊥. Se dice que una cláusula es *unitaria* cuando solamente contiene una literal.

Definición 2-14 Forma normal conjuntiva

Una fórmula proposicional φ esta en *FNC* cuando sólo esta formada por cláusulas en conjunción.

Para mayores referencias acerca de la lógica proposicional vea [BAR07], [HTH04].

2.2 Lógica temporal lineal

La lógica temporal lineal o *LTL* es una lógica que extiende la lógica proposicional mediante la introducción de operadores modales de tiempo futuro, estos operadores permiten expresar cambios en los valores de verdad de las proposiciones a través del tiempo discreto [BAR07] [HTH04].

2.2.1 Sintaxis

Las fórmulas en LTL se definen mediante la siguiente recurrencia:

- Toda proposición atómica en AP es una fórmula LTL.
- Si φ es una fórmula LTL (¬ φ) también lo es.
- Si φ y ψ son fórmulas LTL, también lo son (φ ∨ ψ), (φ ∧ ψ) y (φ U ψ).
- Si φ es una fórmula de LTL también lo son (X φ), (G φ) y (F φ).

2.2.2 Semántica

Las fórmulas en LTL nos permiten expresar el comportamiento de un sistema a través del tiempo, estos sistemas deben a su vez poder describirse formalmente, para esto utilizamos un tipo especial de autómata finito no determinista denominado estructura de Kripke o EK [BRW87].

Definición 2-15 Estructura de Kripke

Una estructura de Kripke K sobre AP es una cuadrupla del tipo:

$$K := (S, I, R, L)$$

donde:

- S es un conjunto finito de estados.
- I es un conjunto de estados iniciales y es subconjunto de S.
- R es una relación de transición subconjunto de SxS que cumple que para todo $s \in S$, existe $s' \in S$ tal que $(s, s') \in R$.
- L es una función de etiquetado de $E \rightarrow 2^{AP}$ que asocia cada estado $s \in S$ con un conjunto L(s) de proposiciones atómicas en AP

La validez de una fórmula LTL se evalúa en una secuencia infinita de estados que representan instantes de tiempo, a estas secuencias se les denomina *cómputos* o *trayectorias*.

Definición 2-16 Cómputo

Sea K := (S, I, R, L) una estructura de Kripke. Un *cómputo* π es una función $\pi:\mathbb{N}_0 \rightarrow S$. Esta función representa la secuencia infinita de estados $\pi=<s^0,s^1,s^2,...>$, en la cual para todo $i \in \mathbb{N}_0$ se cumple que $(s^i, s^{i+1}) \in R$. La función π describe una evolución el

sistema a través del tiempo. Para referirnos al *j*-ésimo estado en π escribiremos π(*j*).

Las EK representan de manera compacta conjuntos posiblemente infinitos de cómputos, los comportamientos de un sistema.

Definición 2-17 Semántica de las fórmulas LTL

Dados un cómputo π y un número $i \in \mathbb{N}_0$. Denotamos con la expresión π, $i \models \varphi$ que la fórmula LTL φ se cumple en π a partir del i-ésimo estado. Definimos ⊨ como sigue:

- $\pi, i \models \top$
- $\pi, i \not\models \bot$
- $\pi, i \models p$ *si y sólo si* $p \in L(\pi(i))$
- $\pi, i \models \neg \varphi$ *si y sólo si* $\pi, i \not\models \varphi$
- $\pi, i \models \varphi \wedge \psi$ *si y sólo si* $\pi, i \models \varphi$ *y* $\pi, i \models \psi$
- $\pi, i \models \varphi \vee \psi$ *si y sólo si* $\pi, i \models \varphi$ *o* $\pi, i \models \psi$
- $\pi, i \models X \varphi$ *si y sólo si* $\pi, i+1 \models \varphi$
- $\pi, i \models G \varphi$ *si y sólo si* $\forall j \geq i$ *tenemos que* $\pi, j \models \varphi$
- $\pi, i \models E \varphi$ *si y sólo si* $\exists j \geq i$ *tal que* $\pi, j \models \varphi$
- $\pi, i \models \varphi U \psi$ *si y sólo si* $\exists k \geq i$ *tal que* $\pi, k \models \psi$ *y* $\forall j$ *tal que* $k > j \geq i$ *tenemos que* $\pi, j \models \varphi$

Al igual que los operadores proposicionales los operadores X, G, F y U de la LTL tienen interpretaciones simples. *X* φ indica que la fórmula LTL φ se cumple en el siguiente instante de tiempo y se lee "en el siguiente φ" *G* φ indica que φ se cumple en todo instante de tiempo a partir del estado actual y se lee "siempre φ" F φ asegura que en algún instante futuro se cumplirá φ y se lee "eventualmente φ". φ U ψ indica que φ se cumplirá en todo instante previo a que ψ se cumpla, φ U ψ se lee "φ hasta que ψ"

Definición 2-18 Modelo

Dados un cómputo π y una fórmula LTL φ. Decimos que π es un *modelo para φ* si y sólo si π, *0* ⊨ φ.

Dos fórmulas LTL φ y ψ son equivalentes cuando para cualquier cómputo π se cumple que π es un modelo para φ si y solo si π es un modelo para ψ.

Definición 2-19 Validez de una fórmula LTL en una EK

Dados una EK *K := (S, I, R, L)* y una fórmula LTL φ, decimos que φ es *valida en K* y lo denotamos con *K* ⊨ φ, si y sólo si todos los cómputos de *K* que inician en un estado perteneciente a *I* son modelos para φ.

Definición 2-20 Vocabulario de una estructura de Kripke

Sea *K := (S, I, R, L)* una estructura de Kripke definida sobre *AP*, denotamos por VOC(K) al conjunto de proposiciones atómicas resultante de la unión de las etiquetas de todos los estados de S. Al conjunto VOC(K) le denominamos vocabulario de K o vocabulario del modelo y es un subconjunto de *AP*.

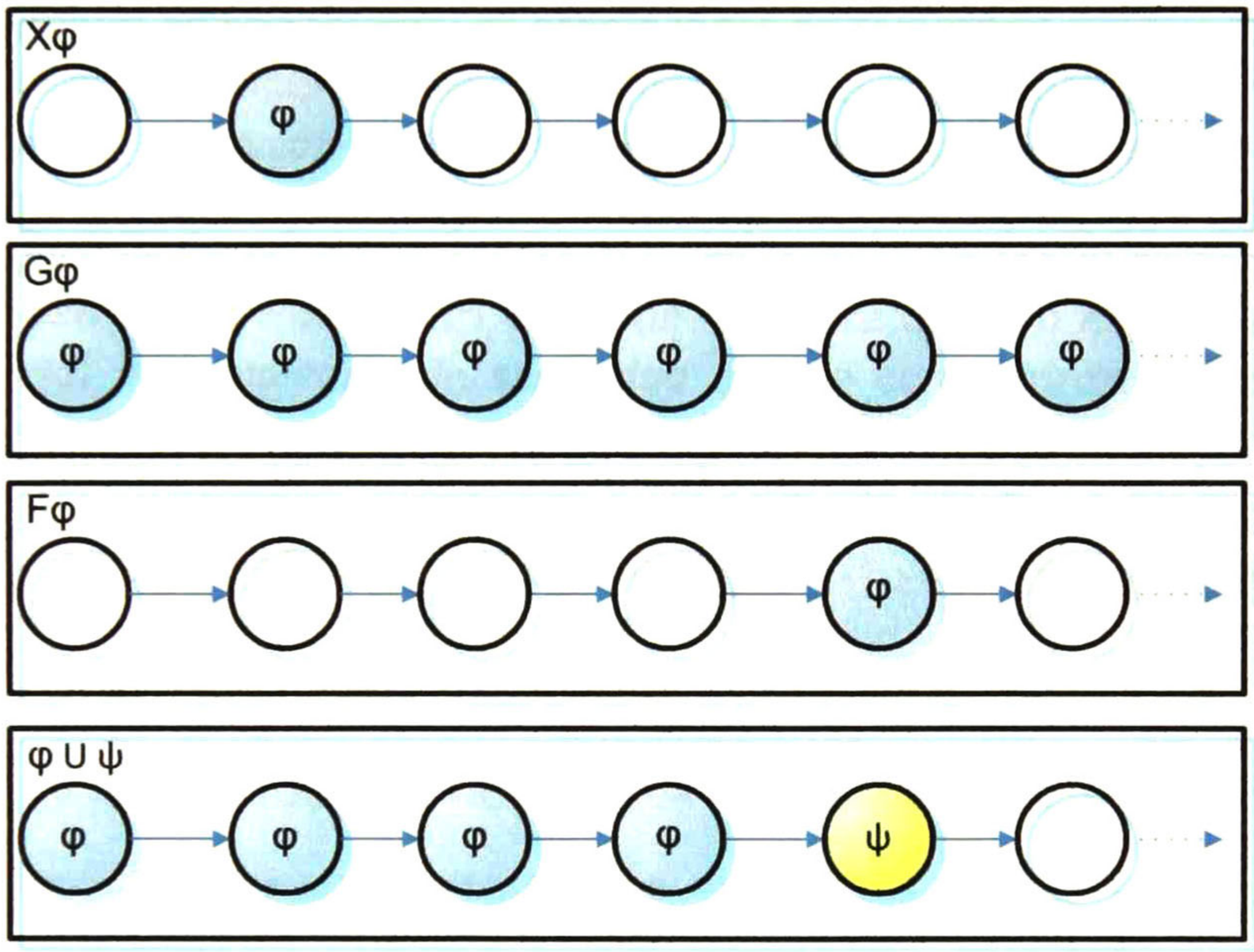


Figura 2-1 Interpretación de los operadores temporales X, G, E y U.

2.3 Caracterización de las fórmulas LTL

La semántica de la lógica temporal lineal nos permite expresar formalmente comportamientos o propiedades que deben o no presentarse en un sistema a través del tiempo, por esta razón, llamaremos *propiedades LTL* o simplemente *propiedades* a las fórmulas LTL; es conveniente saber qué clase de propiedades

pueden describirse usando LTL, para esto las caracterizaremos en tres tipos relevantes: *propiedades de vivacidad*, *propiedades de seguridad* e *invariantes*.

- ***Propiedad de vivacidad:*** *Mediante una propiedad de vivacidad podemos expresar que "eventualmente pasará algo bueno". Para demostrar que no se cumple una propiedad de vivacidad, es necesario recurrir a cómputos de longitud infinita*

- ***Propiedad de seguridad:*** *Las propiedades de seguridad expresan que "nunca ocurrirá algo malo". Para demostrar que no se cumple una propiedad de seguridad basta con encontrar el prefijo finito de un cómputo que contradiga la propiedad.*

- ***Invariante:*** *Las invariantes expresan "siempre ocurrirá que", y son un tipo especial de propiedad de seguridad que se caracterizan por el siguiente patrón: Gψ, donde ψ es una fórmula proposicional.*

2.4 Comprobación de modelos usando propiedades en LTL

El objetivo de la comprobación de modelos es verificar que un modelo formal de sistema cumpla con algunas propiedades expresadas en lógica temporal. Si se determina que el modelo no cumple con alguna propiedad, es deseable que el proceso de comprobación proporcione una trayectoria de ejecución que muestre la falla, a este tipo de trayectorias se les denomina *contraejemplos*.

Las propiedades expresadas en LTL denotan comportamientos de *trayectoria,* esto significa que para verificar propiedades LTL tenemos que asegurarnos que se cumplen en toda la trayectoria infinita de ejecución partiendo de cierto estado. Por otra parte, tenemos que las estructuras de Kripke nos permiten modelar sistemas con una cantidad posiblemente infinita de comportamientos distintos. Por tanto, para verificar que una estructura de Kripke

cumple cierta propiedad es necesario verificar que todos los posibles cómputos cumplen la propiedad. A continuación formalizaremos esta noción y expondremos un algoritmo automatizable capaz de determinar si un sistema modelado como una estructura de Kripke cumple o no con determinada propiedad.

Definición 2-21 Contraejemplo

Sean $K := (S, I, R, L)$ una estructura de Kripke, φ una fórmula temporal y π un un cómputo de K *que inicia en un estado que pertenece a* I. Se dice que π es un *contraejemplo* a φ en K si y sólo si $\pi, 0 \not\models \varphi$.

2.4.1 Algoritmo LTL de comprobación de modelos

Sean φ una fórmula en LTL y K una estructura de Kripke.

1. Construya un autómata $T_{-\varphi}$ para $-\varphi$. $T_{-\varphi}$ representa todos aquellos comportamientos cuya existencia demuestra que φ no se cumple. Al autómata construido con estas características le llamaremos Tableau.

2. Construya un nuevo autómata PS calculando el producto de $T_{-\varphi}$ y K.

3. Calcule el lenguaje de PS, si existe al menos una palabra de longitud infinita que pertenezca al lenguaje de PS, entonces φ no se cumple en K y esa palabra es un contraejemplo, si el lenguaje de PS es vacío podemos afirmar que K cumple la propiedad.

2.5 Comprobación de modelos acotada usando propiedades en LTL

En esta técnica se fija una cota, digamos k, y se procede a analizar todos los posibles comportamientos que el sistema pueda adoptar en k instantes de tiempo

en búsqueda de contraejemplos de longitud k. Es común la aplicación incremental de esta técnica con el fin de asegurar que no existen contraejemplos de longitud menor o igual a la cota establecida.

La comprobación de modelos acotada es una técnica sólida, sin embargo, en general no es completa. Aún aplicándola de forma incremental, sólo cuando k es mayor o igual al diámetro de recurrencia del modelo, es posible asegurar que no existen contraejemplos de longitud mayor k. Es inusual que se conozca el diámetro de recurrencia del modelo y en la mayoría de los casos no es práctico calcularlo, por estos motivos la comprobación acotada de modelos es más una técnica de refutación que de validación.

2.5.1 Algoritmo LTL de comprobación de modelos acotada

Sean φ una fórmula en LTL, M la estructura de Kripke y k una cota mayor o igual a cero.

1. Construya bajo el siguiente criterio una fórmula proposicional $[\![M]\!]_k$ correspondiente al desdoblamiento desde cero hasta k de la relación de transición T:

$$[\![M]\!]_k := I(S_0) \wedge \bigwedge_{i=0}^{k-1} T(S_i + S_{i+1})$$

donde:

- $I(S_0)$ corresponde a la función característica del conjunto de estados iniciales
- $T(S_i + S_{i+1})$ corresponde a la función característica de la relación de transición.

2. Construya la fórmula $[\![\varphi]\!]_k$, esta fórmula sólo deberá tomar valor $\top$ cuando φ se cumpla en todos los comportamientos de longitud k en M.

3. Forme una fórmula $[\![M, \varphi]\!]_k$ con la conjunción de $[\![M]\!]_k$ y $[\![\varphi]\!]_k$.
4. Determine si existe al menos una valuación que haga que $[\![M, \varphi]\!]_k$ tome como valor de verdad T; en caso de existir, dicha valuación corresponderá al prefijo de un contraejemplo a φ, si no existe podemos afirmar la no existencia de contraejemplos de longitud k para φ.

2.6 Abstracción

Intuitivamente, una abstracción es una representación que ignora u oculta determinados detalles de un modelo con el fin de enfatizar los puntos de interés en el mismo. En este estudio diferenciaremos las abstracciones en base a la forma en que soslayan información irrelevante.

2.6.1 Abstracciones exactas

Se dice que una abstracción es exacta cuando esta exhibe exactamente los mismos comportamientos que el modelo concreto. Cuando el modelo concreto es una estructura de Kripke M es posible crear una abstracción exacta M' con el siguiente procedimiento:

1. Particione el espacio de estados de M en clases de equivalencia; dos estados e y f serán equivalentes si y sólo si cumplen las siguientes condiciones:
   a. Sus etiquetados son exactamente iguales.
   b. Para todo sucesor de e existe un sucesor de f equivalente.
2. Cree un nuevo modelo M'. Este modelo solo deberá contener un solo estado de cada clase de equivalencia, esto se logra colapsando todos lo estados equivalentes en uno solo.

Debido a que las abstracciones exactas tienen exactamente los mismos comportamientos que el modelo concreto, es posible afirmar que para toda propiedad φ de la LTL $M \models \varphi$ si y sólo si M' cumple φ.

2.6.2 Sobre aproximaciones

Se dice que una abstracción es una *sobre aproximación* cuando esta exhibe un conjunto de comportamientos que es superconjunto de los comportamientos del modelo concreto.

Agregar más comportamientos a un modelo puede convertir en equivalentes algunos estados que antes no lo eran, logrando así reducir el número de clases de equivalencia en el modelo.

Abstracción existencial

La abstracción existencial es una técnica para la generación de sobre aproximaciones. Cuando el modelo concreto es una estructura de Kripke M:= (S, I, R, L) es posible crear una sobre aproximación M':= (S', I', R', L') con el siguiente procedimiento:

1. Defina una *función de abstracción* $h': S \rightarrow S'$
2. Calcule M' como sigue:
   a. $S' := \{s' \mid \exists s.\ s \in S \wedge h'(s) = s'\}$
   b. $I' := \{s' \mid \exists s.\ h'(s)=s' \wedge s \in I\}$
   c. $R' := \{(e', g') \mid \exists e\ \exists g.\ h'(e)=e' \wedge h'(g)=g' \wedge (e, g) \in R\}$
   d. $L'(s') := \cap_{h'(s)=s'} L(s)$

Eliminación de variables mediante abstracción existencial

Cuando se ve el etiquetado de cada estado como una palabra es posible crear abstracciones existenciales definiendo la función h' como la función "prefijo" tenemos así que todos los estados concretos que compartan el mismo prefijo deberán corresponder al estado abstracto etiquetado con dicho prefijo, nótese que el orden de los etiquetados puede ser cambiado en cualquier momento, esto nos permite utilizar este método para eliminar cualquier conjunto de variables.

