Universidad Nacional Autónoma de México

Posgrado en Ciencia e Ingeniería de la Computación

Actualización de modelos para lógica de árboles de cómputo mediante protecciones

T E S I S

que para optar por el grado de:
Doctor en Ciencias (Computación)

PRESENTA:
Miguel Carrillo Barajas

Tutor principal:

Dr. David Arturo Rosenblueth Laguette.
Instituto de Investigaciones en Matemáticas Aplicadas y en Sistemas, UNAM.

Miembros del comité tutor:

Dra. Atocha Aliseda Llera.
Instituto de Investigaciones Filosóficas, UNAM.

Dr. Francisco Hernández Quiroz.
Facultad de Ciencias, UNAM.

México, D.F. Agosto de 2014

Actualización de modelos para lógica de árboles de cómputo mediante protecciones

por

Miguel Carrillo Barajas

Tesis presentada para obtener el grado de
Doctor en Ciencias (Computación)

en el

Instituto de Investigaciones en Matemáticas Aplicadas y en Sistemas

Universidad Nacional Autónoma de México

México, D.F. Agosto de 2014

Agradecimientos

Quiero expresar mi reconocimiento y gratitud a mi tutor principal, el Dr. David Rosenblueth, por compartir conmigo su conocimiento y experiencia en todos los aspectos del proceso de mi investigación. La honestidad y destreza del Dr. Rosenblueth han sido fundamentales en el desarrollo de dicho proceso, y me siento afortunado por haber sido aprendiz de su determinación y firmeza para librar obstáculos.

Agradezco a mis tutores y sinodales –la Dra. Atocha Aliseda y los Drs. Francisco Hernández, Favio Miranda, Pablo Padilla, y Sergio Rajsbaum– por sus comentarios y preguntas con una perspectiva enriquecedora.

Quiero agradecer también al departamento de Ciencias de la Computación, al Instituto de Investigaciones en Matemáticas Aplicadas y en Sistemas, y a la UNAM, por los recursos proporcionados generosamente para la realización de mi investigación.

Doy las gracias a la Facultad de Ciencias de la UNAM, y al Tecnológico de Monterrey, por darme la oportunidad de obtener un sustento económico durante mi investigación, trabajando como profesor.

También quiero agradecer a mis alumnos, y a mis profesores, por motivar mi deseo de aprender más.

Finalmente, agradezco a mi familia y amigos por apoyarme siempre.

Índice general

Actualización de modelos para lógica de árboles de cómputo mediante protecciones

por

Miguel Carrillo Barajas

Resumen

Presentamos un algoritmo recursivo no determinista para actualizar un modelo de Kripke a fin de satisfacer una fórmula de la lógica de árboles de cómputo (CTL, por sus siglas en inglés). Los algoritmos recursivos para la actualización modelos enfrentan dos dificultades duales: (1) *Eliminar* transiciones de un modelo de Kripke para satisfacer una subfórmula *universal* puede provocar la insatisfacción de algunas subfórmulas *existenciales*. Contrariamente, (2) *agregar* transiciones para satisfacer una subfórmula *existencial* puede provocar la insatisfacción de algunas subfórmulas *universales*. Para superar estas dificultades, utilizamos protecciones de la forma $\langle E, A, L\rangle$, que registran información sobre la satisfacción de subfórmulas previamente tratadas por el algoritmo. Intuitivamente, (1) E es el conjunto de transiciones que *no se pueden eliminar* sin comprometer la satisfacción de subfórmulas previamente tratadas. Por el contrario, (2) A es el conjunto de transiciones que *se pueden añadir*. Por lo tanto, el proceso de actualización procede sin disminuir E y sin aumentar A. Por último, (3) L es un conjunto de literales que protege a las etiquetas del modelo. Ilustramos el comportamiento de nuestro algoritmo a través de varios ejemplos: el problema de exclusión mutua de Emerson y Clarke, el ejemplo del horno de microondas de Clarke, contadores síncronos y modelos y fórmulas generadas aleatoriamente. Además, comparamos nuestro método con otros métodos de actualización, ya sea para CTL o para fragmentos de CTL. Por último, demostramos la corrección y completitud de nuestro algoritmo, y proporcionamos un análisis de su complejidad.

Model update for computation-tree logic through protections

by

Miguel Carrillo Barajas

Abstract

We present a nondeterministic, recursive algorithm for updating a Kripke model so as to satisfy a given formula of the computation-tree logic (CTL). Recursive algorithms for model update face two dual difficulties: (1) *Removing* transitions from a Kripke model to satisfy a *universal* subformula may dissatisfy some *existential* subformulas. Conversely, (2) *adding* transitions to satisfy an *existential* subformula may dissatisfy some *universal* subformulas. To overcome these difficulties, we employ protections of the form $\langle E, A, L\rangle$, recording information about the satisfaction of subformulas previously treated by the algorithm. Intuitively, (1) E is the set of transitions that we *cannot remove* without compromising the satisfaction of previously treated subformulas. Conversely, (2) A is the set of transitions that we *can add*. Hence, update proceeds without diminishing E and without augmenting A. Finally, (3) L is a set of literals protecting the model labels. We illustrate the behavior of our algorithm through several examples: Emerson and Clarke's mutual-exclusion problem, an example of a microwave oven of Clarke et al., synchronous counters, and randomly generated models and formulas. In addition, we compare our method with other update approaches for either CTL or fragments of CTL. Lastly, we prove soundness and completeness of our algorithm, and provide a complexity analysis.

Capítulo 1

Introducción

En esta tesis presentamos un método guiado por la semántica para actualizar modelos de Kripke respecto a fórmulas de la lógica de árboles de cómputo (CTL). A diferencia de otros métodos, este método es correcto, completo, y permite resolver el problema de actualización para modelos y fórmulas de la CTL no triviales. Nuestro método utiliza un mecanismo nuevo, al que denominamos "protecciones", diseñado para superar la dificultad asociada a la actualización de fórmulas que contienen tanto cuantificadores universales como existenciales. Este mecanismo de protecciones permite generalizar la semántica de la CTL y formalizar el concepto de modificación de un modelo. Usando esta formalización demostramos que nuestro método es tanto correcto como completo, e implementamos un prototipo que muestra en forma práctica que tiene un desempeño aceptable.

1.1. Antecedentes

Un verificador de modelos para lógica de árboles de cómputo (CTL) es una herramienta automatizada que generalmente tiene como entrada (1) un modelo de Kripke $\mathcal{M}$ que formaliza un sistema, (2) una fórmula CTL φ que expresa una propiedad deseable de este sistema, y (3) un conjunto de estados iniciales. La salida del verificador de modelos es una confirmación (o un rechazo) de que $\mathcal{M}$ satisface φ en todos los estados iniciales,

lo que significa respectivamente que el sistema tiene, o no tiene, la propiedad requerida. En caso de rechazo, los verificadores de modelos suelen producir un *contraejemplo* que consiste de una traza de error. Este contraejemplo pretende ser una guía para actualizar o reparar de forma manual el modelo $\mathcal{M}$, o una descripción de alto nivel de $\mathcal{M}$, para que la propiedad no satisfecha se cumpla.

1.2. El problema de actualización de modelos

El problema de actualización de modelos para la CTL consiste en lo siguiente: dados un modelo de Kripke M y una fórmula φ de la CTL, tales que M no satisface a φ, modificar automáticamente M para producir otro modelo, M', que satisfaga a φ. Usualmente, el modelo de entrada M es resultado de un conjunto de condiciones de diseño. Es decir, el modelo M fue diseñado de acuerdo a una especificación ψ esperando que, como consecuencia del diseño, M cumpliera una propiedad φ. Dos condiciones naturales para la modificación M' son las siguientes: (1) las modificaciones aplicadas a M son minimales, y (2) M' sigue satisfaciendo la especificación ψ.

Es importante notar que cualquier método recursivo para resolver el problema de actualización de modelos, respecto a fórmulas de la CTL, enfrenta dos dificultades duales: la eliminación (adición) de una transición puede causar que se pierda la satisfacción de subfórmulas existenciales (universales) previamente tratadas por el método.

Creemos que incluso una automatización parcial del proceso de actualización puede tener un impacto significativo en el uso de la técnica de verificación de modelos. Además, el grado de abstracción del problema de actualización de modelos para la CTL permite relacionar este problema con problemas en los que se utilizan modelos similares a los modelos de Kripke y lógicas similares a la CTL. Por ejemplo, el problema de actualización de modelos para la CTL está estrechamente relacionado con el problema de razonamiento acerca de acciones y cambio [e.g. 16]. Por lo tanto, la actualización de modelos es un tema importante en Inteligencia Artificial, a saber, la modificación de sistemas de transición.

1.3. Soluciones existentes

A pesar de su relevancia, el problema de actualizar automáticamente un modelo de Kripke se ha estudiado poco. Por lo que sabemos, Buccafurri *et al.* [2] propusieron el primer trabajo sobre actualización CTL de modelos: un actualizador para reparar, por medio de razonamiento abductivo, modelos de Kripke con la adición y la eliminación de transiciones. En un trabajo posterior, Calzone *et al.* [3] dieron un método para actualizar modelos de Kripke con la adición y la eliminación no sólo de transiciones, sino de etiquetas, dependiente de sesgos determinados por el dominio de aplicación (redes bioquímicas). Más recientemente, Zhang y Ding [26] idearon un algoritmo de reparación que produce modelos "admisibles".

En general, las soluciones del problema de actualización de modelos, anteriores a nuestro método, son métodos *basados en contraejemplos*. Estos métodos, para reparar un modelo M que no satisface una propiedad universal P, operan *invalidando* el contraejemplo C producido por el verificador, i.e. estos métodos transforman M en un modelo M' de forma tal que el verificador, aplicado a M', ya no produce el contraejemplo C. El uso de contraejemplos en el problema de actualización de modelos, sin embargo, tiene inconvenientes descritos a continuación.