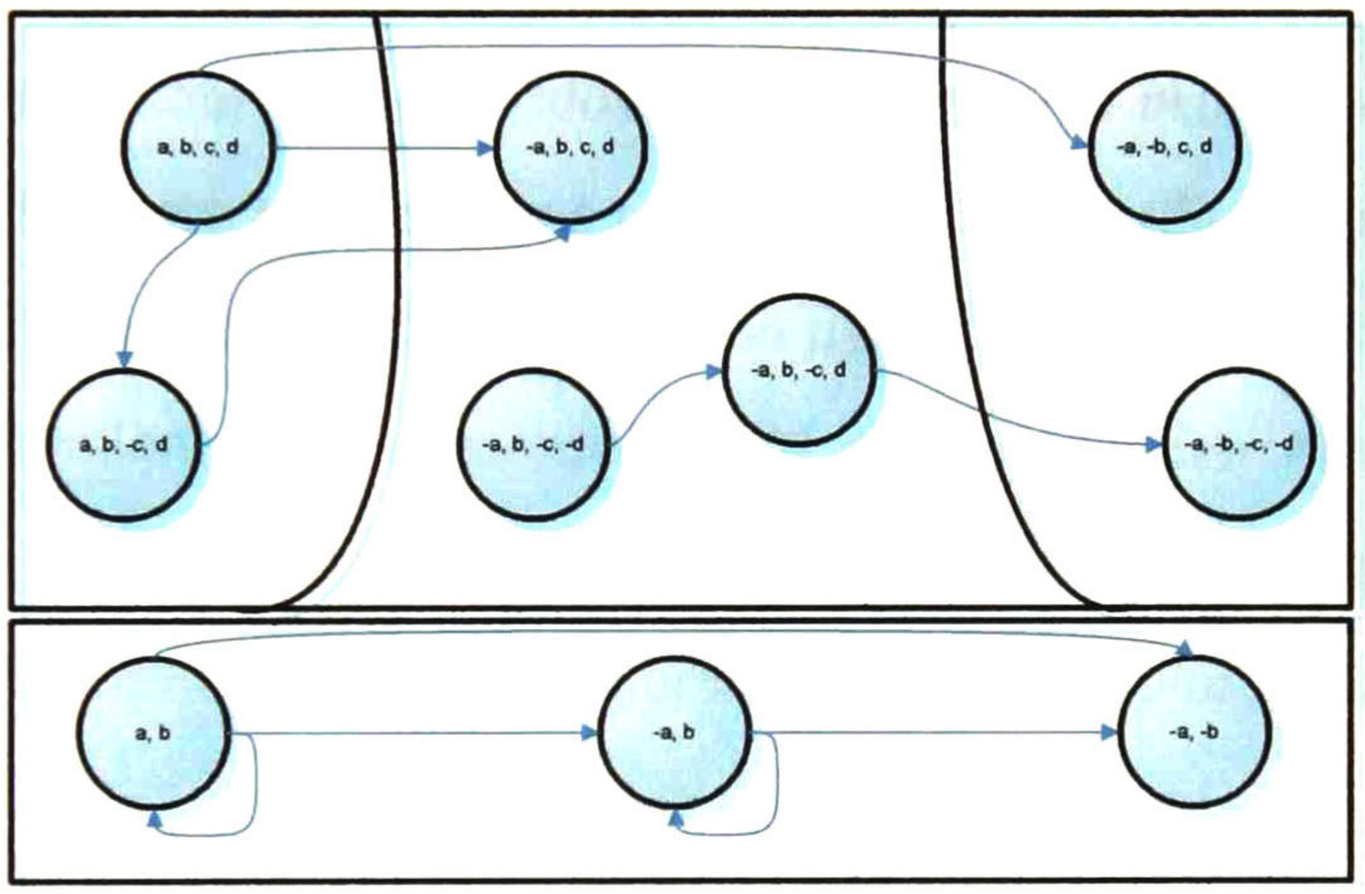


Figura 2-2 Abstracción existencial de variables.

La Figura 2-2 presenta un ejemplo de la abstracción existencial de variables. En el marco superior se muestra el modelo original, en el marco inferior se muestra el modelo después de abstraer existencialmente las variables c y d.

Teorema 2-1 Preservación de comportamientos en sobre aproximaciones

Dados un modelo M y fórmula φ de la LTL, se cumple para cualquier abstracción *sobre aproximada* M' que M'⊨φ → M⊨φ. Sin embargo no es posible afirmar la implicación inversa [CLK03].

El teorema de la preservación de comportamientos en sobre aproximaciones nos brinda las bases para la utilización de estas en la comprobación de modelos. Sabiendo que M'⊨φ → M⊨φ podemos intentar verificar las propiedades en pequeños modelos sobre aproximados. Sin embargo si la propiedad no se cumple en el modelo abstracto M', debemos tener cuidado de verificar que realmente no se cumpla en el modelo concreto M; esto se debe a que como hemos introducido en M' comportamientos no presentes en M podemos haber encontrado, como contraejemplo, precisamente uno de estos comportamientos adicionales; a este tipo de contraejemplos se les denomina *contraejemplos espurios* o falsos negativos.

Capítulo 3
Herramientas actuales

Gran relevancia en el creciente desempeño de la verificación formal tienen las implementaciones, en ellas se conjuntan los mejores algoritmos y las estructuras de datos más adecuadas a cada problema, cualquier implementación exitosa reúne ambas características.

En este capítulo expondremos una de las herramientas de verificación formal más destacadas: el comprobador de modelos NuSMV. También estudiaremos un tipo de herramienta que ha tenido un gran despunte en los últimos años: los SAT Solvers, específicamente hablaremos de la implementación realizada en el SAT Solver MiniSAT.

En el marco de NuSMV repasaremos brevemente su implementación del comprobador no acotado para propiedades LTL y los diagramas de decisión

binarios, por ser éstos las estructuras de datos que sustentan dicha implementación. Respecto a los SAT Solvers estudiaremos el algoritmo DPLL [MSK01] con aprendizaje guiado por análisis de conflictos [EEN03], se hará énfasis en las heurísticas que se utilizan en MiniSAT para mejorar el rendimiento de dicho algoritmo.

3.1 NuSMV

NuSMV es un comprobador de modelos simbólico basado en SMV que permite verificar propiedades escritas en Lógica Computacional Ramificada (CTL) y en Lógica Temporal Lineal (LTL) [CMT02] mediante comprobación de modelos simbólica basada en BDDs, para lo que utiliza la librería CUDD, además permite realizar comprobación de modelos acotada de propiedades LTL, basada en circuitos booleanos reducidos, para esto se auxilia de SAT Solvers del estado del arte como SIM, MiniSAT y zChaff.

3.1.1 Comprobación de propiedades LTL en NuSMV

Un solo algoritmo de comprobación de modelos simbólica con BDDs es implementado en NuSMV: el algoritmo de comprobación de propiedades CTL. Sin embargo, este ha sido extendido para soportar condiciones de justicia.

El último paso del algoritmo descrito en la sección 2.4.1 corresponde a la búsqueda de una palabra de longitud infinita en el autómata formado mediante el producto del modelo (K) y el Tableau de la negación de la propiedad ($T_{\neg\varphi}$). NuSMV realiza esta búsqueda usando el algoritmo de comprobación CTL con condiciones de justicia. Sin embargo es necesario añadir dos pasos adicionales a la verificación:

la generación de condiciones de justicia durante la creación de $T_{\neg\varphi}$ y la verificación adicional de que las palabras de longitud infinita en $KxT_{\neg\varphi}$, cumplan con dichas condiciones de justicia.

Las estructuras de datos usadas en la verificación CTL con condiciones de justicia en NuSMV son los diagramas de decisión binarios, por este motivo a continuación describiremos sus características principales y algunas operaciones sobre ellos que nos son de importancia particular.

3.1.2 Diagramas de decisión binarios

Los diagramas de decisión binarios o BDD (por sus siglas en inglés) son estructuras de datos que nos permiten representar de forma eficiente y compacta fórmulas proposicionales.

3.1.3 Estructura de un BDD

Los diagramas de decisión binarios son estructuras arbóreas cuyos nodos no terminales se etiquetan con una proposición atómica y los terminales se etiquetan con $\top$ o $\bot$. En los BDDs las aristas se utilizan para representar las diferentes valuaciones que puede tener una proposición, es por ello que utilizamos dos tipos de ellas: las aristas afirmadas, que se representan con una flecha sólida, y las aristas negadas, que se representan con una flecha punteada, cada nodo no terminal es el origen de dos aristas, una afirmada y una negada.

Cuando utilizamos BDDs al igual que con fórmulas proposicionales podemos aplicar algunas reglas de reducción, a continuación se enumeran las tres principales:

1. Fusionar hojas equivalentes: un BDD puede tener muchos nodos terminales etiquetados con $\bot$ y muchos etiquetados con $\top$, sin embargo, sólo son necesarios uno de cada tipo; para aplicar esta reducción creamos un nuevo nodo $\bot$ y redirigimos hacia él todas las aristas que terminan en un nodo $\bot$, hacemos lo mismo con los nodos $\top$.

2. Eliminar pruebas redundantes: supongamos dos nodos A y B si las dos aristas que salen de A inciden en B entonces el nodo A es innecesario y podemos eliminarlo, para hacerlos redirigimos a B todas las aristas que inciden en A, entonces procedemos a eliminar tanto el nodo A como las aristas que salen de él.

3. Eliminar nodos no terminales redundantes: supongamos dos nodos C y D ambos etiquetados con la misma proposición, supongamos además que C y D son raíces de sub-BDDs estructuralmente idénticos entonces podemos eliminar uno de ellos, supongamos C, con todo y su descendencia, para ello redirigimos a D todas las aristas incidentes en C y procedemos a borrar C y todo su sub-BDD.

Las tres reglas de reducción antes mencionadas son de vital importancia en el uso de los BDDs, ya que ellas les dan una de sus principales fortalezas: el ser una estructura compacta. Nótese que al aplicar estas reglas de simplificación el árbol se convierte en un grafo dirigido acíclico. Otra característica de los BDDs es que facilitan grandemente algunas de las operaciones que en otras estructuras de datos son muy difíciles de realizar, por ejemplo, suponga que se desea obtener el complemento de la función representada por un BDD, calcularla es tan sencillo como redirigir a $\top$ las aristas incidentes en $\bot$ y a $\bot$ las incidentes en $\top$. Este mismo

problema resulta intratable cuando usamos formas normales conjuntiva o disyuntiva. Otro problema NP cuya solución se obtiene trivialmente con el uso de BDDs, es el de satisfacibilidad: para saber si la función representada por un BDD es satisfacible basta con saber si existe al menos una arista incidente en el nodo ⊤.

A pesar de las ventajas antes mencionadas, es importante solucionar algunos problemas antes de utilizar los BDDs en problemas realmente grandes, uno de ellos es que es muy difícil probar la equivalencia entre dos BDDs, puesto que nada restringe el número y orden de aparición de las proposiciones en cada trayecto. Para solucionar este problema a continuación definiremos un tipo especial de BDDs.

Definición 3-1 Diagrama de decisión binario ordenado (ROBDD)

Sea L una lista ordenada de proposiciones que no contiene elementos duplicados y sea B un BDD que representa una función proposicional cuyo vocabulario es un subconjunto de las proposiciones en L.

- Ordenado: decimos que B tiene el orden L si para cualquier par de proposiciones x_i y x_j se cumple que si x_i precede a x_j en la misma trayectoria de B entonces x_i precede a x_j en L.
- Enraizado: se considera que un BDD está enraizado si el nodo origen de todas las trayectorias en él es el mismo.

Teorema 3-1 Forma canónica en ROBDDs

El ROBDD que representa la función *f* es único después de aplicar las reglas de reducción. En otras palabras si tenemos dos ROBDDs reducidos que representan *f* y que comparten el mismo orden ambos tendrán estructuras idénticas.

El uso de ROBDDs reduce el problema de prueba de equivalencia antes mencionado a probar que los BDDs involucrados tienen estructura idéntica.

Un factor a considerar cuando se usan ROBDDs es que el orden de las proposiciones influye directamente en el tamaño del BDD y determinar el mejor orden es un problema NP, sin embargo, existen numerosas heurísticas que nos ayudan a saldar este punto.

A partir de este momento nos referiremos a los ROBDD's simplemente como BDDs.

3.1.4 Operaciones sobre BDDs

Existe un gran conjunto de operaciones que podemos realizar sobre los BDDs, sin embargo, restringiremos nuestra atención en aquellas que son trascendentes para nuestros propósitos.

Restrict(F, x, VAL).

Esta operación recibe como parámetros un BDD F que representa la función *f*, una proposición x y un valor VAL en el dominio $\mathbb{B}$. Los tres operandos antes mencionados se usan para calcular el BDD de la función *f*[VAL/x]. La implementación de esta operación es bastante sencilla, suponga que se desea calcular *f*[$\top$ /x], para hacerlo basta redirigir las aristas incidentes en cada nodo n etiquetado con x hacia el nodo apuntado por la arista afirmada que sale de n, hecho esto se elimina n y las aristas que de él salen, si VAL fuera $\perp$ las aristas se redirigirían al nodo apuntado por la arista negada.

ITE(F, G, H) (If-Then-Else).

Se reciben tres BDDs F, G y H que respectivamente representan las funciones *f*, *g*, *h*. Esta operación también es simple: si *f* =⊤ entonces el resultado es *g* en cualquier otro caso el resultado es *h*.

NEG(F)

Dado un BDD F que representa la función *f*, NEG(F) se calcula intercambiando las aristas incidentes en los nodos ⊥ y ⊤. El BDD resultante representa la función *–f*.

Estas operaciones constituyen un conjunto suficiente, con ellas es posible realizar todo el conjunto de operaciones de la lógica booleana. La última operación que revisaremos es la más importante para nuestros fines ya que nos permitirá abstraer existencialmente proposiciones en un BDD.

Exist(F, x).

Dados como parámetros un BDD F que representa la función *f* y una proposición x, Exist se define como $\exists x.f$ (F, x):= $f[\top/x] \vee f[\bot/x]$. La operación ∨ se define como A ∨ B:= NEG(ITE (NEG(A), NEG(B), ⊥)).

La siguiente figura ilustra la operación Exist, observe que las incidencias de la proposición resaltada desaparecen en el BDD, esto sucede porque se han supuesto para ésta todos los valores del dominio $\mathbb{B}$ de forma tal que no importe más cuál es su valuación.