Por un lado, si P es una propiedad universal, la invalidación de C no garantiza que el modelo reparado satisfaga P, porque P puede tener más de un contraejemplo. Además, todos los contraejemplos de P deben ser tratados simultáneamente, pero los verificadores de vanguardia (e.g. NuSMV [11]) producen solamente un contraejemplo a la vez.

Por otro lado, solamente las propiedades universales pueden tener contraejemplos. Si M no satisface una propiedad *existencial* P, el verificador de modelos no proporciona información útil para reparar M.

En el fondo, el concepto de contraejemplos para fórmulas CTL requiere un tratamiento más profundo. Por ejemplo, Buccafurri *et al.* [2, p. 25] observan que en muchos casos un contraejemplo es (esencialmente) una trayectoria, pero hay casos en los que se requiere un *árbol contraejemplo* [2, 13]. Sin embargo, los árboles contraejemplo, al igual que las

trayectorias contraejemplo, tienen los inconvenientes que ya hemos señalado.

Por lo tanto, los inconvenientes anteriores invitan a considerar un método de actualización basado en un concepto distinto al de contraejemplos.

1.4. Contribuciones

En esta tesis presentamos un método recursivo, no determinista, para reparar modelos respecto a fórmulas CTL. Este método no está basado en contraejemplos, está basado en la preservación de la satisfacción de subfórmulas a través de un mecanismo de "protecciones". Para actualizar un modelo con respecto a una fórmula φ, nuestro método actualiza recursivamente el modelo para satisfacer las subfórmulas de φ. Cada vez que nuestro método actualiza un modelo para satisfacer una subfórmula α, la satisfacción de α se protege. Esta protección asegura que, si α es una subfórmula propia de β, entonces una actualización para satisfacer β no causa que se pierda la satisfacción de α alcanzada con una actualización anterior. Para facilitar el tratamiento de fórmulas con ocurrencias del operador de negación, nuestro algoritmo transforma las fórmulas de entrada a una *forma normal de negación* que utiliza un conjunto de operadores *cerrado bajo dualidad*.

La evolución de las contribuciones de esta tesis aparecen en varios trabajos [4, 5, 6, 7, 8].

En [6] estudiamos el comportamiento de una herramienta de software que implementa nuestro algoritmo utilizando el problema de exclusión mutua de Emerson y Clarke [17]. En [8] hemos extendido ese estudio para incluir más ejemplos: el modelo de un horno de microondas de Clarke [12] ilustrando una especificación, un contador escalable exhibiendo modelos más grandes, y modelos y las fórmulas generados al azar que también muestran modelos más grandes. Carrillo y Rosenblueth [6] tiene un esbozo de demostraciones de la corrección y la completitud de nuestro método, mientras que Carrillo y Rosenblueth [8] tiene pruebas más detalladas, y muestra un análisis de la complejidad de nuestro algoritmo. La comparación superficial con dos métodos en Carrillo y Rosenblueth [6] se detalla y amplía en [8] con una revisión de las publicaciones más relevantes sobre

actualización de modelos.

En un trabajo anterior [5] extendimos una antigua versión de nuestro método con el fin de modificar una representación concisa de un modelo de Kripke en lugar de modificar el modelo mismo. Sin embargo, la versión antigua de nuestro método utiliza una forma más primitiva de protección universal (ver el capítulo 2). Esta protección primitiva, a diferencia de la que se utiliza aquí, es insatisfactoria porque no es claro si el algoritmo resultante es completo.

1.5. Alcances y limitaciones

El problema de actualización de modelos de la CTL tiene complejidad exponencial respecto al tamaño de la fórmula (ver la Sección 3.8). La solución que proponemos para este problema muestra que, a pesar la complejidad exponencial, en varios casos prácticos es posible actualizar un modelo con un tiempo de ejecución aceptable. Los alcances de nuestro método se vislumbran en los buenos tiempos de ejecución para los ejemplos estudiados (ver el capítulo 4).

Para atacar con más facilidad el problema de la actualización de modelos, asumimos que, en lugar de usar una descripción concisa de alto nivel, el modelo de entrada se proporciona con una descripción directa, enumerando sus estados, transiciones y etiquetas. Nuestro método modifica esta descripción directa del modelo, no modifica una descripción de alto nivel. Por lo tanto, asumimos que una versión futura de nuestro método podrá reflejar en una descripción de alto nivel las modificaciones que actualmente se realizan en la descripción directa.

El prototipo actual de nuestro método (Url: 4-1) utiliza una representación no simbólica del modelo de entrada, listas de estados, transiciones y etiquetas. Debido a esto, el tamaño máximo de modelo al que se puede aplicar este prototipo está limitado actualmente por la capacidad de memoria. Esta limitación se puede atenuar significativamente agregando al prototipo una representación simbólica del modelo de entrada.

1.6. Análisis

En el análisis de nuestro método para actualizar modelos de Kripke respecto a fórmulas de la CTL, encontramos tanto ventajas teóricas como prácticas.

En el aspecto teórico, construimos una especificación formal (Sección 3.5.1) que permitió: definir pseudocódigo para nuestro método (Sección 3.5.2), demostrar propiedades teóricas importantes como corrección y completitud (Sección 3.7), analizar la complejidad de nuestro algoritmo en el peor de los casos (Sección 3.8).

En el aspecto práctico, una virtud de nuestra especificación formal (Sección 3.5.1) es que permitió programar un prototipo para realizar pruebas de desempeño (Capítulo 4). En una de estas pruebas, encontramos que nuestro método de actualización, sin ser un método diseñado para síntesis de modelos, puede "sintetizar", partiendo de un modelo ficticio (dummy), el modelo de un ejemplo clásico con un tiempo de ejecución muy bueno (Sección 4.1). En otras pruebas de desempeño, encontramos que nuestro método, a pesar de que actualmente no utiliza una representación simbólica para los modelos, es capaz de actualizar modelos con cientos de estados, respecto a fórmulas de tamaño no trivial, con tiempos de ejecución aceptables (Secciones 4.3, 4.4).

Nuestro método resultó ser mejor que otros métodos de actualización en varios aspectos. Un aspecto que da ventajas a nuestro método es que la búsqueda se restringe a modelos con vocabularios que incluyen un conjunto de estados fijo (Sección 3.2). Aunque asumir vocabularios de este tipo es una idea básica, existen implicaciones, teóricas y prácticas, significativas. Por ejemplo, estos vocabularios dan lugar a un concepto no tradicional de satisfactibilidad (Definición 3.7.10) y, en la práctica, permiten separar el uso de operaciones que agregan estados nuevos al modelo. Otros métodos de actualización no utilizan vocabularios con un conjunto de estados fijo y complican, consecuentemente, el tratamiento del problema de actualización.

Otro aspecto que da ventajas a nuestro método es que usa una variante de la CTL con fórmulas construidas a partir de una base de operadores duales (Sección 3.2). Estas bases de operadores duales dan simplicidad y claridad a las definiciones, al pseudocódigo,

y a las demostraciones. Otros métodos, por ejemplo el método de Zhang y Ding [26], sólo utilizan parcialmente operadores duales, no utilizan la dualidad entre los operadores *Until* y *Release*. Otra característica de nuestra variante de la CTL es que usa operadores *OD* para restringir el número de sucesores de un estado (Sección 3.2). Los operadores *OD* ayudan a reducir considerablemente los tiempos de ejecución en casos donde el número de estados es grande pero se sabe que cada estado tiene un número de sucesores limitado (e.g. Sección 4.1).

Otro aspecto que da ventajas a nuestro método es que se enfoca en el fragmento modal de la CTL (Sección 3.2.1) y para el resto de la CTL utiliza caracterizaciones de punto fijo (ecuaciones 3-3). Este aspecto, como el uso de operadores duales, da ventajas de simplicidad y claridad. Otros métodos de actualización no utilizan estas caracterizaciones de punto fijo.

Finalmente, otra ventaja de nuestro método es que, a diferencia de otros métodos, cuenta con demostraciones de corrección y completitud (Sección 3.7). La ventaja que dan estas demostraciones se traduce en una mayor confianza en el método. Esta confianza es importante, por ejemplo, cuando el método se aplica en actualizaciones que requieren mucho tiempo de ejecución; si el método es incompleto la espera puede resultar inútil.

1.7. Estructura de la tesis

Después de la introducción de este capítulo, en el Capítulo 2 presentamos un resumen de los trabajos más relevantes para el problema de actualización de modelos respecto a fórmulas de la CTL. El Capítulo 3 contiene los aspectos teóricos de nuestro método. En la Sección 3.2.1 mostramos la sintaxis de la Σ-CTL, una variante de la CTL con una base de operadores cerrada bajo dualidad. En la Sección 3.2.2 proponemos el concepto de protección y vemos cómo este concepto permite definir una semántica con protecciones para la Σ-CTL y formalizar posteriormente el concepto de modificación de modelos protegidos (Sección 3.4.1). Utilizando pseudocódigo no determinista (Sección 3.3),

mostramos un algoritmo de actualización directa en la Sección 3.4, y en la Sección 3.5 mostramos un algoritmo que implementa la actualización de modelos protegidos. En la Sección 3.6 discutimos el tema del uso de heurísticas para eliminación del no determinismo. Demostramos que nuestro método es correcto y completo en la Sección 3.7, y mostramos un análisis de su complejidad en la Sección 3.8. En el Capítulo 4, utilizando dos prototipos de nuestro método de actualización CTL mediante protecciones, mostramos los resultados de varias pruebas de desempeño. Finalmente, el Capítulo 5 proporciona un resumen de las conclusiones de este trabajo y esboza posibles direcciones de investigación futura.

Capítulo 2

Actualización de modelos

Este capítulo contiene un resumen de los trabajos publicados que son relevantes para el problema de actualización de modelos respecto a fórmulas de la CTL.

A continuación, comparamos nuestro trabajo con otros enfoques, tanto para la CTL completa como para fragmentos de la CTL. Cubrimos estos enfoques en orden cronológico de publicación.