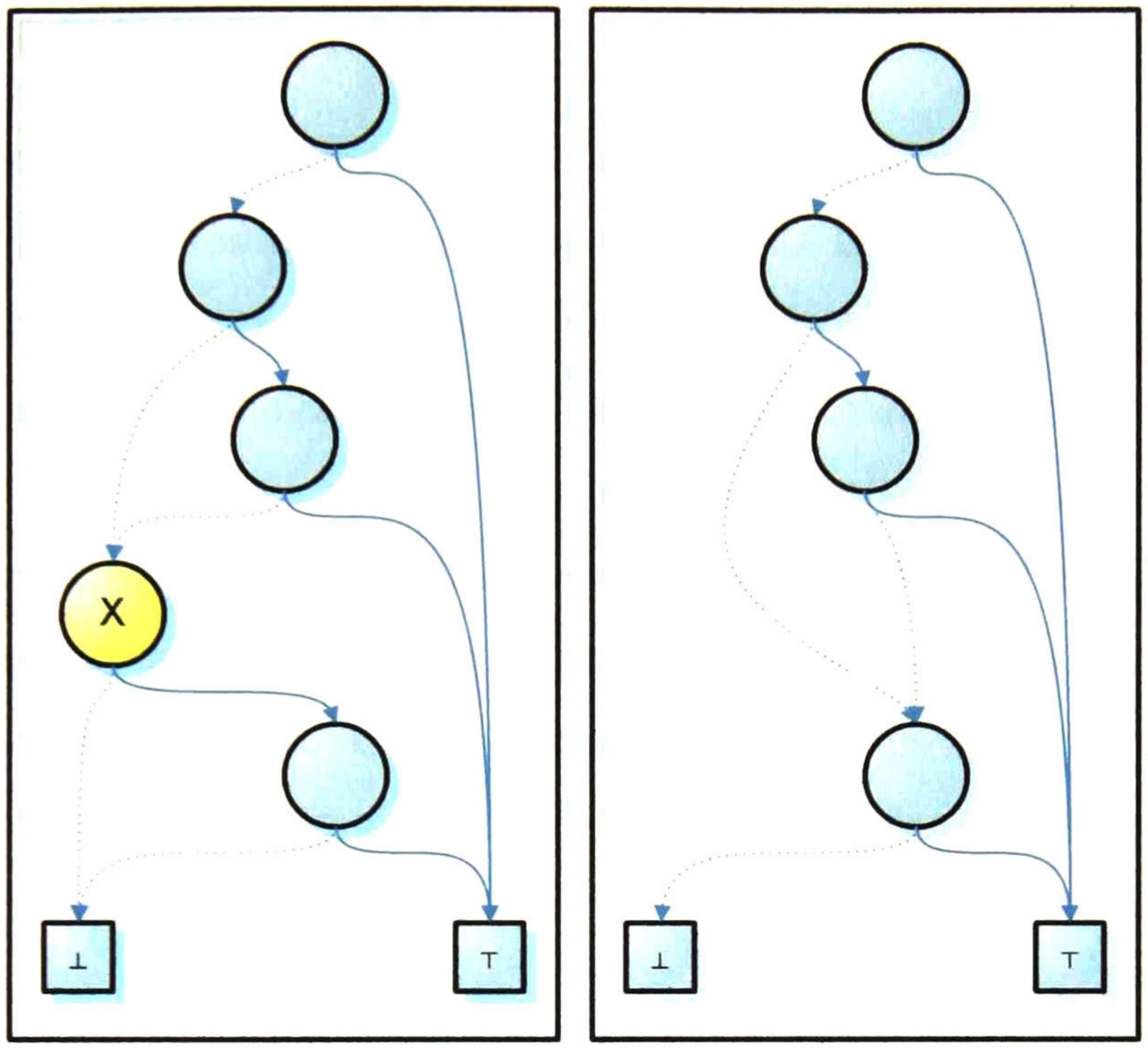


Figura 3-1 Operador Exist

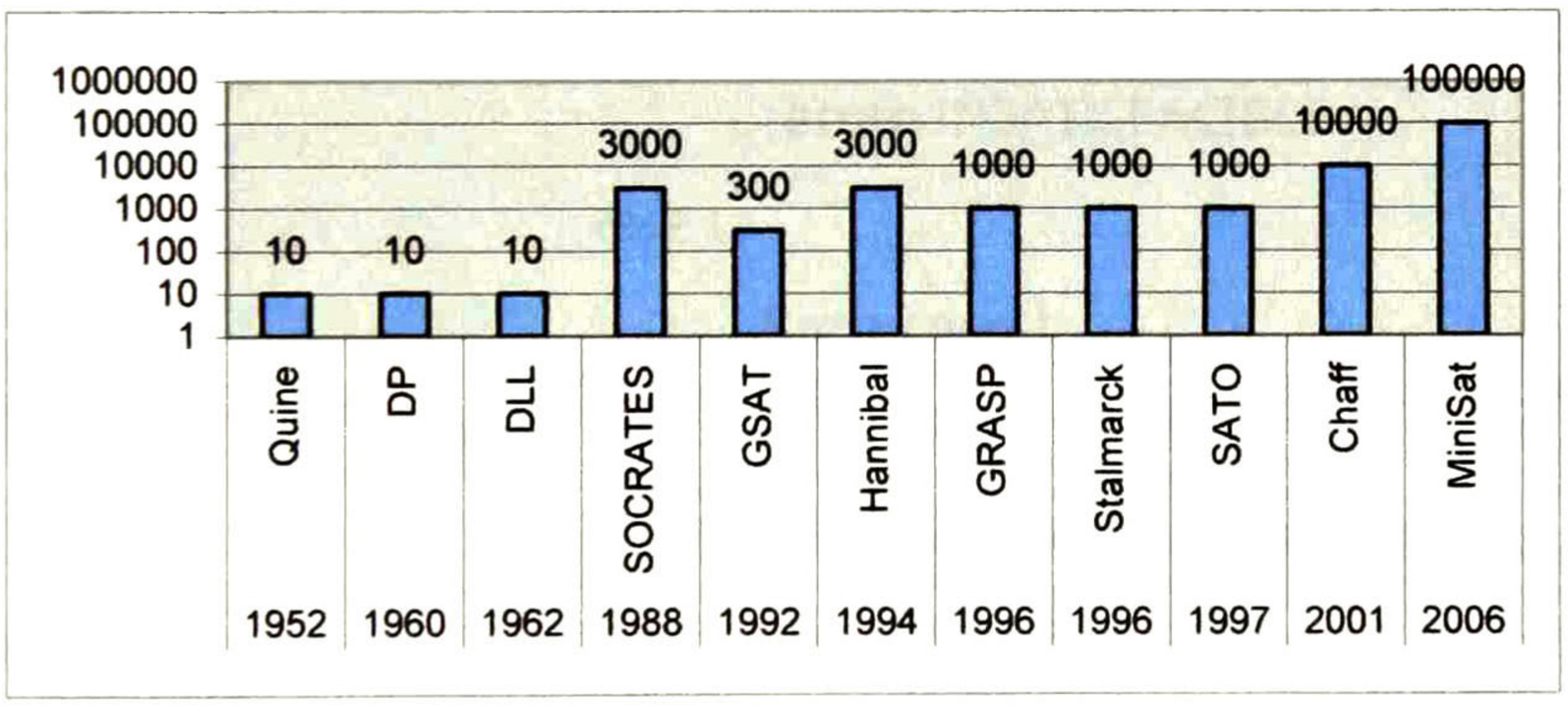
3.2 SAT Solvers y el problema de la satisfacibilidad booleana

El problema SAT es quizá el problema NP por excelencia, por esta razón ha sido ampliamente estudiado y gran parte de los esfuerzos se han enfocado en diseñar herramientas que puedan determinar de forma automatizada si determinada fórmula es o no satisfacible [ZHG03]; de esta manera han sido desarrollados un gran número de algoritmos, entre los se encuentra el utilizado por MINISAT: el DPLL. Este algoritmo es sumamente importante para nuestra investigación y lo explicaremos en detalle más adelante.

3.2.1 SAT Solvers

Los SAT Solvers son herramientas que resuelven de manera automatizada el problema SAT, a estos se les alimenta una fórmula booleana y tenemos como salida una respuesta que indica si la fórmula es satisfacible o insatisfacible, en caso de que sea satisfacible típicamente podemos también obtener una valuación que lo demuestre; existen tantos tipos de SAT Solvers como algoritmos automatizables, recientemente su uso ha permitido determinar satisfacibilidad en fórmulas de hasta 100,000 variables; una de las principales aplicaciones en la verificación formal de estos avances ha sido la comprobación de modelos acotada, en ésta se aprovecha la potencia de los modernos SAT Solvers para buscar contraejemplos a una especificación en un modelo dado.

Figura 3-2 Evolución de los SAT Solvers



El histórico incremento en el número de variables que son capaces de manejar se debe a múltiples factores, entre los que podemos destacar dos: el creciente poder de cómputo y la mejora de los algoritmos. En el siguiente apartado explicaremos uno de los mejores algoritmos que existen: el DPLL, este es la base para algunos de los SAT Solvers más destacados de los últimos años: MiniSAT [EEN03][EEN05] y zChaff [FU06].

3.2.2 Algoritmo DPLL

El algoritmo DPLL es un algoritmo sólido y completo para determinar la satisfacibilidad de una fórmula booleana, este se basa en resolución y trabaja con fórmulas especificadas en forma normal conjuntiva, fue presentado en 1962 y debe sus siglas a los apelativos de sus creadores: Martin Davis, Hilary Putman, George Logemann y Donald Loveland. Actualmente existen diversas variantes al algoritmo original, explicaremos una que utiliza aprendizaje mediante análisis de conflictos [EEN03] [ZHG03]. El núcleo del algoritmo se muestra a continuación:

```
1:      While(TRUE)
2:              Propagar()
3:              if (conflicto) then
4:                      Analizar(conflicto)
5:              if (el conflicto es de alto nivel)
6:                      return INSATISFACIBLE
7:              else
8:                      Retroceso()
9:      else
10:             if (se han asignado todas las variables)
11:                     return SATISFACIBLE
12:             else
13:                     Decide()
```

Un punto importante que no aparece explícitamente en este algoritmo es que cuando no se dispone de cláusulas unitarias para realizar la inferencia se procede a hacer suposiciones, esto implica que sólo la respuesta “satisfacible” del SAT Solver será confiable, por otra parte, si el SAT Solver determina que la fórmula

es insatisfacible bajo suposiciones, procederá a deshacer aquellas que lo han llevado a dicha conclusión.

Las suposiciones dentro del SAT Solver se introducen una a la vez intentando así sacar el mayor provecho de estas, para poder revertir los efectos de cada suposición se utiliza un mecanismo denominado "niveles de decisión", éste consiste en que cada que se realiza una suposición se introduce esta en una pila denominada "historial", también se introducirán en ella las implicaciones de dicha suposición creando con esto un "nivel de decisión" en el historial; así tenemos que nuestro historial tendrá tantos niveles como suposiciones anidadas tengamos. Cuando no se tienen suposiciones hechas en el SAT Solver se dice que éste se encuentra en el nivel de decisión 0.

A continuación explicaremos los conceptos de conflicto y conflicto de alto nivel, así como los métodos Propagar(), Analizar(conflicto), Retroceso(), Decide().

Definición 3-2 Conflicto

Dada una cláusula C y un conjunto de asignaciones A, se tiene un *conflicto* cuando C es insatisfacible bajo A. Se dice que un *conflicto es de alto nivel* cuando éste se da en el nivel de decisión 0, es decir cuando no existen suposiciones en el SAT Solver.

La función Propagar() trabaja bajo el único mecanismo de inferencia en los DPLL: la propagación de unidades; este utiliza la siguiente premisa: "dada una fórmula booleana φ en forma normal conjuntiva es necesario que toda cláusula en φ sea satisfacible para que φ lo sea" En el caso específico de las cláusulas unitarias, tenemos que para que una cláusula unitaria sea satisfacible la literal presente en

ella debe evaluar a ⊤. De esta forma la función Propagar() toma cada cláusula unitaria y la analiza si contiene como literal una variable afirmada entonces sustituye en φ todas las presencias de esta variable con ⊤, en caso de que la cláusula contenga como literal una variable negada sustituirá las presencias de la variable con ⊥. Observemos que la propagación de unidades tiene como efecto la posible generación de más cláusulas unitarias por lo que Propagar() sólo termina cuando se han procesado todas las cláusulas unitarias, incluso las generadas durante la ejecución de dicha función.

La función Analizar(conflicto) es la encargada del proceso de aprendizaje en los DPLL, se le llama cuando se ha encontrado un conflicto tras llamar a Propagar(). El conflicto se analiza mediante un grafo de implicaciones cuyos vértices son las valuaciones que llevaron al conflicto y sus aristas son las implicaciones de éstas [MCM03] [ZHG03]. Un ejemplo de un grafo de implicaciones se muestra en la siguiente figura.

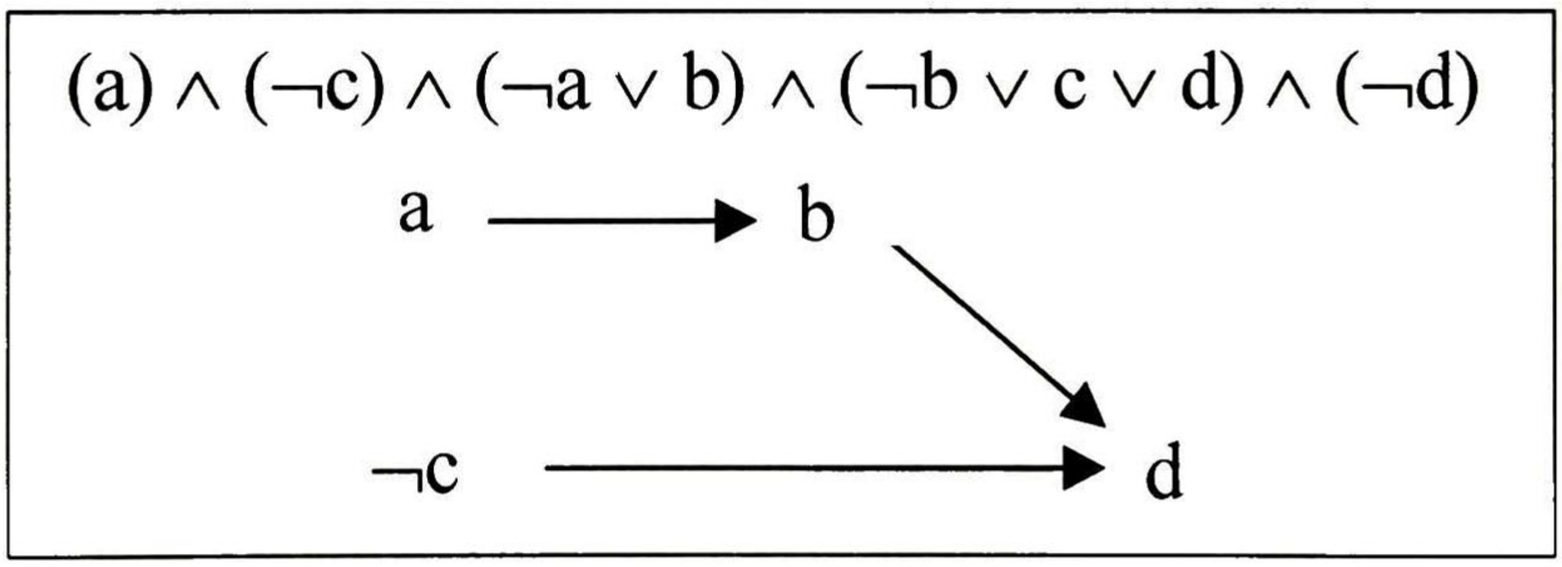


Figura 3-3 Grafo de Implicaciones.

En la Figura 3-3 se tiene que la propagación de 'a' como ⊤ y 'c' como ⊥ han llevado a un conflicto con 'd' puesto que se ha inferido que 'd' debe ser ⊤ y existe una cláusula unitaria que requiere que 'd' sea ⊥.

A este grafo de implicaciones se le denomina grafo de conflicto y se utiliza para generar una nueva cláusula que evitará que se vuelva a caer en el mismo conflicto, para esto se hace un corte de aristas en el grafo y se forma la conjunción de las literales correspondientes a las colas de las aristas cortadas, esta conjunción describe cuáles han sido las razones de este conflicto de tal forma que si la negamos obtendremos una nueva cláusula que será capaz de impedir que se caiga nuevamente en este conflicto, a la cláusula resultante de este procedimiento la llamaremos "cláusula de conflicto"

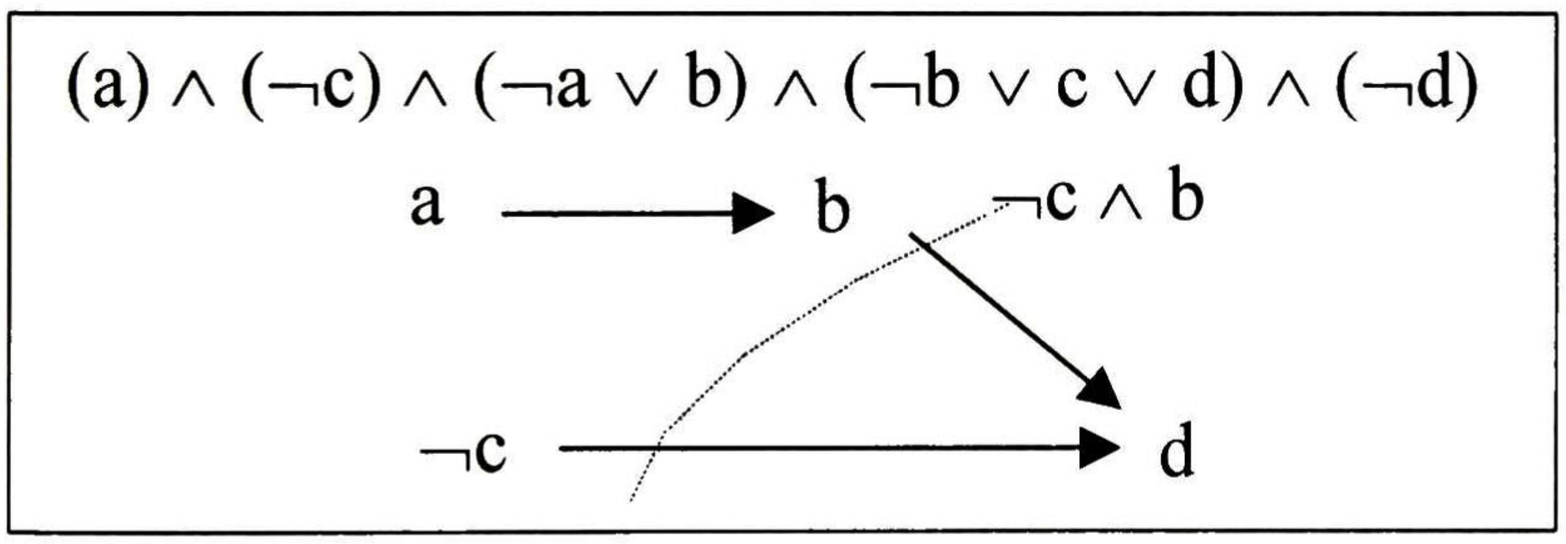


Figura 3-4 Análisis de un conflicto.

En la Figura 3-4 se aprecia que al hacer el corte en el grafo de implicaciones tenemos que mediante la introducción de la nueva cláusula (cV ¬b) se evitará caer en este conflicto mediante el mismo conjunto de asignaciones.

La función Decide() es llamada cuando no se dispone de cláusulas unitarias y no se ha llegado a un resultado concluyente. Cada llamada a Decide() crea un

nuevo nivel de decisión al introducir mediante una nueva cláusula unitaria una suposición, la selección de la literal que contendrá la cláusula se realiza mediante heurísticas. Cuando el SAT Solver determina que la fórmula es insatisfacible bajo suposiciones deberá retroceder a través de los niveles de decisión, mediante una llamada a la función Retroceso(), para después continuar la búsqueda de un resultado concluyente. Pudiera parecer que no existe entonces una ganancia al hacer suposiciones para después tener que deshacerlas; sin embargo, cada que esto sucede es posible conservar las cláusulas de conflicto generadas, ayudando así a guiar el proceso de búsqueda.

3.2.3 Heurísticas utilizadas en MiniSAT

Una parte esencial para el desempeño de cualquier SAT Solver son las heurísticas, que ayudan a guiar el proceso de búsqueda. En MiniSAT se implementan dos heurísticas de actividad, la de mayor relevancia es la utilizada por la función Decide(), esta es una variación de la heurística VSIDS introducida por Chaff.

Heurística de decisión

Esta heurística trata de medir la relevancia que tiene cada variable respecto al proceso búsqueda[EEN03][YI07], para esto a cada variable se le asocia una puntuación inicializada en cero, cada vez que se crea una cláusula de conflicto se incrementa la puntuación de las variables que aparecen en ella; dado que las cláusulas de conflicto suelen ser más relevantes conforme avanza el proceso de búsqueda es deseable hacer una distinción cronológica de la actividad de las variables, por esta razón, las puntuaciones periódicamente se dividen entre una

constante de forma que los incrementos tengan mayor valor cuanto más recientes sean.

Cuando se llama a la función Decide(), esta busca entre las variables que no tienen un valor de verdad asignado aquella que tenga la mayor puntuación e introduce, como suposición, la cláusula unitaria que contiene a esta variable.