2.1. Buccafurri et al. 1999

Buccafurri *et al.* [2] desarrollan un método de actualización, empleando contraejemplos de tipo árbol (treelike) para generar un menor número de modificaciones posibles que las que genera un método ingenuo. En general, las propiedades existenciales no se pueden refutar mediante contraejemplos. Por lo tanto, este método está limitado a ACTL, el fragmento universal de CTL donde sólo se permite aplicar cuantificadores de trayectoria universales, **A**, y la aplicación del operador de negación se limita a fórmulas atómicas (variables).

El método de Buccafurri et al. modifica programas integrados por grupos de procesos que corren en paralelo. Estos programas se modelan como modelos de Kripke con una relación de accesibilidad asíncrona que tiene entrelazado (interleaving) y procesos que

están sincronizados con un planificador equitativo (fair scheduler). Estos autores asumen un mapeo que relaciona un programa con su modelo de Kripke correspondiente. Tal mapeo sólo permite modificaciones a nivel del modelo de Kripke que corresponden a modificaciones a nivel del programa y que pertenecen a un repertorio de modificaciones que respetan las restricciones de equidad. A nivel de programas, las modificaciones son de tres tipos: (a) negación del lado derecho de una instrucción de asignación que tiene o "true" o "false" en el lado derecho, (b) cambio de la variable en el lado izquierdo de una instrucción de asignación, y (c) el intercambio de dos instrucciones de asignación consecutivas. A nivel del modelo de Kripke, estas modificaciones corresponden a la adición y eliminación de transiciones.

Dado un contraejemplo, sólo las instrucciones correspondientes a estados que ocurren en tal contraejemplo se consideran ya sea para cambios en el lado derecho de asignaciones o para intercambio de instrucciones. Del mismo modo, sólo las variables que ocurren en contraejemplos se consideran para cambios en el lado izquierdo de asignaciones. Buccafurri *et al.* [2] ilustran su método en un ejemplo de exclusión mutua, asumiendo un solo error. Un método de reparación ingenua hace 77 intentos de corrección, mientras que el método que proponen hace sólo 17.

Como conclusión, el método de Buccafurri et al. se centra en el uso de contraejemplos de tipo árbol para reducir el número de intentos de corrección de un método ingenuo de reparación.

2.2. Calzone et al., 2006

Calzone *et al.* [3] presentan un sistema de modelado, Biocham, capaz de traducir una red bioquímica N en un modelo de Kripke $\mathcal{M}_N$. Si φ es una fórmula CTL que expresa una propiedad de N, y $\mathcal{M}_N$ no satisface a φ, entonces, en algunos casos de φ, Biocham puede generar una actualización de N, N', para que $\mathcal{M}_{N'}$ satisfaga a φ.

El algoritmo de actualización de Biocham procede de acuerdo con una clasificación

de CTL en tres tipos de fórmulas: las fórmulas *universales* contienen sólo operadores universales no negados, las fórmulas *existenciales* contienen sólo operadores existenciales no negados, y las fórmulas *no clasificadas* contienen ambos operadores, universales y existenciales.

Si φ es universal, entonces Biocham utiliza NuSMV [11] para calcular un *contraejemplo*. Luego, Biocham genera y prueba modelos que resultan de *eliminar transiciones* en dicho contraejemplo. Si φ es existencial, entonces Biocham genera y prueba modelos que resultan de *agregar transiciones* usando un *sesgo* tomado del dominio de aplicación. Si φ es no clasificada, entonces Biocham trata φ de las dos maneras, eliminando y agregando transiciones. Sin embargo, la eliminación (adición) de transiciones para satisfacer fórmulas universales (existenciales) puede provocar que se pierda la satisfacción de algunas fórmulas existenciales (universales) o no clasificadas. Por lo tanto, tratando de satisfacer los tres tipos de fórmulas, Biocham *utiliza una heurística*: primero trata las fórmulas existenciales, a continuación, las no clasificadas, y, finalmente, las universales. Si la satisfacción de alguna de estas fórmulas se pierde por el último paso, entonces el proceso se repite.

Por un lado, un inconveniente de Biocham, en comparación con nuestro método, es que el uso de heurísticas lo convierte en un método incompleto. Otra desventaja es que, debido al uso de sesgos dependientes del dominio, no es un método general. Por otra parte, ya que NuSMV representa modelos mediante diagramas de decisión binarios ordenados (OBDDs) que son generalmente compactos [1], Biocham es capaz de procesar modelos grandes [9]. Esta es una ventaja significativa de Biocham en comparación con $\textsc{Upd}_{prot}$ o con cualquier otro programa de actualización extensional, que no use ningún método simbólico para representar los modelos.

2.3. Zhang y Ding, 2008

Zhang y Ding [26] idearon un método de actualización de modelos, recursivo y dirigido por la sintaxis, para satisfacer fórmulas CTL empleando "restricciones" (constraints). Las

restricciones aparecen en el tratamiento de conjunciones $\alpha \wedge \beta$, que se realiza como sigue.

Primero (línea 03 de la Figura 2-1), el algoritmo de Zhang y Ding actualiza el modelo de entrada respecto a α. Luego (línea 04 de la Figura 2-1), este algoritmo actualiza los modelos obtenidos en el tratamiento de α para producir modelos que satisfagan β, usando α como una restricción.

```
* function Update_∧((M, s_0), φ_1 ∧ φ_2) *
input: (M, s_0) and φ_1 ∧ φ_2, where M = (S, R, L), s_0 ∈ S, and (M, s_0) ⊭ φ_1 ∧ φ_2;
output: (M', s'_0), where M' = (S', R', L'), s'_0 ∈ S' and (M', s'_0) ⊨ φ_1 ∧ φ_2;
01   begin
02     if φ_1 ∧ φ_2 is a propositional formula, then (M', s'_0) = Update_prop((M, s_0), φ_1 ∧ φ_2);
03     else (M*, s*_0) = CTLUpdate((M, s_0), φ_1);
04          (M', s'_0) = CTLUpdate((M*, s*_0), φ_2) with constraint φ_1;
05     return (M', s'_0);
06   end
```

Figura 2-1: Update$_\wedge$ de Zhang y Ding [26, p. 141].

El tratamiento de las restricciones se describe en la Figura 2-2: Los modelos candidatos son simplemente verificados (model-checked) respecto a las restricciones de entrada. Si se encuentra un modelo que satisfaga las restricciones, tal modelo se devuelve; en caso contrario, el algoritmo de Zhang y Ding busca otro modelo.

> "[...] suppose $\mathcal{C}$ is the set of domain constraints for a system specification $M = (S, R, L)$, and we need to update (M, s_0) with formula ϕ, where $s_0 \in S$, and $\mathcal{C} \cup \{\phi\}$ is satisfiable. Then in each function of $CTLUpdate$, we simply add a model checking condition on the candidate model $M' = (S', R', L')$: $(M', s'_0) \models \mathcal{C}$ $(s'_0 \in S')$. The result (M', s'_0) is returned from the function if it satisfies $\mathcal{C}$. Otherwise, the function will look for another candidate model."

Figura 2-2: Manejo de restricciones de Zhang y Ding's [26, p. 143].

Comparemos el tratamiento de la conjunción del procedimiento Update$_\wedge$ de Zhang y Ding (Fig. 2-1) con nuestro algoritmo generar-y-probar, XUPD$_1$, de la Sección 3.4. Observe que Update$_\wedge$ emplea la primera fórmula de $\alpha \wedge \beta$, α, como una restricción para actualizar repetidamente respecto a β hasta encontrar un modelo candidato que satisface a α. Note

que, cuando se trata una conjunción $\alpha \wedge \beta$, nuestro XUPD$_1$ verifica (model-checks) los modelos candidatos respecto a $\alpha \wedge \beta$ (línea 15), y sólo devuelve los modelos que satisfacen $\alpha \wedge \beta$. Por tanto, XUPD$_1$ también verifica (model-checks) respecto a α los modelos que fueron obtenidos al tratar β. En consecuencia, los tratamientos de Update$_\wedge$ y XUPD$_1$ a fórmulas de la forma $\alpha \wedge \beta$ son similares. En comparación, al tratar $\alpha \wedge \beta$, UPD$_{prot}$ no verifica (model-checks) que una actualización respecto a β satisface a α.

Zhang y Ding [26, Theorem 8] afirman que "CTLUpdate$((M, s_0), \phi)$ terminates and generates an admissible model to satisfy ϕ". Sin embargo, estos autores no garantizan que CTLUpdate$((M, s_0), \phi)$ genera todos los posibles modelos "admisibles" [20]. En contraste, demostramos que XUPD$_{prot}$ es correcto y completo (Sección 3.7). Además, el algoritmo de Zhang y de Ding es menos claro que UPD$_{prot}$ ya que utiliza una base de operadores ($\{\mathbf{EX}, \mathbf{AF}, \mathbf{EU}\}$) más adecuada para verificación de modelos (model-checking) que para actualización de modelos. Por ejemplo, siguiendo paso a paso el pseudocódigo proporcionado por los autores [26], no es claro que dicho algoritmo se pueda aplicar para actualizar un modelo respecto a fórmulas de la forma $\mathbf{AX}\,\alpha$, $\mathbf{EF}\,\alpha$, o $\mathbf{A}[\alpha\,\mathbf{U}\,\beta]$.

2.4. Carrillo y Rosenblueth, 2009

En otro trabajo [5], extendemos una versión anterior de UPD$_{prot}$ con el fin de modificar una representación concisa de un modelo Kripke, en lugar de modificar el modelo de Kripke. Esta antigua versión de UPD$_{prot}$ también se basa en protecciones, pero utiliza una protección universal diferente de la que desarrollamos aquí. A diferencia de la protección universal que usamos actualmente, definida con un conjunto de transiciones que se pueden añadir, la protección universal de la antigua versión de UPD$_{prot}$ usa un conjunto de pares (estado, fórmula) destinados a preservar la satisfacción de subfórmulas **AX** ya tratadas. Recordemos que para que $\mathbf{AX}\,\alpha$ se cumpla en s_0, α debe cumplirse en todos los sucesores de s_0. Registrando de un par de la forma (s_0, α) se indica que los sucesores de s_0 deben cumplir α. Así, es posible acompañar la adición de un estado s' a los sucesores de s_0 con

una invocación al algoritmo de actualización de manera que α se cumpla en el nuevo sucesor de s_0, s'. Por lo tanto, el registro de (s_0, α) permite que este método evite que el modelo se actualice nuevamente respecto a α en los antiguos sucesores, como sucedería en un método de generar-y-probar. Las protecciones utilizadas en Carrillo y Rosenblueth [5], sin embargo, no son satisfactorias porque no es claro si el algoritmo resultante es completo.

Para lograr un buen escalamiento, este método estado-por-estado se extiende con una representación concisa de modelos de Kripke, la cual se puede ver como una simplificación del lenguaje para descripción de modelos de SMV. El siguiente valor x_i' de una variable x_i se define mediante una función $f_i : S \to \mathcal{P}(\{0,1\}) - \{\varnothing\}$, que se puede escribir con una serie de abreviaturas, incluyendo un valor predeterminado. Además, Carrillo y Rosenblueth [5] proporcionan operaciones que modifican tales representaciones concisas de modelos de Kripke. Queda por explorar la posibilidad de extender $\textsc{Upd}_{prot}$ con estas representaciones concisas para lograr, posiblemente, un mejor escalamiento para $\textsc{Upd}_{prot}$.

2.5. Ding y Hemer, 2010

Ding y Hemer [15] realizan una mejora sobre [14, 26] en el tratamiento de fórmulas de la forma $\mathbf{AG}\,\varphi$, donde φ es una fórmula proposicional. Esta mejora produce un menor número de modelos candidatos que el método descrito en la tesis de Ding [14]. En esencia, el método original de [14] preserva trayectorias existentes entre un estado inicial y un estado arbitrario. El método mejorado [15], por el contrario, preserva trayectorias existentes entre cualquier par de estados. Ding y Hemer aplican su método de actualización mejorado a un protocolo para coherencia de caché del sistema de archivos Andrew (AFS-1) [25], y generan 125 modelos candidatos, en lugar de los 225 modelos generados por el método de Ding [14].

Como [15] es una mejora sobre [26], ahora debemos evaluar [15]. En el ejemplo sobre

AFS-1, Ding [14] da como entrada a su actualizador un modelo de Kripke y la fórmula:

$$\mathbf{AG}\left((Server.belief = valid) \rightarrow (Client.belief = valid)\right) \tag{2-1}$$

la cual es falsa en el modelo de Kripke de la entrada. Note que esta fórmula representa una propiedad que no es deseable, y se incluye en [25] sólo para ilustrar la forma en que SMV produce contraejemplos. Por lo tanto, los modelos que produce el actualizador de Ding tienen, en este caso, una propiedad que no es deseable. Ding [14, p. 100], sin embargo, observa que: "[...] after our model updating, we do not need to consider the logic outcome of the updated models under the false specification property."

Por otra parte, en el ejemplo sobre AFS-1, Ding ignora otra fórmula que figura en [25]:

$$\mathbf{AG}\left((Client.belief = valid) \rightarrow (Server.belief = valid)\right) \tag{2-2}$$

la cual es cierta en el modelo de entrada y representa una propiedad deseable. Como resultado, es posible que los modelos producidos por el actualizador de Ding, para satisfacer la fórmula (2-1), no satisfagan la fórmula (2-2). Si se hubiera considerado la fórmula (2-2) en la actualización, no sólo se hubieran producido menos modelos, sino que tales modelos satisfarían esta fórmula que representa una propiedad deseable. Mientras más propiedades se dan como entrada a un actualizador, son menos los modelos que éste genera. Por ejemplo, si damos a $\textsc{Upd}_{prot}$ (Sección 3.5.3) una especificación CTL completa del modelo de entrada, junto con la fórmula (2-1), el número de modelos producidos se reduce a cero (la razón es que la fórmula (2-1) no sólo es falsa en el modelo, también es inconsistente con la especificación del modelo).

Llegamos a la conclusión de que la proliferación de modelos producidos por el actualizador Ding no es un problema inherente de la actualización de modelos, sino más bien una consecuencia de ignorar propiedades deseables. Por lo tanto, en lugar de preferir modelos que preservan trayectorias existentes (condición que, por cierto, no se justifica en [14, 26]), se deberían considerar fórmulas que representen propiedades deseables.

2.6. Kelly y Zhang, 2010

Kelly *et al.* [24] y Kelly y Zhang [23] presentan un método basado en contraejemplos para ACTL. Estos autores reportan experimentos sobre el problema de exclusión mutua de Buccafurri *et al.* [2] y sobre el protocolo *sliding-window* de Forouzan [19], respectivamente.

Aunque los autores no proporcionan su algoritmo, sabemos que: "[it] was designed with a top down recursive approach with respect to the given model and properties." [23, p. 16]. Además, Kelly *et al.* [24] observan que: "when we perform a model update, we may require this update not violate other specified functions (e.g. breaking a deadlock should *not* violate a liveness in a concurrent program)." [24, p. 138]. Para este propósito, los modelos de Kripke se extienden con "acciones" que etiquetan transiciones, y con un conjunto de dos tipos de autómatas deterministas finitos, destinados a codificar restricciones sobre los valores de las variables, y restricciones sobre la precedencia de las acciones dentro de una trayectoria, respectivamente. Cada uno de estos autómatas tiene un estado distinguido de "violación". Aunque Kelly *et al.* [24] no describen la construcción de los autómatas, sabemos que tales autómatas codifican "complex constraints that are usually not expressible [...] in the form of ACTL (or CTL) formulas." [24, p. 139]. En los experimentos de ambos casos [24, 23], las fórmulas son de la forma $\mathbf{AG}\,\varphi$, donde φ es una fórmula proposicional. Kelly y Zhang [23] reportan actualizaciones de modelos con hasta 512 estados (aunque aparentemente los autómatas no se utilizaron para estos experimentos).

2.7. Guerra y Wassermann, 2010

En el siguiente trabajo que resumimos, Guerra y Wassermann [20] proporcionan un algoritmo para realizar revisión CTL de modelos de Kripke utilizando el algoritmo CTLUpdate de Zhang y Ding [26].

La actualización de creencias supone un mundo dinámico, y la información nueva representa cambios en tal mundo. La revisión de creencias, por el contrario, asume un

mundo estático y el objetivo es restaurar la consistencia cuando se añade información nueva.

Guerra y Wassermann proporcionan el siguiente algoritmo para revisión de modelos: Sean ψ una base de creencias y ϕ una nueva creencia. Sea S un conjunto de modelos inicializado a $\varnothing$. Se enumeran los modelos que satisfacen ψ. Los modelos que satisfacen ψ y que también satisfacen ϕ se añaden a S. Los modelos que satisfacen ψ pero no satisfacen ϕ se actualizan con CTLUpdate y los resultados se agregan a S. Por último, Guerra y Wassermann eliminan de S los modelos que no son minimales respecto a los modelos que satisfacen ψ.

Estos autores afirman que su algoritmo para revisión de modelos puede ser más adecuado que el método para actualización CTL de modelos propuesto por Zhang y Ding, cuando éste se aplica a modificaciones de un sistema en un contexto estático. Dado que las ventajas que estos autores encuentran sobre el método de actualización de modelos de Zhang y Ding no dependen de los mecanismos internos de tal método, parece justo concluir que, para revisión de modelos, el algoritmo de Guerra y Wassermann también sería preferible si nuestro algoritmo $\textsc{Upd}_{prot}$ reemplazara al algoritmo CTLUpdate de Zhang y Ding.

2.8. Chatzieleftheriou et al., 2012

Por último, Chatzieleftheriou *et al.* [10] desarrollan un método de actualización CTL empleando abstracción de modelos. La abstracción de un modelo de Kripke está determinada por una función que mapea un conjunto de estados concretos a un estado abstracto. En lugar de una relación de accesibilidad, la abstracción tiene dos relaciones: R_{must} y R_{may}. Hay una transición en R_{must} de $\hat{s}_1$ a $\hat{s}_2$ si hay transiciones de todos los estados concretos de $\hat{s}_1$ a algún estado concreto de $\hat{s}_2$. Por el contrario, hay una transición en R_{may} de $\hat{s}_1$ a $\hat{s}_2$ si hay una transición de un estado concreto de $\hat{s}_1$ a algún estado concreto de $\hat{s}_2$. Un literal etiqueta a un estado abstracto *sólo si* tal literal etiqueta a

todos los estados concretos de tal estado abstracto. Por lo tanto, la función de etiquetado abstracto es parcial. Consecuentemente, la abstracción proporciona una semántica trivaluada para literales y fórmulas. Si en un estado abstracto el valor de verdad de una fórmula no está definido, se realiza un paso de refinamiento con el fin de obtener una abstracción más concreta que proporcione un valor verdadero o falso a dicha fórmula.

Al igual que nuestro método y el método de Zhang y Ding [26], el método de Chatzieleftheriou *et al.* [10] es recursivo y emplea información auxiliar adicional generada por otras subfórmulas. Dicha información, en este caso, es un conjunto de pares (estado,fórmula), también llamados restricciones, y se utiliza de la siguiente manera. El algoritmo de Chatzieleftheriou *et al.* [10] produce como salida modelos que satisfacen tanto a la fórmula de entrada como las restricciones. Al mismo tiempo, la actualización en un estado s respecto a una fórmula $\alpha \wedge \beta$ procede tratando primero α con (s, β) agregada al conjunto de pares, tratando luego β con (s, α) agregada al conjunto de pares, y finalmente "combining both results appropriately" [10, p. 349].

Ahora evaluemos el método de Chatzieleftheriou et al. En primer lugar, aunque Chatzieleftheriou et al. afirman que su método es correcto, estos autores no afirman que el mismo es completo.