Heurística de eliminación de cláusulas

Con el uso de esta heurística se ataca uno de los grandes inconvenientes de la búsqueda guiada por aprendizaje: el exceso de cláusulas de conflicto [EEN05]. Cada que se llega a un conflicto se introduce una cláusula que evita que la búsqueda vuelva sobre los pasos que le llevaron a dicho conflicto. El problema con este enfoque radica en que cuando la búsqueda se alarga, llega el punto en que las cláusulas de conflicto son más que las cláusulas originales del problema, al incrementarse el número de cláusulas se incrementan los tiempos de propagación y el uso de memoria. Para solucionar este problema se fija a cada cláusula de conflicto una puntuación, inicializada en cero, esta puntuación se incrementa cada que la cláusula de conflicto etiqueta una arista en un grafo de conflicto, al igual que con la heurística de decisión estas puntuaciones se dividen periódicamente, por último tenemos que se eliminan aquellas cláusulas de conflicto cuya puntuación sea demasiado baja.

Capítulo 4
Comprobación de propiedades LTL usando técnicas de abstracción

La lógica temporal lineal es sumamente útil en la especificación de propiedades temporales puesto que mediante su uso se pueden expresar comportamientos a través del tiempo. Sin embargo, la verificación de propiedades LTL es en la práctica mucho más complicada que la verificación de propiedades expresadas en otras lógicas temporales como CTL; es por esta razón que la mayor parte de los esfuerzos de investigación se han enfocado en optimizar la verificación CTL.

En este capítulo expondremos un marco de trabajo para agilizar la verificación de propiedades LTL.

Nuestro marco de trabajo integra diversas técnicas de abstracción dependientes de la propiedad y se describe en el siguiente algoritmo:

1. Simplificación del modelo.
2. Generación de la abstracción inicial.
3. Comprobación del modelo abstracto
   3.1. Si la propiedad se cumple terminar.
4. Validación del contraejemplo abstracto.
   4.1. Si el contraejemplo es válido terminar, la propiedad no se cumple.
5. Generar un refinamiento del modelo abstracto y volver al paso 3

La aplicación de todos estos pasos preserva solidez y completud en la verificación de propiedades LTL.

Con el objetivo de apoyar esta tesis con resultados experimentales de la aplicación de este algoritmo se realizó una implementación del mismo extendiendo el comprobador de modelos NuSMV, a esta extensión le llamaremos AbsNuSMV, al final de cada uno de los siguientes subtemas se incluye una breve explicación de los detalles relevantes de la implementación.

La versión de NuSMV usada en esta investigación es nativa para Win32, la hemos portado para realizar nuestros experimentos en entorno Windows, además la hemos extendido incorporándole a AbsNuSMV, de esta forma ambas implementaciones conviven en el mismo programa ejecutable. A este programa le hemos llamado XNuSMV.

4.1 Simplificación del modelo

Desde nuestra perspectiva cualquier esfuerzo por verificar modelos realmente grandes debería iniciar con una simplificación del modelo, una buena forma de hacerlo es aplicando la *reducción al cono de influencia*. Con esta técnica se explota el hecho de que las variables de interés en las propiedades a verificar suelen ser un subconjunto muy pequeño de las variables del sistema, por lo tanto es posible que existan variables en el sistema que son completamente irrelevantes a la propiedad. Para detectar y eliminar estas variables irrelevantes es necesario calcular primero cuáles variables sí tienen influencia sobre el comportamiento de las variables de la propiedad.

El algoritmo básico para el cálculo del cono de influencia se describe a continuación:

Sean φ y M la propiedad a verificar y el modelo del sistema respectivamente.

1. Para cada variable $v \in M$ calcule el conjunto de variables P_v de las cuales v depende directamente.
2. Calcule el conjunto VOC(φ).
3. Creé un conjunto de variables $CONO_0 := \{ v \mid v \in VOC(\varphi)\}$.
4. Para i>1 calcule de la siguiente forma un nuevo conjunto de variables $CONO_i$:
   - 4.1. $CONO_i := \{v \mid v \in CONO_{i-1}\}$.
   - 4.2. Para toda $v \in CONO_{i-1}$ hacer que $CONO_i := CONO_i \cup P_v$.
5. Si $CONO_i \neq CONO_{i-1}$ ir al paso 4.
6. Eliminar del modelo del sistema toda variable que no pertenezca a $CONO_i$.

Cuando el sistema está especificado mediante ecuaciones de estado siguiente el algoritmo se simplifica grandemente, puesto que para cada variable a la derecha de una ecuación el paso 1 corresponde al cálculo del vocabulario de la parte izquierda de su ecuación, si una variable no aparece a la izquierda de una ecuación entonces el conjunto de las variables de las cuales depende es vacío. Por otra parte la eliminación de variables es tan simple como no tomar en cuenta aquellas ecuaciones cuya variable de la parte derecha no pertenezca al cono de influencia; dado que NuSMV representa el sistema mediante ecuaciones de estado siguiente, la reducción al cono de influencia se calcula en tiempo lineal explotando las simplificaciones anteriormente expuestas. Al modelo reducido al cono de influencia le llamaremos M_{Red}.

4.2 Generación de la abstracción inicial

Se genera mediante una sobre aproximación a M_{Red} un modelo abstracto M' Es deseable que la abstracción inicial tenga un tamaño mucho menor que el modelo original.

En esta investigación hemos optado por la abstracción existencial de variables como técnica para la generación de abstracciones; en el caso de la abstracción inicial creamos un nuevo modelo que sólo contenga presencias de las variables del vocabulario de la propiedad.

En NuSMV el modelo de sistema es representado mediante BDDs en ellos es posible realizar la cuantificación existencial de variables mediante el operador Exist. Dada la capacidad expresiva de los BDDs no es posible representar un modelo de sistema usando un solo BDD, sin embargo es posible representarlo si se usan

tres: uno para el conjunto de estados, otro para el conjunto de estados iniciales y uno más para la relación de transición.

Cabe recordar que la comprobación de propiedades LTL en NuSMV se implementa mediante comprobación de modelos CTL con condiciones de justicia; por esta razón es necesario añadir al modelo del sistema un BDD para las condiciones de justicia.

Anteriormente mencionamos que nuestra abstracción inicial sólo contiene presencias de las variables del vocabulario de la fórmula, esto es cierto en teoría, sin embargo en AbsNuSMV la abstracción inicial sólo contiene presencia de las variables del vocabulario del tableau de la negación de la propiedad, la diferencia estriba en que con el objetivo de generar una fórmula verificable con un comprobador CTL con condiciones de justicia, se agregan variables adicionales al tableau, estas variables hacen posible comprobar fórmulas de trayectoria con algoritmos de comprobación de fórmulas de estado y por lo tanto no deben ser abstraídas.

A continuación se presenta el algoritmo implementado en AbsNuSMV para la generación de la abstracción inicial:

Sean φ y M la propiedad a verificar y el modelo del sistema, respectivamente, sean además bddE, bddI, bddR, bddJ los BDDs del conjunto de estados, los estados iniciales, la relación de transición y las condiciones de justicia del modelo original, respectivamente, y sean las versiones primadas de los mismos los correspondientes BDDs del modelo abstracto.

1. Genere $T_{-\varphi}$ que es el Tableau de $-\varphi$.
2. bddE':= bddE
3. bddI':= bddI
4. bddR:=bddR'
5. bddJ':= bddJ.
6. Calcule el conjunto de variables INV:= VOC(M)-VOC($T_{-\varphi}$).
7. Para toda variable v ∈ INV aplique:
    a. Exist (bddE', v)
    b. Exist (bddI', v)
    c. Exist (bddR', v)
    d. Exist (bddJ', v).

4.3 Comprobación del modelo abstracto

Dados un modelo abstracto M' y una propiedad φ, se procede a verificar que M', 0 ⊨ φ usando comprobación CTL con condiciones de justicia. A continuación presentamos el algoritmo para la realización de dicha verificación:

1. Cree $T_{-\varphi}$ que es el Tableau de la negación de la propiedad. Agregando las condiciones de justicia necesarias.
2. Calcule M'x$T_{-\varphi}$ mediante el producto de M' y $T_{-\varphi}$.
3. Compruebe que M'x$T_{-\varphi}$ ⊨ EG T.
    3.1. Si M'x$T_{-\varphi}$ ⊭ EG T entonces M⊨φ, y la verificación termina.
    3.2. Si M'x$T_{-\varphi}$ ⊨ EG T entonces debemos generar una traza abstracta π' que lo demuestre.

NuSMV representa las trazas por medio de vectores de BDDs y debido a las simplificaciones aplicadas a los BDDs en el proceso de verificación, es posible que no contengan presencias de algunas variables (esto indica que el valor de la misma puede ser 0 o 1), sin embargo para la creación de la traza abstracta π' todas las variables son relevantes, por lo que hay que asignar un valor de verdad incluso a aquellas variables que no aparezcan en el BDD. Un punto sutil a observar en la generación de trazas de modelos abstractos es que estas sólo deben contener presencias de las variables del modelo abstracto, es decir que las variables cuantificadas existencialmente no deben de aparecer en ellas.

4.4 Validación del contraejemplo abstracto

Debido a que el contraejemplo procede de la verificación de un modelo sobre aproximado, es posible que estemos tratando con un contraejemplo espurio. Por lo tanto, para poder afirmar que el modelo concreto no cumple la propiedad, es necesario asegurarnos de la correctud del contraejemplo.

Existen diferentes métodos para realizar la validación de un contraejemplo abstracto, en este estudio retomamos la idea expuesta en [CLK02], haciendo la validación por medio de un SAT Solver del estado del arte; las premisas de este método son simples:

- La comprobación de modelos acotada permite generar contraejemplos de longitud K.

- Todo contraejemplo abstracto conocido tiene una longitud conocida k'

La fórmula booleana generada por el comprobador de modelos acotado [CLK01][BIR03] para una cota K, una fórmula φ y un modelo M, se puede interpretar como:

¿Existe un contraejemplo a φ en M que tenga longitud K?

De manera que si generamos la fórmula BMC para la cota k' y le agregamos, como restricción, una fórmula proposicional $C_{\pi'}$ que corresponde al contraejemplo abstracto π' obtenemos la siguiente interpretación:

¿Existe un contraejemplo a φ en M que tenga longitud K y que además corresponda a π'?

La fórmula en cuestión es la siguiente:

$$[\![M, \varphi]\!]_k \wedge C_{\pi'}$$

Si la fórmula anterior es satisfacible, entonces existe un contraejemplo a la propiedad en el modelo concreto y la verificación termina. Si la fórmula es insatisfacible entonces no existe un contraejemplo concreto de longitud K y tenemos que el contraejemplo π' es espurio.

Recordemos que el comprobador de modelos acotado crea variables nuevas que representan las variables del problema instanciadas en el tiempo. Es necesario conocer dichas variables para generar $C_{\pi'}$.

El algoritmo para la verificación del contraejemplo abstracto se describe a continuación, observe que fórmula $C_{\pi'}$ se genera en los pasos 4 y 5 del algoritmo:

1. Asigne a K la longitud del contraejemplo π'

2. Usando un comprobador de modelos acotado genere $[\![M, \varphi]\!]_k$.

3. Cree una fórmula BMC' en CNF que sea equisatisfacible[1] a $[\![M, \varphi]\!]_k$.

4. Cree una fórmula proposicional $C_{\pi'}$ tal que $C_{\pi'} := \top$.

5. Para todo instante de tiempo t menor o igual a K y para toda x variable en π' presente en el tiempo t:

   5.1. Determine la variable x^t creada por el comprobador de modelos para la variable x, en el tiempo t.

   5.2. Cree una fórmula ψ en CNF equisatisfacible a $(x^t \leftrightarrow val_{x,t})$. Donde $val_{x,t}$ es el valor asignado en π' a x en el instante t.

   5.3. Redefina $C_{\pi'}$ como $C_{\pi'} := C_{\pi'} \wedge \psi$, observe que $C_{\pi'}$ conserva la CNF

6. Verifique la satisfacibilidad de BMC' $\wedge$ $C_{\pi'}$.

   6.1. Si es satisfacible, entonces φ no se cumple en M. La valuación extraída del SAT Solver lo demuestra.

   6.2. Si es insatisfacible, entonces π' es un contraejemplo espurio.

---

[1] Dos fórmulas proposicionales φ y ψ son equisatisfacibles si se cumple que φ es satisfacible si y solo si ψ es satisfacible. [NNN98][JCK04].

A pesar de que cada verificación de un contraejemplo implica una comprobación acotada del modelo, el proceso es mucho más rápido que una corrida habitual del comprobador. Esto se debe a que las restricciones agregadas por el modelo abstracto suelen ser cláusulas unitarias y permiten al SAT Solver la deducción rápida de información adicional.

4.5 Generar un refinamiento del modelo abstracto

En este punto sabemos que el modelo abstracto de sistema M' contiene un comportamiento π' que invalida la propiedad φ en él; además sabemos que no existe dicho comportamiento en el modelo concreto. Por lo tanto, a fin de llevar a buen término la verificación, es necesario crear un nuevo modelo abstracto de sistema M'' que no contenga a π'.

Dado que hemos creado el modelo abstracto eliminando variables del modelo original, es posible crear un modelo abstracto diferente eliminando un conjunto de variables diferente.

Definición 4-1 Refinamiento

Sea M un modelo concreto y sean M' y M'' dos modelos sobre aproximados a M obtenidos mediante abstracción existencial de variables. Decimos que M'' es un refinamiento de M' si y sólo si $VOC(M') \subset VOC(M'') \subset VOC(M)$.

Teorema 4-1 Complejidad de encontrar el refinamiento mínimo

El problema de encontrar un refinamiento de tamaño minimal es de complejidad NP-difícil [CLK03].

Meses antes del comienzo de esta investigación implementamos un SAT Solver utilizando el método de Stålmarck [SHR99], utilizamos estructuras eficientes y uno de los algoritmos que han dado mejores resultados en la verificación de problemas industriales; aunque los tiempos que obtuvimos con dicho SAT Solver eran buenos, quedaban muy lejos de los que obtienen SAT Solvers como MiniSAT o zChaff. Esto nos llevó a un análisis de qué factores hacían tales diferencias.

Para realizar el análisis de diferencias, en primer lugar identificamos las tareas principales realizadas en MiniSAT, enseguida determinamos sus equivalencias en nuestra implementación. Con este sencillo análisis encontramos que prácticamente todas las partes eran equivalentes, todas excepto una: nuestra implementación carecía de un método de aprendizaje que guiara la búsqueda. Esa única diferencia en los algoritmos se ve reflejada en una enorme diferencia en el desempeño de ambas implementaciones, puesto que realiza una distinción de la información relevante de la que no lo es.

Al tiempo en que obtuvimos estas conclusiones ya habíamos iniciado esta investigación y estábamos en búsqueda de una forma de distinguir las variables relevantes de las irrelevantes, para así realizar los refinamientos de modelos abstractos.

Es así que concluimos que las heurísticas utilizadas en los SAT Solvers podían también ser utilizadas para el refinamiento de modelos abstractos, evitando así la complejidad NP. Esto con los pros y los contras habituales del uso de heurísticas: se obtiene una solución rápidamente, pero no es esta una solución óptima.

Investigando si los SAT Solvers ya habían sido utilizados con este fin, encontramos los trabajos de [HNZ04] [GPT03] [CHN02], destacando entre ellos la investigación de Pankaj Chauhan [CHN02], la cual tomamos como base para hacer los refinamientos y la extendemos con un análisis de la estructura estática inicial del modelo.

El uso de las heurísticas de aprendizaje de los SAT Solvers tiene la ventaja adicional de que no son necesarios para el refinamiento cálculos complejos ni análisis dedicados, simplemente reutilizamos los cálculos realizados en el proceso de validación del contraejemplo para obtener un refinamiento del modelo abstracto.

4.5.1 Refinamiento basado en heurísticas

Hemos utilizado cinco métricas de calificación de variables, de modo que al reintegrar al modelo abstracto aquellas que obtengan mayor calificación, se maximicen las expectativas de obtención de un refinamiento minimal. Las métricas seleccionadas son las siguientes:

- **Número de apariciones en el modelo simplificado (NA)**: es natural suponer que las variables con mayor número de apariciones en el modelo concreto tienen mayor influencia en él y por tanto tendrán mayor influencia en el modelo abstracto. Sin embargo, hay que ser cuidadosos al respecto, puesto que incluirlas sólo por este criterio, puede llevar a que los modelos generados no presenten reducciones significativas.

- **Posición en el cono de influencia (POS)**: la relación de dependencia de variables no suele ser una relación de orden parcial; sin embargo, es posible determinar la distancia minima de cada variable hacia el

vocabulario de la propiedad, con el propósito de dar más importancia a las variables cuya distancia sea menor.

- **Distancia respecto al vocabulario del modelo abstracto (DA)**: al utilizar abstracción existencial eliminamos algunas dependencias de variables en el modelo abstracto, por lo tanto, para restaurar las dependencias correctas, es necesario que las variables que serán reintegradas, tengan influencia directa sobre las variables del vocabulario del modelo abstracto.

- **Número de apariciones en las cláusulas de conflicto generadas en la verificación del contraejemplo (NC)**: como hemos visto las cláusulas de conflicto nos sirven para distinguir las causas de la insatisfacibilidad de un problema; esto lo hacen almacenando en ellas las variables cuyo comportamiento causa la insatisfacibilidad. Por lo tanto, tenemos que cuantas más apariciones tenga una variable en las cláusulas de conflicto, más responsable es de que el problema sea insatisfacible. Como estamos verificando la veracidad de un contraejemplo, podemos asumir que si una variable causa que el problema sea insatisfacible, entonces esa variable causa que el contraejemplo sea espurio.

- **Relevancia como variable de reingreso en la verificación del contraejemplo (RR)**: cada que un conflicto es detectado en el SAT Solver, se verifica si es de alto nivel o se debe a las suposiciones hechas. Cuando un conflicto se da debido a suposiciones incorrectas, es necesario analizar las implicaciones de éstas y deshacer el trabajo de búsqueda infructuoso. El análisis arroja dos productos: una cláusula de conflicto y una variable de reingreso que promete mejores resultados cuando se hagan suposiciones respecto a ella. Tenemos así que entre más niveles de decisión hayan sido desechos en el conflicto, más importante es la variable para el problema.