Segundo, Chatzieleftheriou *et al.* [10] proporcionan un algoritmo determinista, a diferencia de nuestro método, que es no determinista. Creemos que un enfoque no determinista tiene la ventaja de permitir una separación entre la esencia y la estrategia de búsqueda del algoritmo.

Finalmente, Chatzieleftheriou *et al.* [10] utilizan abstracciones, introduciendo así un elemento de mejora relevante en términos de eficiencia. Esto sugiere que sería prometedor explorar la posibilidad de desarrollar un algoritmo que combine las abstracciones de Chatzieleftheriou et al. con nuestras protecciones para obtener un algoritmo con lo mejor de ambos métodos.

Capítulo 3

Actualización mediante protecciones

En este capítulo presentamos nuestro método para actualización de modelos respecto a fórmulas de una variante de la CTL (computation-tree logic). Este método utiliza un nuevo mecanismo, al que denominamos "protecciones", para superar las dificultades que implica la actualización respecto a fórmulas que tienen tanto cuantificadores universales como existenciales.

Después de una breve explicación de la notación (Sec. 3.1), en la Sección 3.2 describimos la sintaxis y la semántica de la Σ-CTL, una variante de la CTL. En la Sección 3.3 damos una explicación de las operaciones del pseudocódigo no determinista que utilizamos para describir nuestro algoritmo. En la Sección 3.4 definimos $\textsc{Upd}_1$, un algoritmo para actualización directa de modelos, mientras que la Sección 3.5 definimos $\textsc{Upd}_{prot}$, un algoritmo de actualización mediante protecciones para fórmulas de la Σ-CTL. En la Sección 3.6, para mejorar la eficiencia de $\textsc{Upd}_{prot}$, proponemos algunas heurísticas basadas en un orden sobre la generación de elecciones no deterministas entre los elementos: de conjuntos de estados, conjuntos de conjuntos de estados, y conjuntos de dos fórmulas. Finalmente, la Sección 3.7 incluye una demostración de la corrección y la completitud de XUpd_{prot} (núcleo de $\textsc{Upd}_{prot}$), mientras que la Sección 3.8 proporciona un análisis de la complejidad de $\textsc{Upd}_{prot}$.

3.1. Notación

Además del lenguaje matemático básico (e.g. notación para funciones y conjuntos, y los cuantificadores $\forall$ y $\exists$), usamos la siguiente notación.

Si $A \cap B = \varnothing$, $f\colon A \to C$, y $g\colon B \to D$, entonces la función $(f \cup g)\colon A \cup B \to C \cup D$ se define mediante

$$(f \cup g)(t) = \begin{cases} f(t) & \text{si } t \in A \\ g(t) & \text{si } t \in B. \end{cases}$$

Usamos I_D para denotar la identidad sobre D, mientras que C_D denota una función constante tal que, $\forall t \in D$, $C_D(t) = C$.

Para denotar la diferencia simétrica de C y D, usamos $\Delta(C, D)$.

Si $\mathcal{E} = \langle c_1, \ldots, c_n \rangle$, denotamos el i-esimo componente de $\mathcal{E}$ con $c_i^{\mathcal{E}}$. Por ejemplo, si $\mathcal{M} = \langle S, R, L \rangle$ y $\Sigma = \langle S, V \rangle$, entonces $S^{\mathcal{M}}$ denota al primer componente de $\mathcal{M}$, y S^{Σ} denota al primer componente de Σ.

3.2. Σ-CTL, una variante de la CTL

En esta sección definimos la Σ-CTL, una variante de la CTL. La Σ-CTL utiliza una base de operadores *cerrada bajo la dualidad* con los siguientes pares de *operadores duales*: (F, T), $(\vee, \wedge)$, $(\mathbf{EX}, \mathbf{AX})$, $(\mathbf{EU}, \mathbf{AR})$, y $(\mathbf{AU}, \mathbf{ER})$. Por construcción, las fórmulas de la Σ-CTL están en *forma normal de negación* (NNF, por sus siglas en inglés), la cual limita la aplicación del operador de negación a variables. Sin embargo, ya que utilizamos una base de operadores cerrada bajo la dualidad, otras instancias del operador de negación se pueden considerar como una abreviatura. Además, la Σ-CTL tiene fórmulas de la forma $\mathbf{OD}_{\leq} n$, y sus duales $\mathbf{OD}_{>} n$, con una semántica que permite limitar el número de transiciones que *salen* de un estado (usamos "**OD**" haciendo referencia al término "outdegree" de teoría de grafos).

3.2.1. Sintaxis de la Σ-CTL

Las fórmulas de la Σ-CTL se construyen usando un *vocabulario* $\Sigma = \langle S, V\rangle$ que se compone de un par de conjuntos finitos no vacíos. A los elementos de S y V los llamamos *estados* y *variables (proposicionales)*, respectivamente. A menos que se indique lo contrario, asumimos que $\Sigma = \langle \mathbb{S}, \mathbb{V}\rangle$ es un vocabulario fijo arbitrario. El conjunto de *literales* sobre $\mathbb{V}$ es $\mathrm{Lit}(\mathbb{V}) = \mathbb{V} \cup \{\neg p \mid p \in \mathbb{V}\}$. El *complemento* de un literal se define mediante $\overline{p} = \neg p$ y $\overline{\neg p} = p$, $\forall p \in \mathbb{V}$. Si $X \subseteq \mathrm{Lit}(\mathbb{V})$ es un conjunto de literales, el complemento de X se define como $\overline{X} = \{\overline{\ell} \mid \ell \in X\}$.

La sintaxis de las fórmulas de la Σ-CTL es como sigue.

Definición 3.2.1 (Σ-CTL y Σ-XCTL)**.** Las fórmulas de la *lógica de árbol de cómputo con vocabulario* Σ, Σ-CTL (abreviado Φ), tienen la siguiente sintaxis:

$$\Phi ::= \mathsf{F} \mid \mathsf{T} \mid \ell \mid (\Phi \vee \Phi) \mid (\Phi \wedge \Phi) \mid (\mathbf{EX}\,\Phi) \mid (\mathbf{AX}\,\Phi) \mid \mathbf{OD}_{\leq} n \mid \mathbf{OD}_{>} n$$
$$\mid \mathbf{E}[\Phi\,\mathbf{U}\,\Phi] \mid \mathbf{A}[\Phi\,\mathbf{U}\,\Phi] \mid \mathbf{E}[\Phi\,\mathbf{R}\,\Phi] \mid \mathbf{A}[\Phi\,\mathbf{R}\,\Phi]$$

donde ℓ representa cualquier literal en $Lit(\mathbb{V})$ y $0 < n \leq |\mathbb{S}|$. Observe que, al igual que F, T y ℓ, las fórmulas $\mathbf{OD}_{\leq} n$ y $\mathbf{OD}_{>} n$ son *atómicas*.

Distinguimos el *fragmento modal* de la Σ-CTL. Las fórmulas modales de la Σ-CTL, Σ-*XCTL* (abreviado Ψ), tienen la siguiente sintaxis:

$$\Psi ::= \mathsf{F} \mid \mathsf{T} \mid \ell \mid (\Psi \vee \Psi) \mid (\Psi \wedge \Psi) \mid (\mathbf{EX}\,\Psi) \mid (\mathbf{AX}\,\Psi) \mid \mathbf{OD}_{\leq} n \mid \mathbf{OD}_{>} n$$

Usamos $\varphi \in$ Σ-CTL para indicar que φ es una fórmula de la Σ-CTL.

Si φ se construye a partir de las fórmulas atómicas F, T, y ℓ, mediante los operadores $\vee$ y $\wedge$, decimos que φ es *proposicional*; en caso contrario decimos que φ es una *fórmula no-proposicional*. Consideramos a otros operadores proposicionales ($\rightarrow$, $\leftrightarrow$, disyunción

exclusiva $\veebar$) y a otros operadores temporales (**EF** , **AF** , **EG** , **AG**) como abreviaturas:

$$\begin{aligned} \mathbf{EF}\,\alpha &= \mathbf{E}[\mathsf{T}\ \mathbf{U}\ \alpha] \\ \mathbf{AF}\,\alpha &= \mathbf{A}[\mathsf{T}\ \mathbf{U}\ \alpha] \\ \mathbf{EG}\,\alpha &= \mathbf{E}[\mathsf{F}\ \mathbf{R}\ \alpha] \\ \mathbf{AG}\,\alpha &= \mathbf{A}[\mathsf{F}\ \mathbf{R}\ \alpha]. \end{aligned} \tag{3-1}$$

Nuestra sintaxis para la Σ-CTL, que usa una base de operadores cerrada bajo la dualidad y que sólo permite fórmulas en NNF, es un punto clave en la simplicidad de nuestro algoritmo para actualización de modelos. Otros tipos de sintaxis, como las sintaxis que utilizan una base de operadores similar a las que se usan en verificación CTL de modelos [12], dificultan la definición de un algoritmo para actualización de modelos en los casos de algunas fórmulas. Por ejemplo, si la sintaxis (y la actualización de modelos) se define utilizando la base $\{\neg, \wedge, \mathbf{EX}, \mathbf{AF}, \mathbf{EU}\}$, entonces no resulta sencilla la definición de la actualización de modelos respecto a fórmulas de la forma $\neg\alpha$, $\alpha \vee \beta$, $\mathbf{AX}\,\alpha$, $\mathbf{EG}\,\alpha$, y $\mathbf{A}[\alpha\ \mathbf{R}\ \beta]$ (i.e. fórmulas en las que ocurre un operador que es dual de un operador de la base).

Otro punto clave es que nuestro algoritmo se enfoca en fórmulas de la Σ-XCTL, y trata a los operadores de trayectoria (**EU**, **AU**, **ER**, y **AR**) a través de sus caracterizaciones de punto fijo (ecuaciones 3-3).

3.2.2. Semántica de la Σ-CTL mediante modelos protegidos

En esta sección definimos la semántica de la Σ-CTL utilizando modelos protegidos. La semántica estándar de la CTL resulta un caso particular de la semántica con modelos protegidos.

Si $X \subseteq \mathrm{Lit}(\mathbb{V})$, decimos que X es *consistente* si $\forall \ell \in X$, $\overline{\ell} \notin X$, y decimos que X es $\mathbb{V}$-*maximal* si $\forall p \in \mathbb{V}$, $p \in X$ o $\neg p \in X$. Si $R \subseteq \mathbb{S} \times \mathbb{S}$ entonces R es *total*[1] si $\forall s \in \mathbb{S}$, $\exists t \in \mathbb{S}$

[1] Algunos autores usan "serial". Preferimos usar "total" porque así se usa en verificación de modelos [12]

tal que $(s,t) \in R$. El conjunto de *sucesores* de s bajo R es $R[s] = \{t \in \mathbb{S} \mid (s,t) \in R\}$, y la identidad sobre $\mathbb{S}$ es $I_{\mathbb{S}} = \{(t,t) \in \mathbb{S} \times \mathbb{S} \mid t \in \mathbb{S}\}$.

Definición 3.2.2 (Σ-modelos de Kripke)**.** Decimos que $\mathcal{M} = \langle S, R, L\rangle$ es un Σ-*modelo de Kripke*, brevemente Σ-modelo, si $S = \mathbb{S}$, $R \subseteq S^2$ es total, y $L\colon S \to \mathcal{P}(\mathrm{Lit}(\mathbb{V}))$ es tal que $\forall t \in S$, $L(t)$ es consistente y $\mathbb{V}$-maximal.

Usamos $\mathbf{K}_\Sigma$ para denotar al *conjunto de* Σ-*modelos* de Kripke. Si $\mathcal{M} = \langle S, R, L\rangle \in \mathbf{K}_\Sigma$, decimos que $(s,t) \in R$ es una *transición de* s *a* t y abreviamos esto con sRt. Llamamos a L la *función de etiquetamiento* de $\mathcal{M}$. Si $\ell \in \mathrm{Lit}(\mathbb{V})$ y $s \in S$, entonces $L[s \oplus \ell]$ denota la función de etiquetamiento tal que $L[s \oplus \ell](s) = (L(s) \cup \{\ell\}) - \{\overline{\ell}\}$ y $L[s \oplus \ell](t) = L(t)$ para $t \neq s$.

Si $\mathcal{M}, \mathcal{M}' \in \mathbf{K}_\Sigma$, definimos la *distancia* entre $\mathcal{M}$ y $\mathcal{M}'$ como

$$d(\mathcal{M}, \mathcal{M}') = |\Delta(S^{\mathcal{M}}, S^{\mathcal{M}'})| + |\Delta(R^{\mathcal{M}}, R^{\mathcal{M}'})| + \sum_{t \in S^{\mathcal{M}}} (|\Delta(L^{\mathcal{M}}(t), L^{\mathcal{M}'}(t))|/2).$$

Se puede demostrar que d es una métrica sobre $\mathbf{K}_\Sigma$.

Los modelos frecuentemente se representan gráficamente como en la Sección 4, escribiendo sólo los literales positivos como etiquetas de los estados.

A continuación, tenemos definiciones básicas para el mecanismo de protecciones.

Definición 3.2.3 (Σ-Protecciones)**.** $P = \langle E, A, L\rangle$ es una Σ-*protección* si:

1. $E \subseteq A \subseteq \mathbb{S} \times \mathbb{S}$, y
2. $L\colon \mathbb{S} \to \mathcal{P}(\mathrm{Lit}(\mathbb{V}))$ es tal que $\forall t \in \mathbb{S}$, $L(t)$ es consistente.

Las protecciones son el núcleo de nuestro algoritmo de actualización. Intuitivamente, una protección $P = \langle E, A, L\rangle$ registra información acerca de la satisfacción de las subfórmulas tratadas previamente por nuestro algoritmo. En P, E es el conjunto de transiciones (s,t) tales que (s,t) *no se puede remover* sin comprometer la satisfacción de las subfórmulas previamente tratadas. Análogamente, A es el conjunto de transiciones

(s,t) tales que (s,t) *se puede agregar*. Finalmente, L es el conjunto de literales ℓ tales que ℓ *no se puede cambiar a* $\bar{\ell}$ sin comprometer la satisfacción de las subfórmulas previamente tratadas. Las transiciones relacionadas con la satisfacción de las subfórmulas universales (existenciales) se registran en A (E). Así, para preservar la satisfacción de las subfórmulas previamente tratadas, procedemos sin disminuir E o aumentar A, y sin modificar literales que pertenecen a $L(t), \forall t \in \mathbb{S}$.

Nos referimos a los estados en $E[s]$ ($A[s]$) como los sucesores de s *protegidos existencialmente (universalmente)*. Usaremos $\mathbf{P}_\Sigma$ para denotar al conjunto de Σ-protecciones.

Definición 3.2.4 (Modelos protegidos, $P_\mathcal{M}$, y $P_\perp$)**.** Sean $\mathcal{M} = \langle S, R, L\rangle \in \mathbf{K}_\Sigma$ y $P = \langle E, A, L\rangle \in \mathbf{P}_\Sigma$. Decimos que $\mathcal{M}$ *está protegido por* P, y escribimos $\mathcal{M} \rhd P$ si:

1. $E \subseteq R \subseteq A$, y
2. $\forall t \in S : L^\mathcal{M}(t) \supseteq L^P(t)$.

Decimos que $(\mathcal{M}, P)$ *es un* Σ*-modelo protegido* si $\mathcal{M} \rhd P$, y usamos $\mathbf{KP}_\Sigma$ para denotar al conjunto de Σ*-modelos protegidos*.

La Σ*-protección completa* de un modelo $\mathcal{M}$ es $P_\mathcal{M} = \langle R^\mathcal{M}, R^\mathcal{M}, L^\mathcal{M}\rangle$.

La Σ*-protección vacía* es $P_\perp = \langle \varnothing, \mathbb{S}\times\mathbb{S}, L_\perp\rangle$, donde $L_\perp(t) = \varnothing$ para todo $t \in \mathbb{S}$.

Si $R \subseteq S^2$ y $s \in S$, una *trayectoria* en R *que comienza en* s, es a sucesión $\pi\colon \mathbb{N} \to S$, tal que $\pi(0) = s$ y $\forall n \in \mathbb{N}$, $\pi(n)R\pi(n+1)$. Escribimos π_n en lugar de $\pi(n)$ y usamos $\Pi_{R,s}$ para denotar al *conjunto de trayectorias* en R que comienzan en s.

Definición 3.2.5 (Semántica protegida de la Σ-CTL)**.** Si $(\mathcal{M}, P) \in \mathbf{KP}_\Sigma$ es un Σ-modelo protegido, $s \in S^\mathcal{M}$ y $\varphi \in \Sigma$-CTL, entonces definimos la relación $(\mathcal{M}, P)$ *satisface* φ en s, $(\mathcal{M}, P), s \models \varphi$, recursivamente sobre φ:

1. $(\mathcal{M}, P), s \not\models \mathsf{F}$. $\quad (\mathcal{M}, P), s \models \mathsf{T}$.
2. $(\mathcal{M}, P), s \models \ell$ si $\ell \in L^P(s)$.
3. $(\mathcal{M}, P), s \models \alpha \vee \beta$ si $(\mathcal{M}, P), s \models \alpha$ o $(\mathcal{M}, P), s \models \beta$.

4. $(\mathcal{M}, P), s \models \alpha \wedge \beta$ si $(\mathcal{M}, P), s \models \alpha$ y $(\mathcal{M}, P), s \models \beta$.

5. $(\mathcal{M}, P), s \models \mathbf{EX}\,\alpha$ si $\exists t \in E^P[s]$ tal que $(\mathcal{M}, P), t \models \alpha$.

6. $(\mathcal{M}, P), s \models \mathbf{AX}\,\alpha$ si $\forall t \in A^P[s]$, $(\mathcal{M}, P), t \models \alpha$.

7. $(\mathcal{M}, P), s \models \mathbf{E}[\alpha\,\mathbf{U}\,\beta]$ si $\exists \pi \in \Pi_{E^P,s}$, y $\exists j \in \mathbb{N}$ tales que

   $(\mathcal{M}, P), \pi_j \models \beta$ y $\forall i \in \mathbb{N}$, $i < j \rightarrow (\mathcal{M}, P), \pi_i \models \alpha$.

8. $(\mathcal{M}, P), s \models \mathbf{A}[\alpha\,\mathbf{U}\,\beta]$ si $\forall \pi \in \Pi_{A^P,s}$, $\exists j \in \mathbb{N}$ tal que

   $(\mathcal{M}, P), \pi_j \models \beta$ y $\forall i \in \mathbb{N}$, $i < j \rightarrow (\mathcal{M}, P), \pi_i \models \alpha$.

9. $(\mathcal{M}, P), s \models \mathbf{E}[\alpha\,\mathbf{R}\,\beta]$ si $\exists \pi \in \Pi_{E^P,s}$ tal que

   a) $\forall k \in \mathbb{N}$, $(\mathcal{M}, P), \pi_k \models \beta$, o

   b) $\exists j \in \mathbb{N}$ tal que $(\mathcal{M}, P), \pi_j \models \alpha$ y $\forall i \in \mathbb{N}$, $i \leq j \rightarrow (\mathcal{M}, P), \pi_i \models \beta$.

10. $(\mathcal{M}, P), s \models \mathbf{A}[\alpha\,\mathbf{R}\,\beta]$ si $\forall \pi \in \Pi_{A^P,s}$,

    a) $\forall k \in \mathbb{N}$, $(\mathcal{M}, P), \pi_k \models \beta$, o

    b) $\exists j \in \mathbb{N}$ tal que $(\mathcal{M}, P), \pi_j \models \alpha$ y $\forall i \in \mathbb{N}$, $i \leq j \rightarrow (\mathcal{M}, P), \pi_i \models \beta$.

11. $(\mathcal{M}, P), s \models \mathbf{OD}_{\leq}\, n$ si $|A^P[s]| \leq n$.