A cada variable se le asocia una puntuación para cada métrica. Usaremos notación de función para referirnos a la puntuación que ha obtenido una variable en una métrica específica; así tenemos que NC(x) significa: "Número de cláusulas de conflicto en que aparece la variable x"

NA se incrementa con cada incidencia de la variable en la parte derecha de una ecuación de estado siguiente. NC se incrementa por cada aparición de la variable en una cláusula de conflicto. A POS se le asigna el resultado de |VOC(M)| menos la distancia minima de la variable a VOC(φ). DA es un entero en el dominio {0, 1} que tiene valor 1 si y sólo si para la variable evaluada, existe alguna variable en VOC(M') que depende directamente de ella en M. RR es un número real que se incrementa en $2^{(|Inv|-dl)/c}$, donde: *Inv* es el número de variables abstraídas existencialmente, *dl* es el número niveles de decisión desechos y *c* es una constante de normalización.

La puntuación total (PT) de cada variable se obtiene al evaluar el siguiente polinomio:

$$PT = DA*(C^1*NA + C^2*POS + C^3*NC + C^4*RR)$$

Observemos que NA y POS se pueden calcular directamente a partir del modelo simplificado. Por otra parte NC y RR serán calculadas en el SAT Solver en cada iteración. Observemos también que sólo existe una puntuación NC y una RR para cada variable del modelo original, de manera que las puntuaciones serán independientes del tiempo en que hayan sido instanciadas las variables. En lo que respecta a DA, está deberá ser calculada en cada paso tomando en cuenta el modelo abstracto actual y el modelo original.

Las variables que obtengan mayor puntuación durante la evaluación de veracidad del contraejemplo abstracto, serán reintegradas en el nuevo modelo abstracto.

El algoritmo para realizar el refinamiento se presenta a continuación. Este utiliza un contraejemplo abstracto π', una propiedad φ y un modelo M para generar un nuevo modelo M'' que refina a M'

1. Calcule el conjunto de variables abstraídas existencialmente Inv:=VOC(M)-VOC(π').
2. Calcule los PT de las variables en Inv.
3. Seleccione las *n* variables con mayor PT y forme con ellas un conjunto de nombre R.
4. Calcule el conjunto de variables Inv':=Inv-R.
5. Cree un nuevo modelo abstracto M'' abstrayendo existencialmente las variables de Inv'

Se propone que la constante n usada en el paso 3 corresponda a un porcentaje pequeño del número de variables del modelo simplificado.

Capítulo 5
Trabajo experimental

En este capítulo presentamos los resultados experimentales obtenidos con AbsNuSMV y los comparamos con los resultados obtenidos por NuSMV. Para la realización de las pruebas hemos incluido modelos escalares, booleanos, multiproceso y mono proceso. Todos los modelos están especificados en archivos .smv, los cuales también contienen una propiedad LTL.

5.1 Metodología de pruebas

Hemos realizado dos grupos principales de pruebas: en el primero todas las propiedades a verificar son verdaderas, en el segundo todas son falsas. Ambos grupos han sido verificados por AbsNuSMV usando las técnicas de abstracción antes descritas y por NuSMV usando BDDs.

Los factores comunes a evaluar en AbsNuSMV y NuSMV son:

- Cantidad de milisegundos requerida para la verificación (ms).
- Longitud del contraejemplo encontrado (Cex).

Los factores a evaluar sólo en AbsNuSMV son:

- Número de refinamientos (#Ref).
- Numero de variables invisibles al final de la verificación (#Inv).
- Milisegundos promedio por paso de verificación (msP).

Todas las pruebas se realizaron en una computadora con procesador Intel Core2 duo a 1.6GHz con 2GB de memoria RAM corriendo sobre Windows Vista Home Premium de 32bits.

Tanto NuSMV como AbsNuSMV se ejecutaron en versiones nativas para Win32. Originalmente NuSMV no se encuentra disponible para este tipo de plataformas, por lo que fue necesario portarlo antes de comenzar con la implementación de AbsNuSMV.

5.2 Resultados de los experimentos

La tabla 5-1 presenta los resultados obtenidos al verificar el primer grupo de benchmarks, en este grupo todas las propiedades son verdaderas. En cada refinamiento hecho durante la verificación con AbsNuSMV hemos reintegrado una variable.

Resultados con Propiedades Verdaderas							
Modelo	NuSMV		AbsNuSMV				
Benchmark	ms	Cex	ms	Cex	#Ref	#Inv	msP
ken.flash^03.C.smv	8.981937	x	9.795588	x	0	39	9.795588
ken.flash^04.C.smv	49.680546	x	842.206848	x	0	52	842.206848
ken.flash^05.C.smv	41.588943	x	388.870239	x	0	52	388.870239
ken.flash^06.C.smv	5.515575	x	25.291412	x	0	36	25.291412
ken.flash^07.C.smv	23.260637	x	524.0755	x	0	52	524.0755
ken.flash^08.C.smv	25.212702	x	586.389343	x	0	50	586.389343
ken.flash^09.C.smv	16.409418	x	37.240276	x	0	49	37.240276
ken.flash^10.C.smv	23.429235	x	39.083179	x	0	37	39.083179
ken.flash^11.C.smv	93.807243	x	373.293335	x	0	62	373.293335
ken.flash^13.C.smv	2.086857	x	6.687023	x	0	35	6.687023
ken.flash^14.C.smv	21.379742	x	191.092438	x	0	48	191.092438
ack.smv	7.805881	x	75.376968	x	13	0	5.798228308
arbiter.smv	69.096634	x	514.165527	x	23	0	22.35502291
snddrv.smv	8.366706	x	90.828377	x	12	2	7.569031417
cmu.gigamax.B.smv	13.51771	x	940.240479	x	18	21	49.486341

Tabla 5-1 Resultados del primer grupo de benchmaks.

Las tabla 5-2 presenta los resultados obtenidos al verificar el segundo grupo de benchmarks, en este grupo todas las propiedades son falsas.

Resultados con Propiedades Falsas							
Modelo	NuSMV		AbsNuSMV				
Benchmark	ms	Cex	ms	Cex	#Ref	#Inv	msP
ken.flash^02.C.smv	576.027283	6	3199.627197	6	21	8	145.4375999
ken.flash^03.C.smv	52.397999	3	557.919373	3	4	35	111.5838746
ken.flash^04.C.smv	97.509529	2	940.828369	2	0	52	940.828369
ken.flash^05.C.smv	89.019974	2	529.174316	2	0	52	529.174316
ken.flash^06.C.smv	19.340237	2	61.217628	2	0	36	61.217628
ken.flash^07.C.smv	74.120941	2	633.220032	2	0	52	633.220032
ken.flash^08.C.smv	64.59243	2	834.829102	2	0	50	834.829102
ken.flash^09.C.smv	105.999718	2	213.025253	2	0	49	213.025253
ken.flash^10.C.smv	114.545357	2	637.248291	2	0	37	637.248291
ken.flash^11.C.smv	198.998688	2	592.761047	2	0	62	592.761047
ken.flash^12.C.smv	26316.32813	6	22280.99219	6	25	11	856.961238
ken.flash^13.C.smv	13.181214	2	36.971809	2	0	35	36.971809
ken.flash^14.C.smv	52.599842	2	318.889557	2	0	48	318.889557
ack.smv	55.156731	3	60.148705	3	12	0	4.626823462
arbiter.smv	252.60675	5	331.947845	5	22	0	14.432515
snddrv.smv	120.151863	4	76.445396	4	13	0	5.460385429
eijk.S208.S.smv	1577.596558	2	667.65741	2	0	2	667.65741
eijk.S208c.S.smv	1516.328857	2	712.90625	2	0	3	712.90625
eijk.S208o.S.smv	978.046021	2	1262.160156	2	0	0	1262.160156
eijk.S298.S.smv	131.220032	11	4921.579102	11	34	0	140.6165458
eijk.S386.S.smv	706.389221	3	49427.4375	3	15	5	3089.214844
eijk.S444.S.smv	584103.6875	5	11871.98145	3	2	46	3957.327148
Multiplicador.smv	Out of Mem	-	566321.50	4	39	0	14521.0641
cmu.gigamax.B.smv	166.873032	3	23.57206	2	0	38	23.57206

Tabla 5-2 Resultados del segundo grupo de benchmaks.

A continuación presentamos los resultados obtenidos al reintegrar cinco variables por refinamiento, en la siguiente tabla solo incluimos los resultados de la verificación de aquellos modelos en los que hemos tenido que refinar para fin de terminar una verificación con AbsNuSMV.

Modelo	NuSMV		AbsNuSMV				
Benchmark	ms	Cex	ms	Cex	#Ref	#Inv	msP
Resultados con Propiedades Verdaderas							
cmu.gigamax.B.smv	13.51771	x	211.892288	x	4	18	42.3784576
Resultados con Propiedades Falsas							
ken.flash^02.C.smv	576.027283	6	949.04895	6	5	4	158.174825
ken.flash^03.C.smv	52.397999	3	261.560974	3	1	34	130.780487
ken.flash^12.C.smv	26532.27148	6	5561.665527	6	5	11	926.9442545
ack.smv	55.156731	3	22.231178	3	12	0	5.5577945
arbiter.smv	252.60675	5	100.799263	5	5	0	16.7998771
snddrv.smv	120.151863	4	26.117565	4	3	0	6.52939125
eijk.S298.S.smv	131.220032	11	1632.971558	11	7	0	204.12144475
eijk.S386.S.smv	706.389221	3	18613.13867	3	4	0	3722.627734
eijk.S444.S.smv	584103.6875	5	9210.669922	3	1	43	4605.334961
Multiplicador.smv	Out of Mem	-	99575.921875	4	8	0	24893.9804

Tabla 5-3 Resultados al reintegrar cinco variables por refinamiento.

En las los resultados de las pruebas podemos observar que cuando se usa AbsNuSMV se presenta un incremento en el tiempo total de verificación. Esto se debe a que la cuantificación existencial de variables es una operación computacionalmente costosa, sin embargo el costo computacional de la verificación de cada modelo abstracto es menor puesto que en gran parte de los casos la búsqueda del contraejemplo se realiza en un modelo cuyo espacio de estados es mucho menor.

El benchmark cuyos resultados nos parecen mas alentadores es el "Multiplicador.smv". A continuación presentamos los registros de uso de memoria en la verificación de este ejemplo.

En la Figura 5-1 podemos observar cómo al tratar de verificar este benchmark, se dispara el uso de memoria, este comportamiento se mantiene al punto de agotar la memoria disponible.

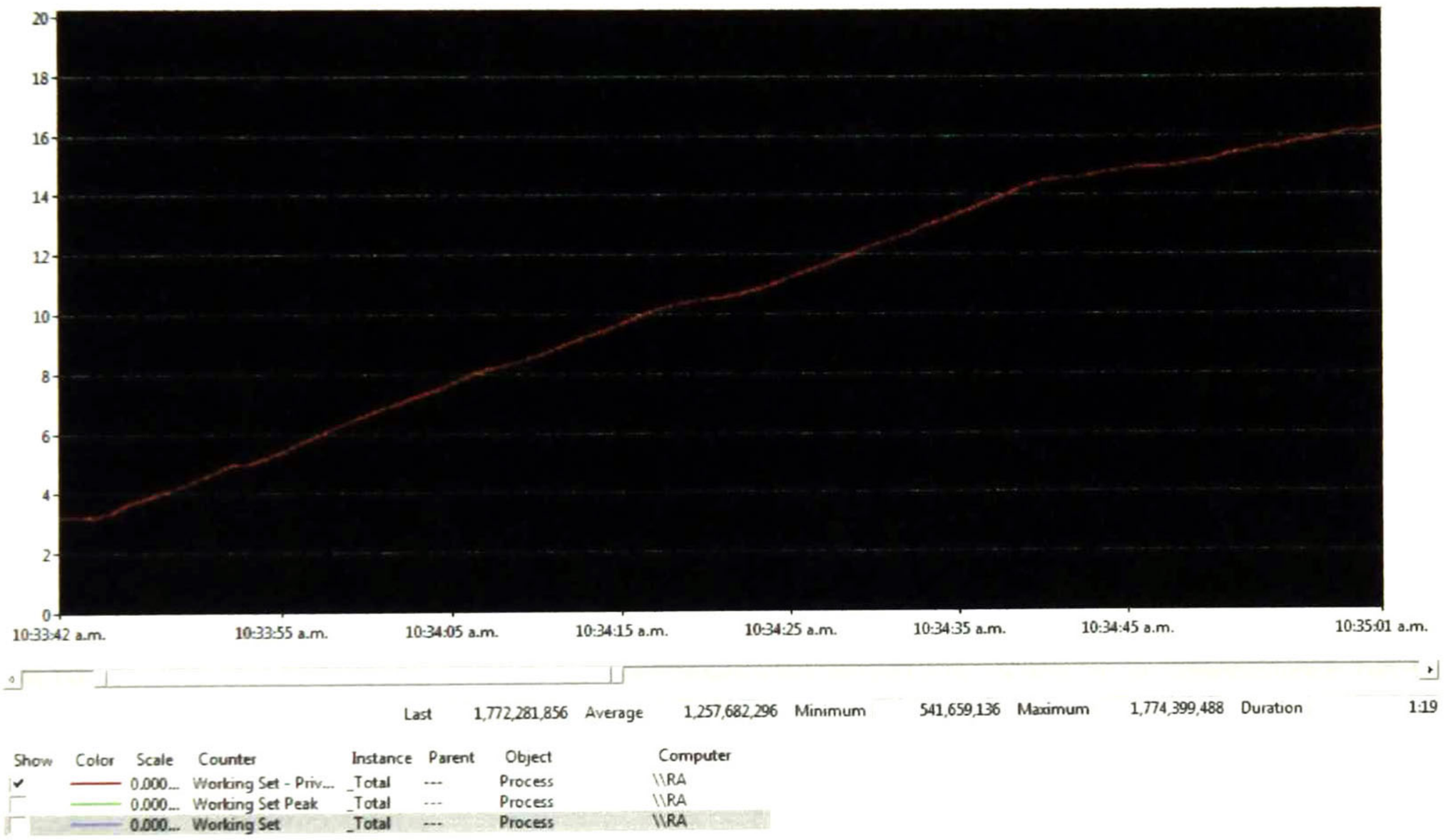


Figura 5-1 Uso de memoria para NuSMV multiplicador.smv

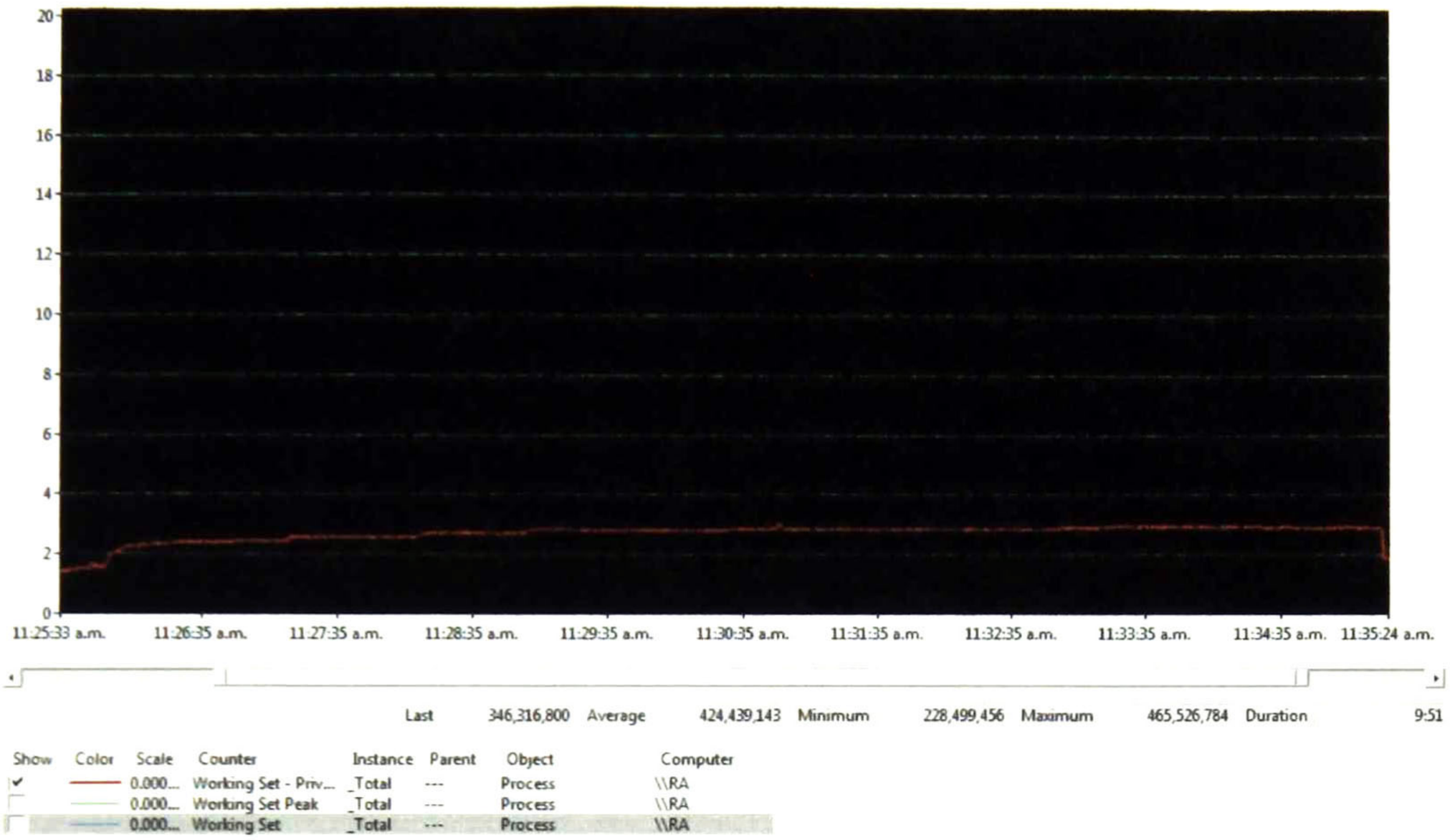


Figura 5-2 Uso de memoria para NuSMV -Abs Mutiplicador.smv

En la Figura 5-2 Uso de memoria para NuSMV -Abs Mutiplicador.smv, podemos observar como el consumo de memoria es mucho menor cuando se utilizan las técnicas de abstracción. Además pueden observarse pequeños escalones en el uso de memoria se incrementa estos se deben a la reincorporación de variables al modelo.