12. $(\mathcal{M}, P), s \models \mathbf{OD}_{>}\, n$ si $|E^P[s]| > n$.

La semántica estándar (no protegida) de la Σ-CTL resulta un caso particular de la semántica protegida de la Σ-CTL. La semántica estándar de φ en $\mathcal{M}$ se obtiene de la semántica protegida usando la protección completa de $\mathcal{M}$, $P_{\mathcal{M}}$.

Definición 3.2.6 (Semántica estándar de la Σ-CTL)**.** Si $\mathcal{M}$ es un Σ-modelo, $s \in S^{\mathcal{M}}$ y $\varphi \in$ Σ-CTL, decimos que $\mathcal{M}$ *satisface* φ en s, $\mathcal{M}, s \models \varphi$, si $(\mathcal{M}, P_{\mathcal{M}}), s \models \varphi$.

Extendemos la semántica protegida de la Σ-CTL a conjuntos de estados y conjuntos de fórmulas. Si $S \subseteq S^{\mathcal{M}}$ entonces $(\mathcal{M}, P), S \models \varphi$ si $\forall s \in S$, $(\mathcal{M}, P), s \models \varphi$. Si $\Gamma \subseteq \Sigma$-CTL entonces $(\mathcal{M}, P), s \models \Gamma$ si $\forall \varphi \in \Gamma$, $(\mathcal{M}, P), s \models \varphi$. Si φ_1, φ_2 son dos fórmulas de la Σ-CTL, decimos que φ_1 y φ_2 son *lógicamente equivalentes*, y escribimos $\varphi_1 \equiv \varphi_2$, si $\forall \mathcal{M} \in \mathbf{K}_\Sigma$ y $\forall s \in S^{\mathcal{M}}$: $\mathcal{M}, s \models \varphi_1$ sii $\mathcal{M}, s \models \varphi_2$.

En la Sección 4, usamos la equivalencia lógica

$$\mathbf{AG}\,\alpha \wedge \mathbf{AG}\,\beta \equiv \mathbf{AG}\,(\alpha \wedge \beta) \tag{3-2}$$

para factorizar conjunciones de fórmulas **AG** . Además, nuestro algoritmo usa las siguientes equivalencias lógicas, conocidas como caracterizaciones de punto fijo, para calcular recursivamente la actualización de un modelo en los casos respectivos.

$$\begin{aligned}
\mathbf{E}[\alpha\,\mathbf{U}\,\beta] &\equiv \beta \vee (\alpha \wedge \mathbf{EX}\,\mathbf{E}[\alpha\,\mathbf{U}\,\beta]) \\
\mathbf{A}[\alpha\,\mathbf{U}\,\beta] &\equiv \beta \vee (\alpha \wedge \mathbf{AX}\,\mathbf{A}[\alpha\,\mathbf{U}\,\beta]) \\
\mathbf{E}[\alpha\,\mathbf{R}\,\beta] &\equiv \beta \wedge (\alpha \vee \mathbf{EX}\,\mathbf{E}[\alpha\,\mathbf{R}\,\beta]) \\
\mathbf{A}[\alpha\,\mathbf{R}\,\beta] &\equiv \beta \wedge (\alpha \vee \mathbf{AX}\,\mathbf{A}[\alpha\,\mathbf{R}\,\beta])
\end{aligned} \tag{3-3}$$

La actualización de un modelo respecto a fórmulas del tipo **EU** o **AU** se puede calcular mediante un operador de punto fijo mínimo (lfp), mientras que la actualización de un modelo respecto a fórmulas del tipo **ER** o **AR** se puede calcular mediante un operador de punto fijo máximo (gfp) [12, p. 63].

3.3. Pseudocódigo no determinista

Para la descripción del algoritmo de actualización con protecciones (Sección 3.5.2) y del algoritmo de actualización directa (Sección 3.4.2), utilizamos un *pseudocódigo no determinista* que cuenta, básicamente, con una instrucción para expresar elecciones no deterministas entre los elementos de un multiconjunto. Además de las instrucciones de

control habituales y la definición de procedimientos y funciones, nuestro pseudocódigo cuenta con las siguientes instrucciones.

Dado un multiconjunto finito $A \neq \varnothing$, el cómputo de la instrucción "**guess** $x \in A$" *elige en forma no determinista* un elemento $u \in A$, y asigna u a la variable x. Análogamente, el cómputo de la instrucción "**pick** $x \in A$" *elige en forma determinista* un elemento $u \in A$, y asigna u a la variable x. Si $A = \varnothing$, el cómputo de ambas instrucciones, "**guess** $x \in A$" y "**pick** $x \in A$", *falla*.

Intuitivamente, cada ocurrencia de una instrucción "**guess** $x \in A$" causa que el cómputo continúe simultáneamente en diferentes *trayectorias de ejecución*, una trayectoria por cada elemento de A. La *falla* de una instrucción durante el cómputo de las instrucciones de una trayectoria de ejecución significa que la ejecución de dicha trayectoria se detiene sin devolver ningún resultado.

Si P es un procedimiento o una función, decimos que P es *no determinista* si hay alguna ocurrencia de **guess** en las instrucciones que definen a P.

Nuestro pseudocódigo cuenta con dos instrucciones para indicar el *final de una trayectoria de ejecución* en el cómputo de un procedimiento, o función, no determinista P. La instrucción "**return** r" detiene la ejecución de la trayectoria y, como uno de los resultados calculados por P, devuelve r. La instrucción "**fail**" detiene la ejecución de la trayectoria sin devolver ningún resultado.

Las trayectorias de ejecución que terminan con **return** son *trayectorias exitosas*, y las que terminan en **fail** son *trayectorias fallidas*. El *multiconjunto de resultados* computado por $P(a_1, \ldots, a_n)$, se denota con $P[a_1, \ldots, a_n]$. De este modo, $P[a_1, \ldots, a_n] = \varnothing$ significa que, para los argumentos dados, todas las trayectorias de ejecución de P son fallidas; en este caso decimos que P *falla*.

La instrucción "$x \leftarrow e$" es una *asignación no determinista* equivalente a "**guess** $x \in A_e$", donde A_e es el multiconjunto de los valores producidos por el cómputo no determinista de e. Si $A_e \neq \varnothing$, y existe c tal que $A_e \subseteq \{c\}$, es decir, si el cómputo de e produce solamente un valor, escribimos "$x := e$" en lugar de "$x \leftarrow e$".

3.4. Actualización directa de modelos

En esta sección definimos un algoritmo para actualización directa de modelos, XUPD_1, que utilizamos como referencia para comparar nuestro algoritmo de actualización con protecciones. Por simplicidad, ignoramos temporalmente algunos detalles que se tendrán en cuenta en las siguientes secciones. Ahora nos enfocamos en Σ-XCTL, no incluimos una operación para añadir estados, y no nos preocupamos por la eficiencia.

3.4.1. Modificación de modelos

Construimos nuestro algoritmo para actualización de modelos mediante el uso de unas cuantas operaciones básicas que permiten cambiar gradualmente el modelo de entrada.

Definición 3.4.1 (Operaciones de actualización)**.** Sean $\mathcal{M} \in \mathbf{K}_\Sigma$, $s, s' \in S^\mathcal{M}$, $\ell \in \text{Lit}(\mathbb{V})$, $S' \subseteq S^\mathcal{M}$, y $S'' \neq \varnothing$ un conjunto finito, tales que $S' \neq \varnothing$ y $S'' \cap S^\mathcal{M} = \varnothing$. Las *operaciones de actualización sobre modelos de Kripke* son las siguientes:

1. $\mathbf{L}^u(\mathcal{M}, s, \ell) = \langle S^\mathcal{M}, R^\mathcal{M}, L^\mathcal{M}[s \oplus \ell]\rangle$. Agregar ℓ a $L^\mathcal{M}(s)$, y remover $\overline{\ell}$.

2. $\mathbf{T}^+(\mathcal{M}, s, s') = \langle S^\mathcal{M}, R^\mathcal{M} \cup \{(s, s')\}, L^\mathcal{M}\rangle$. Agregar (s, s') a $R^\mathcal{M}$.

3. $\mathbf{T}^u(\mathcal{M}, s, S') = \langle S^\mathcal{M}, (R^\mathcal{M} - (\{s\} \times R^\mathcal{M}[s])) \cup (\{s\} \times S'), L^\mathcal{M}\rangle$.

   Reemplazar los sucesores de s, $R^\mathcal{M}[s]$, por S'.

4. $\mathbf{S}^+(\mathcal{M}, S'') = \langle S^\mathcal{M} \cup S'', R^\mathcal{M} \cup I_{S''}, L^\mathcal{M} \cup \overline{\mathbb{V}}_{S''}\rangle$.

   Agregar S'' a $S^\mathcal{M}$, agregar $I_{S''}$ a $R^\mathcal{M}$, y etiquetar todo $t \in S''$ con $\overline{\mathbb{V}}$ (recordar que $\overline{\mathbb{V}}_{S''}$ es la función constante tal que $\forall t \in S'', \overline{\mathbb{V}}_{S''}(t) = \overline{\mathbb{V}}$).

La aplicación sucesiva de las operaciones $\mathbf{L}^u$, $\mathbf{T}^+$, y $\mathbf{T}^u$ es suficiente para transformar un Σ-modelo dado en cualquier otro Σ-modelo. Además, estas operaciones son instrumentales para lograr que la modificación de un modelo satisfaga literales, fórmulas **EX** y fórmulas **AX**, respectivamente. Note que $\mathbf{S}^+(\mathcal{M}, S'')$ no es un modelo con vocabulario Σ, es decir, $\mathbf{S}^+(\mathcal{M}, S'') \notin \mathbf{K}_\Sigma$.

A continuación, damos una definición precisa de lo que consideramos una modificación aceptable de un modelo $\mathcal{M}$ respecto a una fórmula φ.