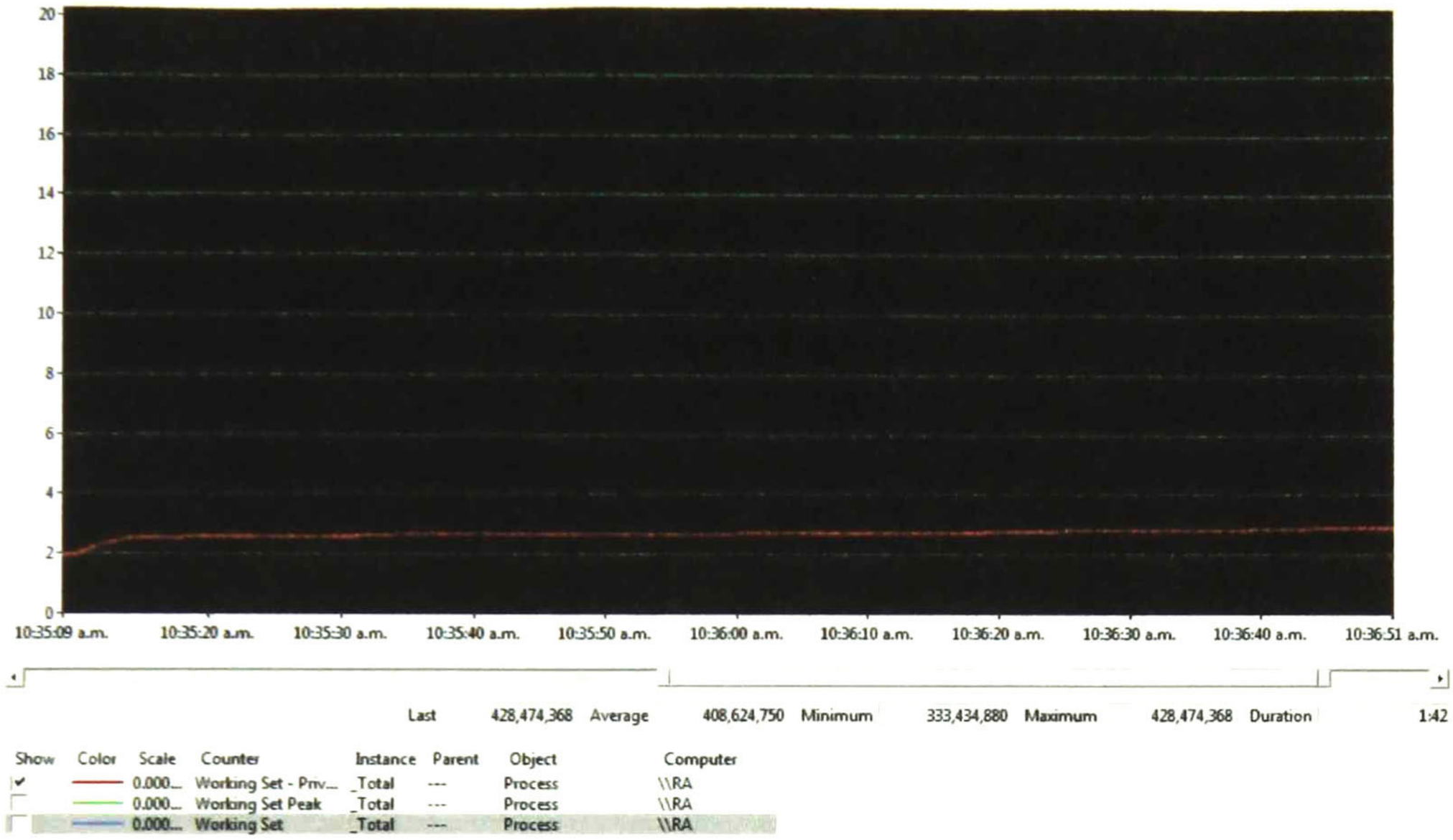


Figura 5-3 Uso de memoria para NuSMV -Abs Multiplicador reincorporando cinco variables por refinamiento

En la Figura 5-3 Uso de memoria para NuSMV -Abs Multiplicador reincorporando cinco variables por refinamientoMV -Abs Multiplicador reincorporando cinco variables por refinamiento}, podemos observar como el consumo de memoria se mantiene bajo incluso con la reincorporación de cinco variables por refinamiento.

Capítulo 6
Conclusiones y trabajo futuro

En este capítulo presentamos un resumen de las contribuciones realizadas durante el desarrollo de esa investigación y las conclusiones a las que hemos llegado; además esbozamos, como posible trabajo futuro, algunas de las ideas que han surgido y que podrían ayudar a mejorar los resultados obtenidos.

6.1 Conclusiones

En esta investigación se propuso un marco de trabajo para agilizar la verificación de propiedades LTL. Este marco utiliza: reducción al cono de influencia, CEGAR, abstracción existencial por eliminación de variables y refinamientos basados en heurísticas. Algunas de las heurísticas de refinamiento utilizadas fueron propuestas

en este trabajo. Además se creo una extensión para el comprobador de modelos NuSMV, esta extensión implementa el marco de trabajo aquí propuesto.

En base al trabajo experimental realizado podemos concluir que la utilización de las técnicas aquí presentadas es una buena opción, especialmente para aquellos modelos para los cuales las técnicas tradicionales han fracasado.

Sugerimos su empleo en este tipo de modelos porque si bien, en la mayoría de los casos no se obtiene una ganancia en tiempo, existen casos en los que la reducción del espacio de estados hace la diferencia entre poder o no poder concluir exitosamente la verificación, esto debido a la reducción en el consumo de recursos que se obtiene al aplicar estas técnicas de abstracción.

Además podemos observar en los experimentos realizados utilizando técnicas de abstracción, que el número de variables reincorporadas en cada refinamiento es un factor que impacta el tiempo total de verificación pero no el tiempo promedio de los pasos de verificación.

Un punto importante a destacar es que el cambio en el número de variables reincorporadas por refinamiento tampoco impacta fuertemente el número de variables invisibles al final de la verificación, esto nos permite afirmar que la heurística presentada es bastante consistente.

Los factores referentes a la estructura estática del modelo, evaluados en nuestra heurística de refinamiento, nos brindan una ventaja adicional: es posible modificar fácilmente el tamaño de la abstracción inicial, para incluir en ella al menos un determinado porcentaje del modelo original Esto se logra calculando las

puntuaciones sin tomar en cuenta los factores evaluados en el SAT Solver, posteriormente se crea la abstracción inicial incluyendo en ella las variables del VOC(φ) y las variables con mayor puntuación inicial. El iniciar con modelo de determinado tamaño podría ser útil cuando se disponga de conocimiento adicional acerca de la estructura del modelo.

Un comportamiento común que no previmos es que el problema SAT generado para determinar la validez del contraejemplo, es en muchos casos trivialmente insatisfacible; entiéndase por trivialmente insatisfacible que con sólo sustituir las cláusulas unitarias y aplicar reglas de simplificación simples, es posible determinar que la fórmula es insatisfacible. El hecho de que sea trivialmente insatisfacible resulta entonces ser un inconveniente al tratar de determinar los puntajes de las variables para el proceso de validación del contraejemplo, puesto que en estos casos la puntuación será cero. Por otra parte tenemos que NuSMV para la generación de fórmulas SAT utiliza Circuitos Booleanos Reducidos los cuales, como su nombre lo indica, aplican una serie de simplificaciones al problema, llegando incluso en muchas ocasiones a determinar que el problema es insatisfacible sin siquiera requerir la intervención del SAT Solver. Tenemos así que los Circuitos Booleanos Reducidos y las heurísticas de refinamiento basadas en el proceso de verificación de satisfacibilidad son técnicas que no deberían usarse juntas sin antes contemplar que la primera puede imposibilitar la ejecución de las segundas.

6.2 Trabajo futuro

Hemos realizado algunos experimentos modificando las heurísticas de refinamiento y hemos encontrado que es posible obtener refinamientos más certeros agregando criterios que califiquen las variables invisibles en base a la actividad de las variables visibles; estas heurísticas añaden un pequeño costo computacional, sin embargo creemos que su desarrollo podría mejorar el rendimiento de los algoritmos actuales.

Otro punto a desarrollar es la utilización de los factores referentes a la estructura estática del modelo que se han evaluado en nuestra heurística de refinamiento, para el cálculo de mejores abstracciones iniciales, esto con el objetivo de reducir la cantidad de refinamientos necesarios.

Por último creemos que existe una mejor forma de generar los modelos abstractos cuando el sistema se especifica mediante ecuaciones de estado siguiente; la idea se describe a continuación:

En este trabajo una vez que se ha construido el modelo a partir de las ecuaciones de estado siguiente, procedemos a calcular el conjunto de variables inicialmente invisibles para posteriormente cuantificar existencialmente todas ellas, todas las operaciones descritas anteriormente consumen recursos. Para evitar este consumo podría construirse el modelo sólo tomando en cuenta aquellas ecuaciones de estado siguiente cuya parte izquierda sea una variable que pertenezca al conjunto de las variables que se desea sean visibles, esto convertiría en entradas a las variables estado de las cuales dependen las variables visibles, estas nuevas pseudo-entradas pueden luego ser cuantificadas existencialmente.

Con este procedimiento, al tomar en cuenta menos ecuaciones de estado siguiente, se reduciría el consumo de recursos durante la construcción del modelo. Además, al reducir el número de variables a cuantificar existencialmente, se reducirían también los recursos empleados en la fase de cuantificación existencial.

Apéndice A
Manual de usuario de XNuSMV

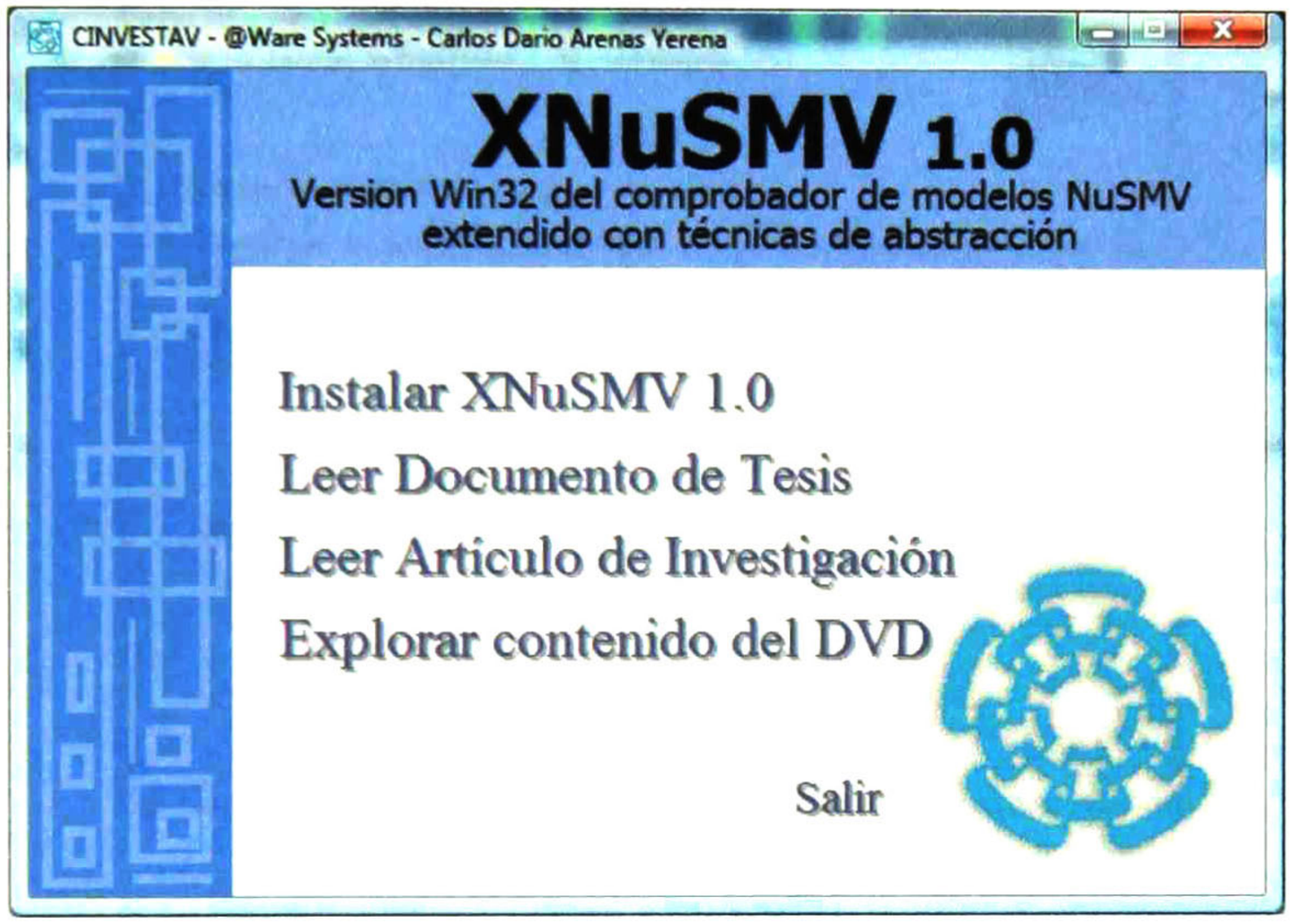
En este apéndice se incluyen las instrucciones de instalación de XNuSMV, además presentamos las instrucciones para la ejecución XNuSMV y las instrucciones para el uso de las técnicas de abstracción refinamiento incorporadas en XNuSMV.

XNuSMV

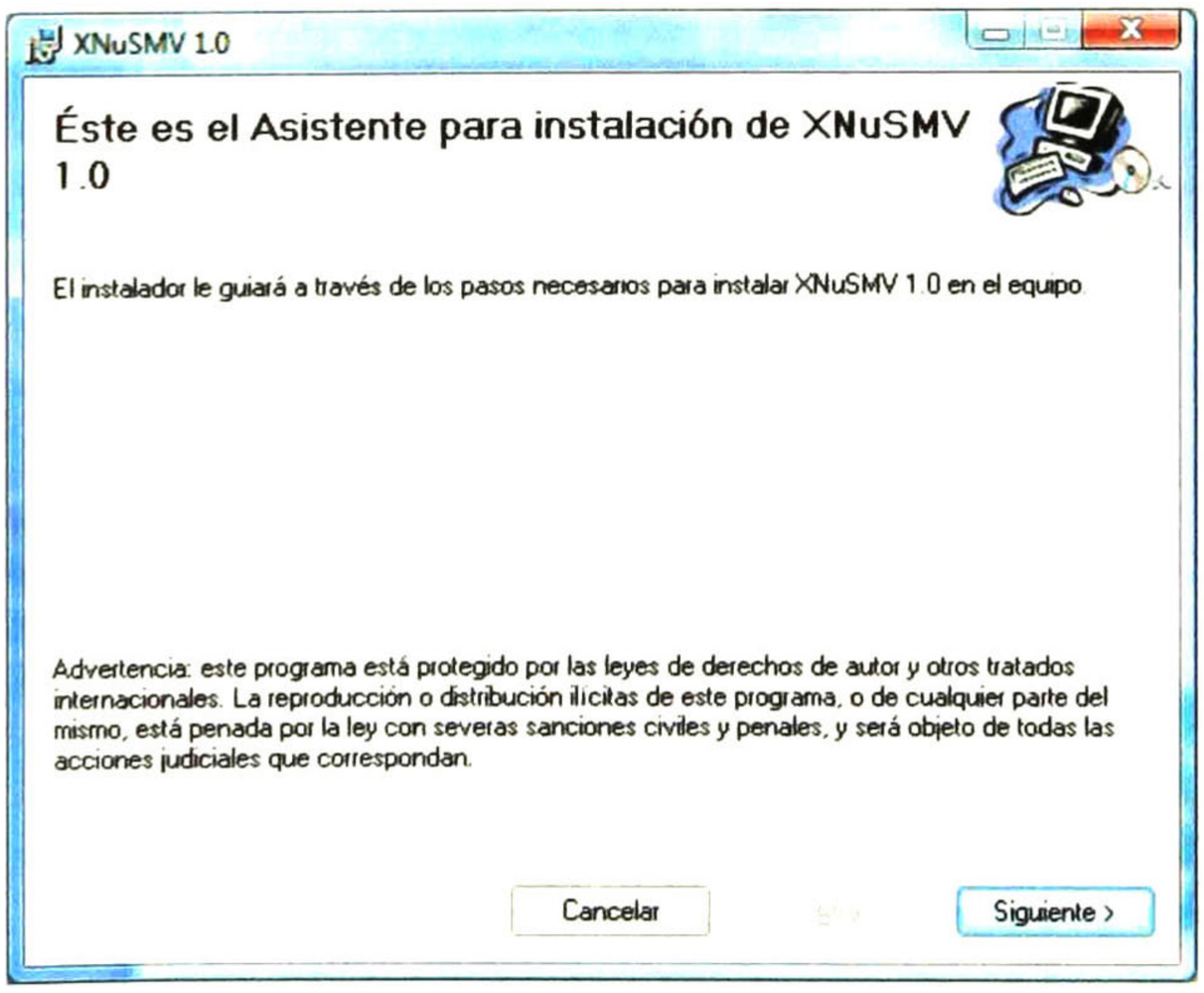
XNuSMV es la versión Win32 de comprobador de modelos NuSMV extendido con técnicas de abstracción-refinamiento. XNuSMV ha sido creado en CINVESTAV Jalisco, para ello hemos portado a Win32 la versión 2.4.3 NuSMV y hemos incorpora además una serie de algoritmos, opciones y comandos que permiten la aplicación de técnicas de abstracción-refinamiento (abstracción existencial de variables y refinamiento guiado mediante heurísticas).

Instalación de XNuSMV

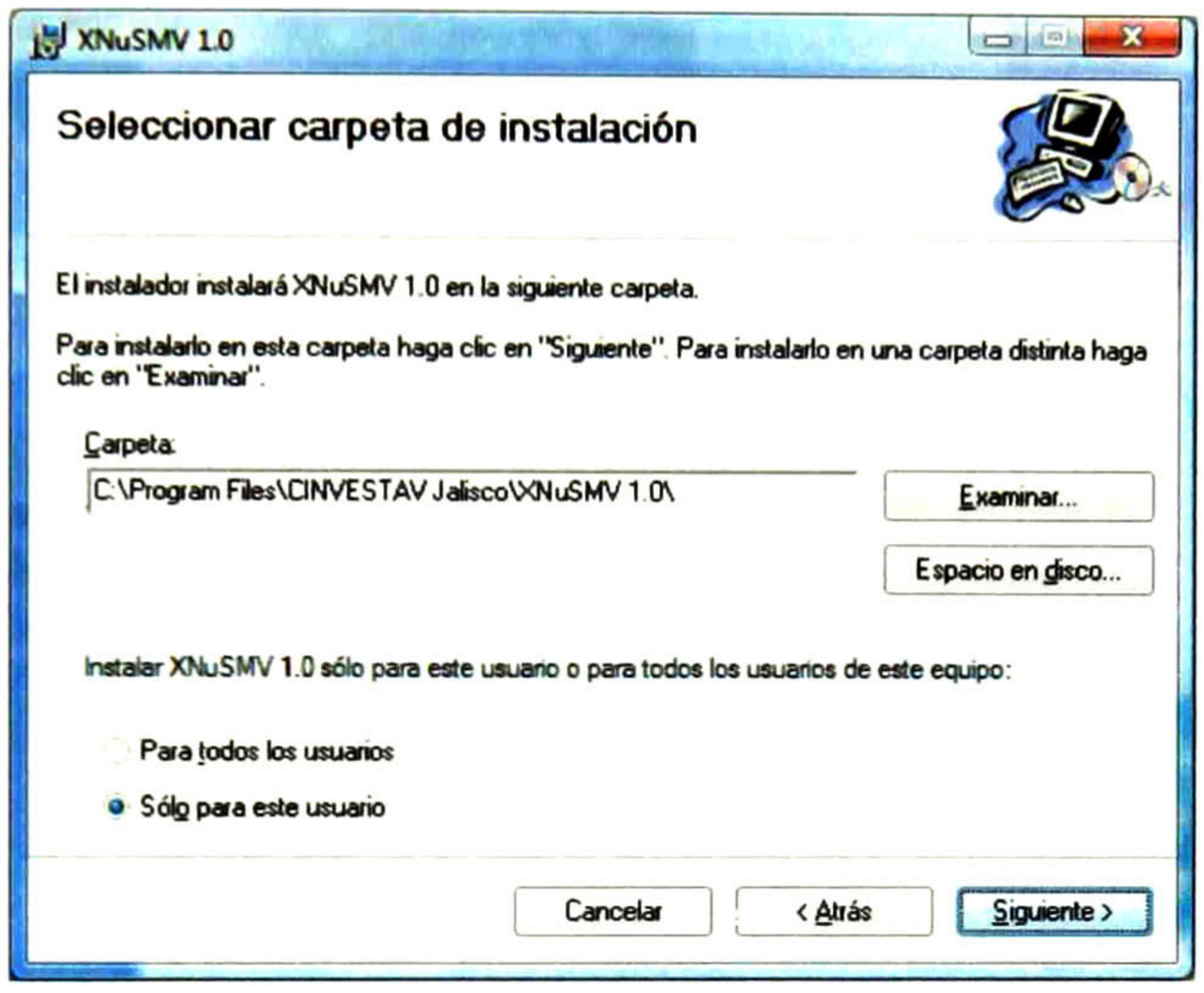
Introduzca su CD de instalación de XNuSMV, deberá aparecer el siguiente cuadro de dialogo:



Seleccione la opción "Instalar XNuSMV 1.0", al hacerlo se iniciara el asistente de instalación de XNuSMV y presentara el siguiente cuadro de dialogo:

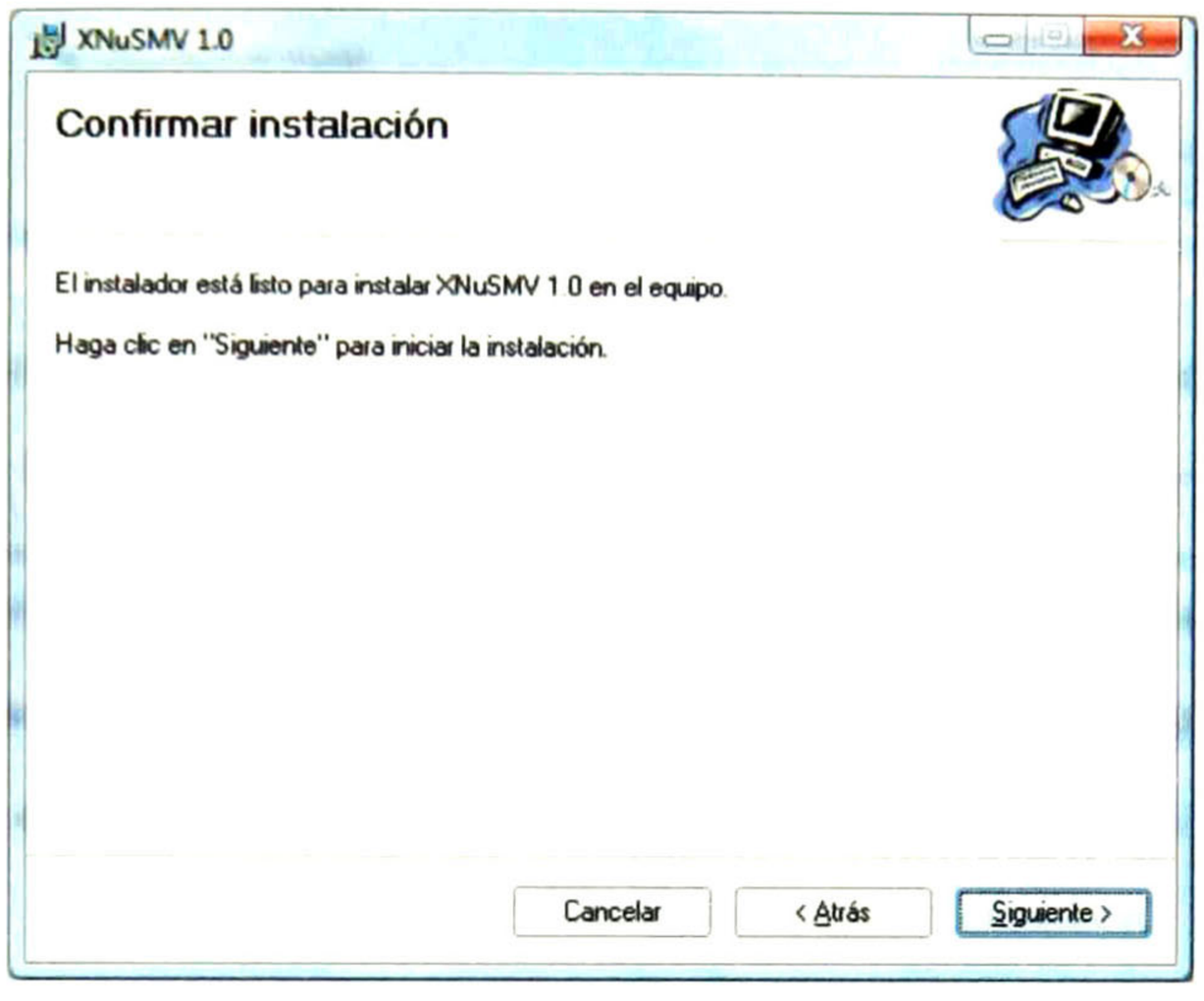


Una vez iniciado de asistente para la instalación de XNuSMV 1.0, presione el botón "Siguiente" para pasar al siguiente paso de la instalación, al hacerlo aparecerá el siguiente cuadro de dialogo:

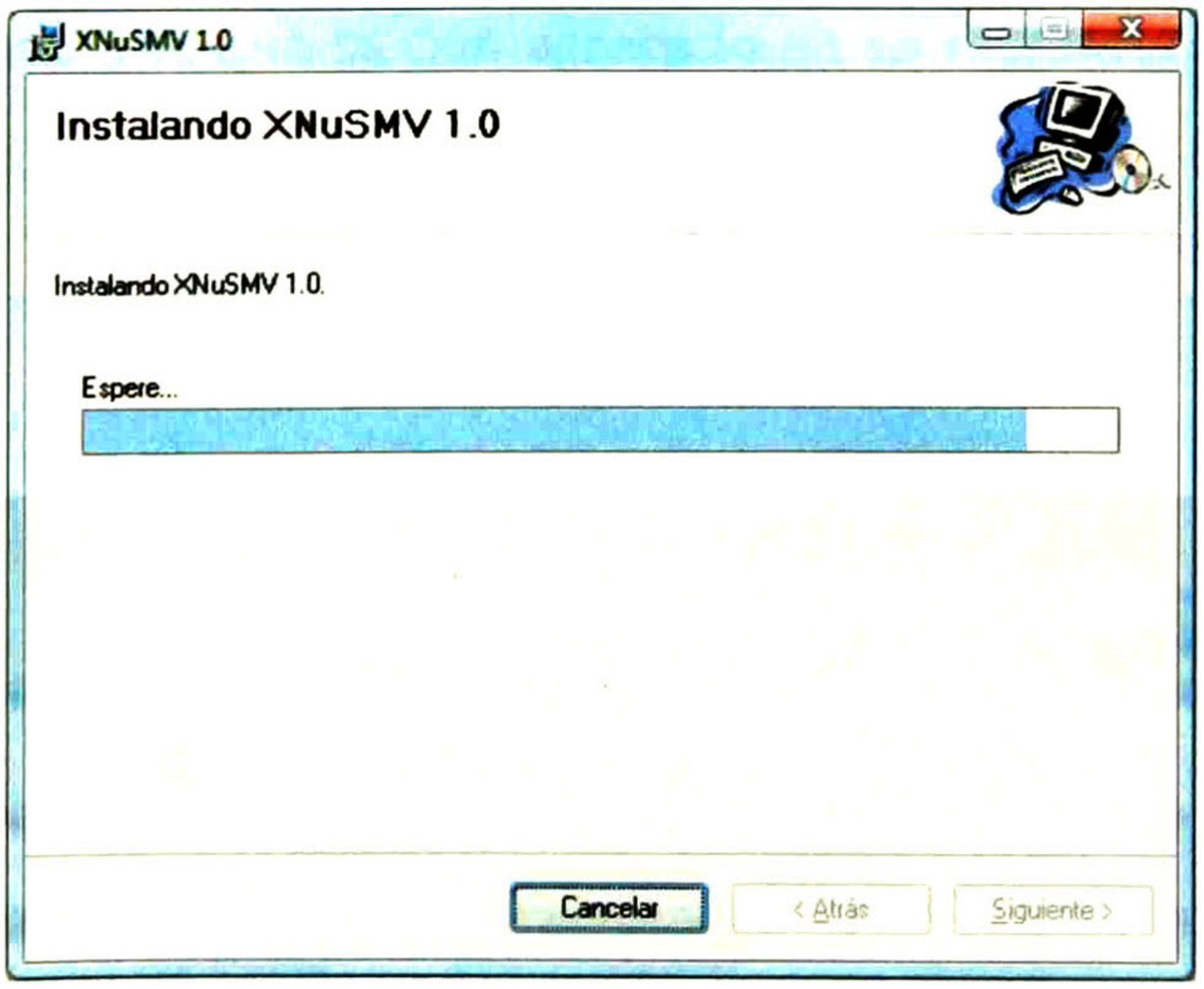


Seleccione la ruta donde desea instalar XNuSMV, por defecto se instalará en "Program Files\CINVESTAV Jalisco\XNuSMV". Si desea que XNuSMV este disponible para todos los usuarios de este equipo seleccione la opción "Para todos los usuarios"

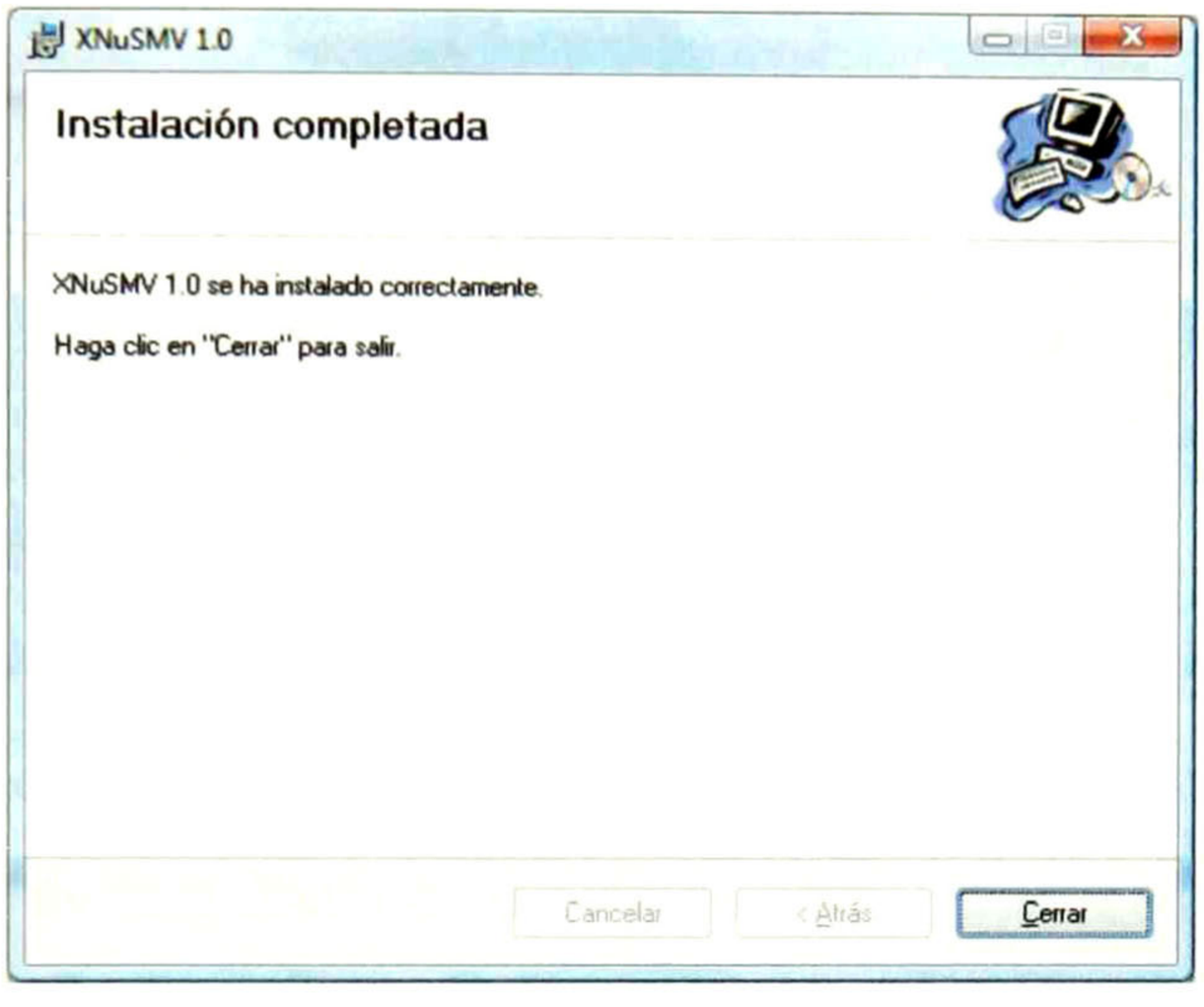
Presione el botón "Siguiente" para pasar al siguiente paso de la instalación, al hacerlo aparecerá el siguiente cuadro de dialogo:



Para iniciar la instalación de XNuSMV presione el botón "Siguiente", si desea modificar las opciones de instalación presiones el botón "Atrás". Si ha presionado el botón "Siguiente" aparecerá la siguiente ventana informando acerca del progreso de la instalación:



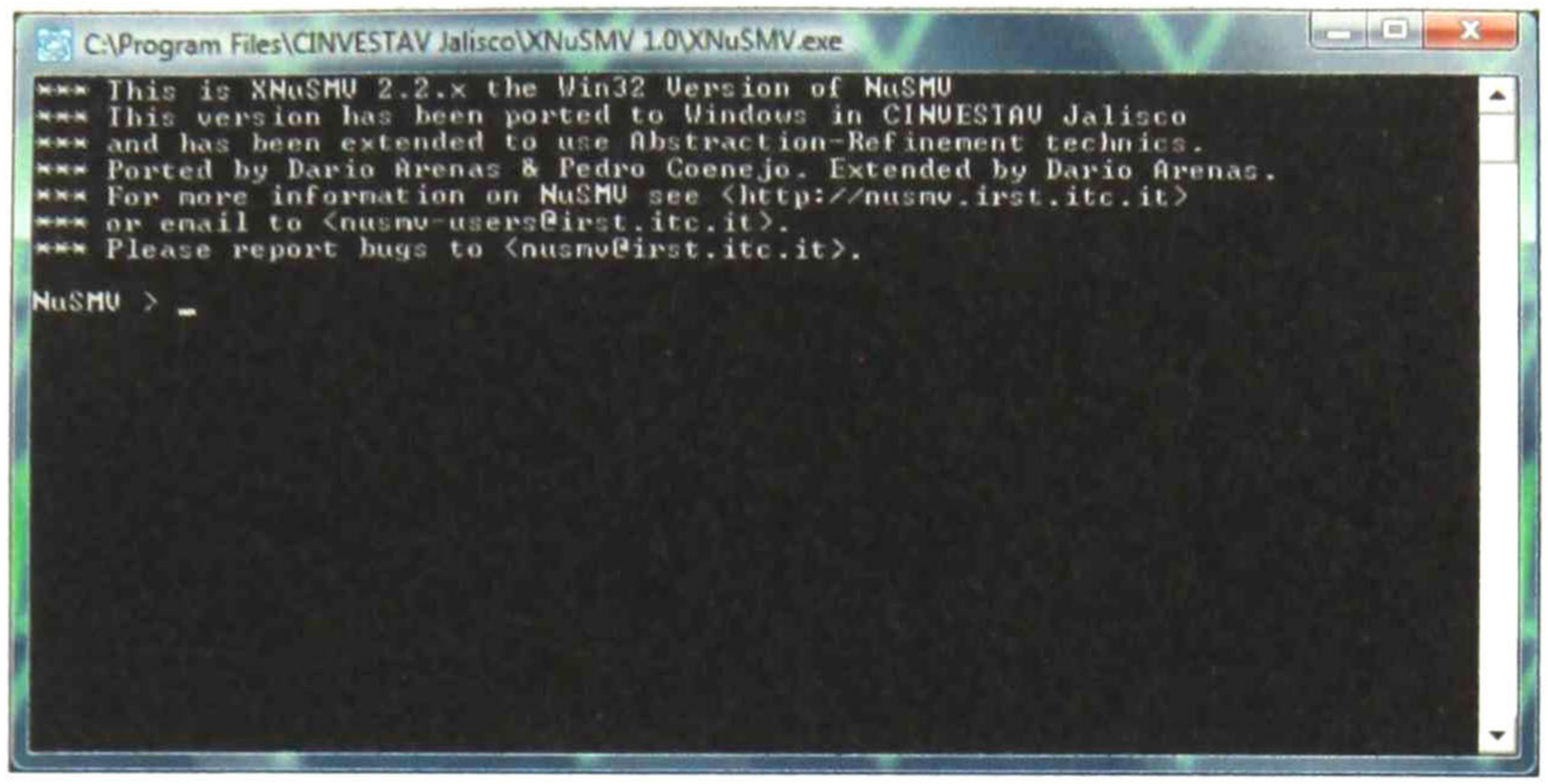
Una vez finalizada la instalación se cerrara automáticamente la ventana anterior y aparecerá el siguiente cuadro de dialogo informando que la instalación ha finalizado:



Presione el botón "Cerrar" para cerrar el asistente de instalación de XNuSMV. En este punto usted dispone de XNuSMV instalado en su computadora.

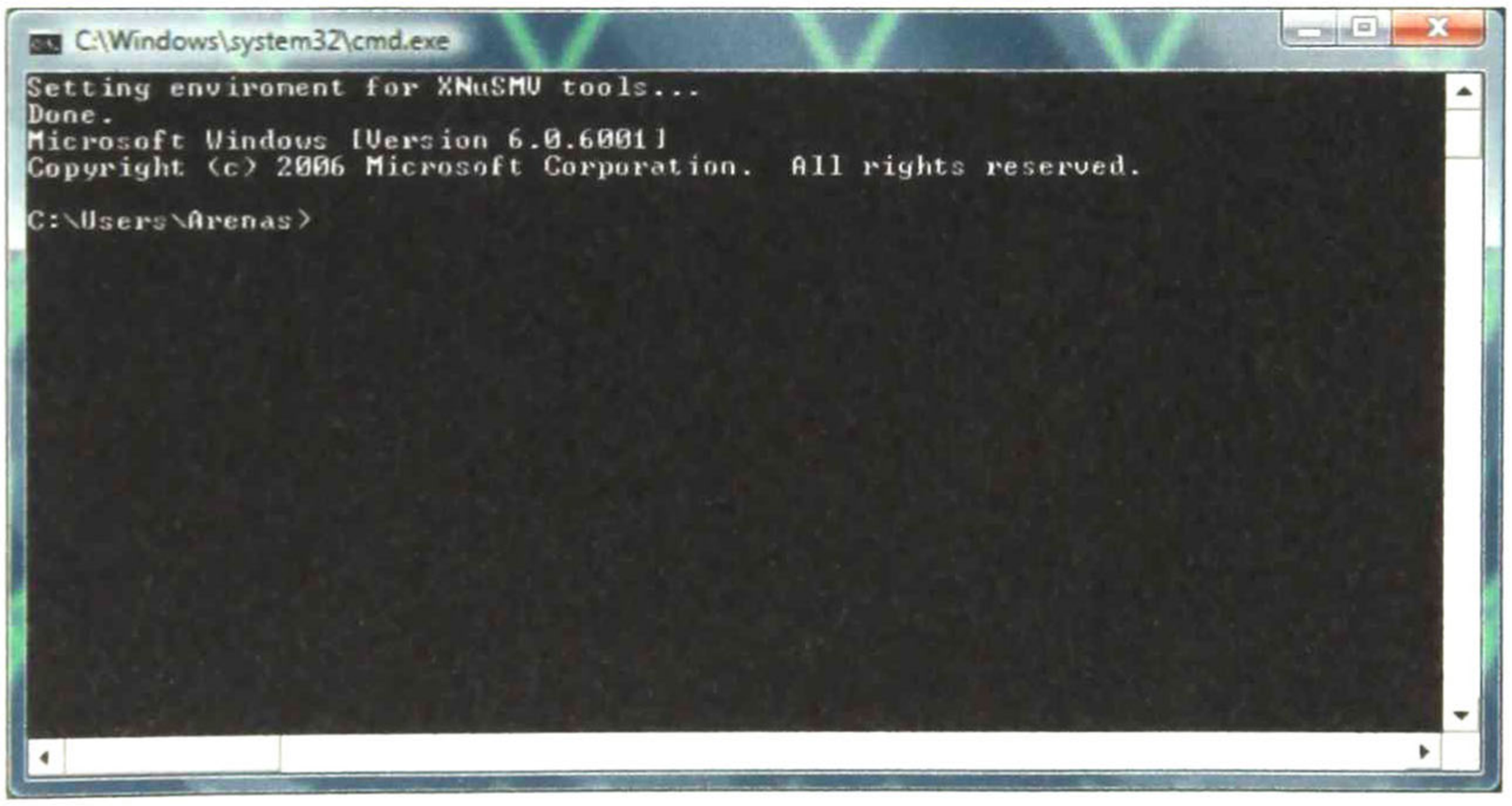
Para ejecutar XNuSMV identifique los iconos de **XNuSMV Interactive** y **XNuSMV Command Line** que se han agregado en su escritorio, estos también se encuentran disponibles en "Inicio->Todos los Programas->CINVESTAV Jalisco"

Al ejecutar **XNuSMV Interactive** aparecerá la siguiente ventana:



En ella podrá ejecutar los comandos de modo interactivo, para referencia de los comandos estándar de NuSMV lea el manual de usuario de NuSMV 2.4.3, para referencia de los comandos de AbsNuSMV vea la sección "Ejecutando NuSMV en modo interactivo" de este documento.

Al ejecutar **XNuSMV Command Line** aparecerá la siguiente ventana:



En ella podrá ejecutar el comando XNuSMV o cualquier otro comando válido, para referencia de las opciones de ejecución de NuSMV lea el manual de usuario de NuSMV 2.4.3, para referencia de las opciones de ejecución de AbsNuSMV vea la sección "Ejecutando NuSMV en modo batch" de este documento.

Ejecutando NuSMV en modo interactivo

Se ha agregado un nuevo comando para modo interactivo al conjunto de comandos de NuSMV este se describe a continuación en el formato de descripción de comandos usado en el manual de usuario de NuSMV.

Para ejecutar NuSMV en modo interactivo ejecute **XNuSMV Interactive,** o si lo prefiere ejecute **XNuSMV Command Line** e introduzca el siguiente comando:

`system prompt>` **XNuSMV** *–int [-AbsNumRefinementVars NUM]* `<RET>`

La opción -int indica que deberá ejecutarse en modo interactivo. La opción -AbsNumRefinementVars asigna la variable de entorno `Num_Refinement_Vars` al valor introducido en `NUM` esta opción afecta al comando **process_model_abstracting**, dado que este reincorporara `Num_Refinement_Vars` por refinamiento, el valor por defecto de `Num_Refinement_Vars` es uno.

process_model_abstracting – *Realiza la verificación del modelo simulando el modo batch y regresa el control al modo interactivo, este comando usa técnicas de abstracción- refinamiento.*

Comando

`Process_model_abstracting [-h][-i model-file]`

Lee el modelo y crea el modelo booleano, el modelo en BDDs, el modelo en circuito booleano reducido, posteriormente realiza la verificación usando técnicas de abstracción (abstracción existencial de variables) y refinamientos guiados mediante heurísticas. Este comando simula el modo batch de NuSMV para después devolver el control al modo interactivo.

Command Options:

`-i model-file`	Asigna la variable de entorno `input-file` al valor introducido en `model-file` y lee el modelo especificado en `model-file`.
`-h`	Muestra la ayuda de este comando.

check_ltlspec_abstracting – *Realiza la verificación de la propiedad LTL dada, en caso de no especificarse la propiedad verifica todas las propiedades LTL. Este comando usa técnicas de abstracción- refinamiento para la verificación de las propiedades.*
Comando

`Check_ltlspec_abstracting [-h] [-n number | -p"ltl-expr [IN context]"]`

Realiza la verificación LTL usando técnicas de abstracción refinamiento. Se pueda especificar una propiedad a verificar mediante la opción -p. Alternativamente es posible usar la opción -n para verificar la n-esima propiedad del conjunto de propiedades. Si no se ha especificado ninguna opción se verificaran todas las propiedades LTLSPEC.

Command Options:

`-p "ltl-expr [IN context]"`	Especifica la propiedad LTL a verificar, `context` especifica el modulo en donde deben evaluarse las variables de `ltl-expr`.
`-n NUM`	Verifica la propiedad n-esima del conjunto de propiedades.
`-h`	Muestra la ayuda de este comando.

Ejecutando NuSMV en modo batch

Cuando no se especifica la opción –int, NuSMV se ejecutará en modo interactivo, al estilo de SMV.

Para ejecutar NuSMV en modo batch ejecute **XNuSMV Command Line** e introduzca el siguiente comando:

`system prompt>` **`XNuSMV [options]`** *[-AbsNumRefinementVars NUM]*
[-Abs]input-file `<RET>`

La opción -AbsNumRefinementVars asigna la variable de entorno `Num_Refinement_Vars` al valor introducido en `NUM` esta opcion afecta al comando

process_model_abstracting, dado que este reincorporara `Num_Refinement_Vars` por refinamiento, el valor por defecto de `Num_Refinement_Vars` es uno.

Se procesa el archivo indicado en *input-file*, construyendo a partir de dicho archivo el modelo a verificar, imprime como salida el resultado de la verificación de cada una de las propiedades especificadas, si las propiedades no son verdaderas además imprime un contraejemplo que lo demuestra. Si no se especifica el *input-file* se leerá el modelo de la estrada estándar.

A continuación se describen las opciones agregadas al modo batch de NuSMV:

`-AbsNumRefinementVars NUM`	Asigna la variable de entorno de XNuSMV `Num_Refinement_Vars` al valor introducido en `NUM` esta opcion afecta al comando `process_model_abstracting`, dado que este reincorporara `Num_Refinement_Vars` por refinamiento, el valor por defecto de `Num_Refinement_Vars` es uno.
`-Abs`	Crea el modelo booleano, el modelo en BDDs, el modelo en circuito booleano reducido, posteriormente realiza la verificación usando técnicas de abstracción (abstracción existencial de variables) y refinamientos guiados mediante heurísticas.

Referencias

[BAR07] Christel Baier and Joost-Pieter Katoen.
Principles of Model Checking.
MIT Press. 2007.

[BRW87] M.C. Browne, Edmund M. Clarke and O. Grümberg.
Characterizing Kripke Structures in Temporal Logic.
TAPSOFT'87, LNCS Volume 249. pp: 256-270. 1987.

[BIR03] Armin Biere, Alessandro Cimatti, Edmund M. Clarke.
Bounded Model Checking.
Vol. 58 of Advances in Computers, 2003.

[ALL93] E. Allen Emerson, A. Prasad Sistla.
Symmetry and Model Checking.
Lecture Notes In Computer Science. pp: 463 - 478 . 1993

[FU06] Zhaohui Fu, Yogesh Mahajan, Sharad Malik.
zChaff SAT Solver.
In SAT-Race 2006, Institute for Formal Methods, Austria, 2006

[CHN02] Alessandro Cimatti, Edmund M. Clarke, E. Giunchiglia, F Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani and A. Tacchella.
NuSMV 2: An OpenSource Tool for Symbolic Model Checking.
Proceeding of International Conference on Computer-Aided Verification. pp: 27-31. 2002

[AML05] Nina Amla, Xiaoqun Du, Andreas Kuehlmann, Robert P Kurshan, Kenneth L Mcmillan.
An Analysis of SAT-Based Model Checking Techniques in an Industrial Environment.
Correct Hardware Design and Verification Methods. pp. 254-268. 2005.

[CHN02] Pankaj Chauhan, Edmund M. Clarke, James Kukula, Samir Sapra, Helmut Veith, Dong Wang.
Automated Abstraction Refinement for Model Checking Large State Spaces using SAT based Conflict Analysis.
Formal Methods in Computer Aided. 2002.

[HTH04] Michael Huth and Mark Ryan.
Logic in Computer Science: Modeling and Reasoning About Systems. Second Edition.
Cambridge University Press, 2004.

[CLK01] Edmund Clarke, Armin Biere, Richard Raimi, Yunshan Zhu.
Bounded model checking using satisfiability solving.
Formal Methods in System Design. Volume 19 pp: 7 – 34. 2001.

[CLK00] E. Clarke, O. Grumberg, S. Jha, Y. Lu, H. Veith.
Counterexample-guided Abstraction Refinement.
Lecture Notes in Computer Science. pp: 154-169. 2000.

[KRO97] Thomas Kropf.
Formal Hardware Verification.
Lecture Notes in Computer Science. 1997.

[EEN03] Niklas Eén and Niklas Sörensson.
An Extensible SAT-Solver (extended version 1.2).
Chalmers University of Technology, Sweden. SAT 2003. 2003.

[EEN05] Niklas Eén and Niklas Sörensson.
MiniSAT v1.13 A SAT Solver with Conflict-Clause Minimization.
Chalmers University of Technology, Sweden. SAT 2005. 2005.

[SHR99] Mary Sheeran and Gunnar Stålmarck.
A Tutorial on Stålmarck's Proof procedure for propositional logic.
Prover Technology AB and Chalmers University of Technology, Sweden. Kluwer Academic Publishers. 1999.

[JCK04] Paul Jackson and Daniel Sheridan.
The Optimality of a Fast CNF Conversion and its use with SAT.
Tech report at University of Edinburgh. 2004.

[CLK03] Edmund Clarke, Orna Grumberg, Somesh Jha, Yuan Lu, Helmut Veith.
Counterexample-guided abstraction refinement for symbolic model checking.
J. ACM, Vol. 50, No. 5. pp: 752-794. 2003.

[CLK04] Edmund Clarke, Muralidhar Talupur, Helmut Veith, Dong Wang. **Sat based predicate abstraction for hardware verification**. Theory and Applications of Satisfiability Testing. pp: 267-268. 2004.

[CLK02] Edmund Clarke, Anubhav Gupta, Ofer Strichman. **Sat based abstraction-refinement using ILP and machine learning techniques.** Proceedings of the 14th International Conference on Computer Aided Verification. pp: 265 – 279. 2002.

[CLK94] E. Clarke, O Grumberg, D. Long. **Model Checking and Abstraction**. ACM Transactions on Programming Languages and Systems Volume 16. pp: 1512 – 1542. 1994.

[GRU01] Orna Grumberg. **Abstractions and Reductions in Model Checking**. NATO Science Series, Vol. 62. 2001.

[GPT03] Gupta, A. Ganai, M. Zijiang Yang Ashar. **Iterative abstraction using SAT-based BMC with proof analysis.** Computer Aided Design. pp: 416- 423. 2003.

[HNZ04] Thomas A. Henzinger, Ranjit Jhala, Rupak Majumdar, and Ken McMillan. **Abstractions from Proofs**. Annual Symposium on Principles of Programming Languages. pp: 232-244. 2004.

[MCM92] Kenneth Lauchlin McMillan. **Symbolic model checking: an approach to the state explosion problem**. Carnegie Mellon University.

[MCM03] K. L. Mcmillan, Nina Amla. **Automatic abstraction without counterexamples**. Tools and Algorithms for the Construction and Analysis of Systems. pp: 2-17.

[MSK01] Moskewicz, Madigan, Zhao, Zhang, Malik. **Chaff: Engineering an Efficient SAT Solver**. Design Automation Conference. pp: 530-535 2001.

[NNN98] Andreas Nonnengart, Georg Rock, Christoph Weidenbach.
On Generating Small Clause Normal Forms.
International Conference on Automated Deduction. pp: 397-411. 1998.

[PRD98] Abelardo Pardo Sánchez.
Automatic abstraction techniques for formal verification of digital systems.
University of Colorado at Boulder. 1998.

[YI07] Jaeheon Yi.
The Effect of VSIDS on SAT Solver Performance.
University of California, Santa Cruz. 1998.

[ZHG03] Lintao Zhang. Thesis.
Searching for Truth: Techniques for Satisfiability of Boolean Formulas.
Princeton University, 2003.

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

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

Generación y Uso de Abstracciones en la Comprobación de Modelos Simbólica

del (la) C.

Carlos Dario ARENAS YERENA

el día 20 de Febrero de 2009.

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

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

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