Definición 3.4.2 (Modificaciones de $\mathcal{M}$ respecto a φ)**.** Si $\mathcal{M}$ es un Σ-modelo, $s \in S^{\mathcal{M}}$ y $\varphi \in \Sigma$-XCTL, definimos, mediante recursión sobre φ, *el conjunto de modificaciones de* $\mathcal{M}$ *que satisfacen* φ *en* s, $\mathit{Modif}(\mathcal{M}, s, \varphi)$:

1. $\mathit{Modif}(\mathcal{M}, s, \mathsf{F}) = \varnothing \quad \text{y} \quad \mathit{Modif}(\mathcal{M}, s, \mathsf{T}) = \{\mathcal{M}\}$

2. $\mathit{Modif}(\mathcal{M}, s, \ell) = \{\mathbf{L}^u(\mathcal{M}, s, \ell)\}$

3. $\mathit{Modif}(\mathcal{M}, s, \alpha \vee \beta) = \mathit{Modif}(\mathcal{M}, s, \alpha) \cup \mathit{Modif}(\mathcal{M}, s, \beta)$

4. $$\begin{aligned}\mathit{Modif}(\mathcal{M}, s, \alpha \wedge \beta) = \; & \{\mathcal{M}' \in \mathbf{K}_\Sigma \mid \exists \mathcal{M}_\alpha \in \mathit{Modif}(\mathcal{M}, s, \alpha). \\ & \mathcal{M}' \in \mathit{Modif}(\mathcal{M}_\alpha, s, \beta) \\ & \;\&\; \mathcal{M}', s \models \alpha \wedge \beta\}\end{aligned}$$

5. $$\begin{aligned}\mathit{Modif}(\mathcal{M}, s, \mathbf{EX}\,\alpha) = \; & \{\mathcal{M}' \in \mathbf{K}_\Sigma \mid \exists s' \in S^{\mathcal{M}}. \\ & \mathcal{M}' \in \mathit{Modif}(\mathbf{T}^+(\mathcal{M}, s, s'), s', \alpha) \\ & \;\&\; \mathcal{M}', s \models \mathbf{EX}\,\alpha\}\end{aligned}$$

6. $$\begin{aligned}\mathit{Modif}(\mathcal{M}, s, \mathbf{AX}\,\alpha) = \; & \{\mathcal{M}' \in \mathbf{K}_\Sigma \mid \exists S' \in \mathcal{P}(S^{\mathcal{M}}) - \{\varnothing\}. \\ & \mathcal{M}' \in \mathit{Modif}^*(\mathbf{T}^u(\mathcal{M}, s, S'), S', \alpha) \\ & \;\&\; \mathcal{M}', s \models \mathbf{AX}\,\alpha\}\end{aligned}$$

7. $$\begin{aligned}\mathit{Modif}(\mathcal{M}, s, \mathbf{OD}_\odot\, n) = \; & \{\mathcal{M}' \in \mathbf{K}_\Sigma \mid \exists S' \in \mathcal{P}(S^{\mathcal{M}}) - \{\varnothing\}. \\ & \quad |S'| \odot n \;\&\; \mathcal{M}' = \mathbf{T}^u(\mathcal{M}, s, S')\}, \\ & \text{para } \odot \in \{\leq, >\}\end{aligned}$$

Donde $\mathit{Modif}^*(\mathcal{M}, S', \varphi)$ *extiende* $\mathit{Modif}(\mathcal{M}, s, \varphi)$ *a un conjunto de estados* $S' \subseteq S^{\mathcal{M}}$:

$$\mathit{Modif}^*(\mathcal{M}, S', \varphi) = \begin{cases} \{\mathcal{M}\} & \text{si } S' = \varnothing \\ \{\mathcal{M}' \in \mathbf{K}_\Sigma \mid \exists t \in S'.\ \exists \mathcal{M}_t \in \mathit{Modif}(\mathcal{M}, t, \varphi). & \\ \quad \mathcal{M}' \in \mathit{Modif}^*(\mathcal{M}_t, S' - \{t\}, \varphi)\} & \text{si } S' \neq \varnothing \end{cases}$$

Observe que, $\forall \mathcal{M}' \in \mathit{Modif}(\mathcal{M}, s, \varphi)$, $\mathcal{M}', s \models \varphi$.

3.4.2. XUPD$_1$, un algoritmo de actualización directa

A continuación definimos XUPD$_1$, un algoritmo para actualización de modelos similar a los métodos generar-y-probar que implementa $\mathit{Modif}(\mathcal{M}, s, \varphi)$, las modificaciones de $\mathcal{M}$ que satisfacen φ. Primero, usando operaciones de actualización básicas, XUPD$_1$ genera modelos para satisfacer las subfórmulas más simples de la fórmula dada. Luego, XUPD$_1$ modifica estos modelos para satisfacer subfórmulas más complejas. Finalmente, los modelos producidos se prueban para verificar si satisfacen o no toda la fórmula.

La Figura 3-1 muestra el pseudocódigo de XUPD$_1^*$, un procedimiento auxiliar en la definición posterior de XUPD$_1$.

```
XUPD*_1(M, S', φ) % Actualiza M en S' respecto a φ
Input: M ∈ K_Σ, S' ⊆ S^M, φ ∈ XCTL
Output: Modif*(M, S', φ)
1. if S' = ∅ then M' := M
2. else {pick t ∈ S' ;
3.       M_t ← XUPD_1(M, t, φ) ;
4.       M' ← XUPD*_1(M_t, S' − {t}, φ)}
5. return M'
```

Figura 3-1: pseudocódigo de XUPD$_1^*$.

El procedimiento XUPD$_1^*(\mathcal{M}, S', \varphi)$ es una implementación de $\mathit{Modif}^*(\mathcal{M}, S', \varphi)$ para fórmulas $\varphi \in \Sigma$-CTL (Figura 3-1). Intuitivamente, XUPD$_1^*(\mathcal{M}, S', \varphi)$ actualiza el modelo $\mathcal{M}$, respecto a una fórmula φ, en un conjunto de estados S'. Para actualizar en S', XUPD$_1^*$ llama a XUPD$_1$ con cada uno de los elementos de S'. Por lo tanto, XUPD$_1^*$ y XUPD$_1$ son procedimientos mutuamente recursivos.

El pseudocódigo de XUPD$_1$, una implementación de $\mathit{Modif}(\mathcal{M}, s, \varphi)$, se muestra en la Figura 3-2.

$\text{XUPD}_1(\mathcal{M}, s, \varphi)$ % Actualiza $\mathcal{M}$ en s respecto a φ
Input: $\mathcal{M} \in \mathbf{K}_\Sigma$, $s \in S^{\mathcal{M}}$, $\varphi \in \Sigma$-XCTL
Output: *Modif*$(\mathcal{M}, s, \varphi)$
1. **case** φ **of**
2. $\quad \mathsf{F}$: **fail**
3. $\quad \mathsf{T}$: $\mathcal{M}' \leftarrow \mathcal{M}$
4. $\quad \ell$: $\mathcal{M}' \leftarrow \mathbf{L}^u(\mathcal{M}, s, \ell)$
5. $\quad \alpha \vee \beta$: {**guess** $\delta \in \{\alpha, \beta\}$ **;**
6. $\qquad\qquad \mathcal{M}' \leftarrow \text{XUPD}_1(\mathcal{M}, s, \delta)\}$
7. $\quad \alpha \wedge \beta$: {$\mathcal{M}_\alpha \leftarrow \text{XUPD}_1(\mathcal{M}, s, \alpha)$ **;**
8. $\qquad\qquad \mathcal{M}' \leftarrow \text{XUPD}_1(\mathcal{M}_\alpha, s, \beta)\}$
9. $\quad \mathbf{EX}\,\alpha$: {**guess** $s' \in S^{\mathcal{M}}$ **;**
10. $\qquad\qquad \mathcal{M}' \leftarrow \text{XUPD}_1(\mathbf{T}^+(\mathcal{M}, s, s'), s', \alpha)\}$
11. $\quad \mathbf{AX}\,\alpha$: {**guess** $S' \in \mathcal{P}(S^{\mathcal{M}}) - \{\varnothing\}$ **;**
12. $\qquad\qquad \mathcal{M}' \leftarrow \text{XUPD}_1^*(\mathbf{T}^u(\mathcal{M}, s, S'), S', \alpha)\}$
13. $\quad \mathbf{OD}_\odot\, n$: {**guess** $S' \in \{X \in \mathcal{P}(S^{\mathcal{M}}) - \{\varnothing\} \mid |X| \odot n\}$ **;**
14. $\qquad\qquad \mathcal{M}' \leftarrow \mathbf{T}^u(\mathcal{M}, s, S')\}$ % para $\odot \in \{\leq, >\}$
15. **if** $\mathcal{M}', s \models \varphi$ **then return** $\mathcal{M}'$ **else fail**

Figura 3-2: pseudocódigo de XUPD_1.

Al final del pseudocódigo, en la línea (15), XUPD_1 hace una prueba para verificar que el modelo generado, $\mathcal{M}'$, satisface la fórmula dada, φ. Si $\mathcal{M}'$ no satisface a φ, XUPD_1 falla ignorando $\mathcal{M}'$. Intuitivamente, el número posible de fallas en la línea (15) de XUPD_1 se traduce en tiempo de ejecución y aumenta conforme al número de posibles modelos $|\mathbf{K}_\Sigma|$, un número exponencial respecto a $|\mathbb{S}|$. Observe, sin embargo, que la línea (15) de XUPD_1 es necesaria para garantizar que el modelo devuelto, $\mathcal{M}'$, cumpla el requisito $\mathcal{M}', s \models \varphi$ en tres casos específicos de φ [6].

Usando caracterizaciones de punto fijo (ecuaciones 3-3, Sec. 3.2.2), y un mecanismo para la detección de ciclos (ver la Sección 3.5.3), extendemos $\text{XUPD}_1(\mathcal{M}, s, \varphi)$ a un algoritmo, UPD_1, que implementa *Modif*$(\mathcal{M}, s, \varphi)$ para $\varphi \in \Sigma$-CTL. El procedimiento UPD_1 utiliza el procedimiento auxiliar UPD_1^* (Figura 3-3).

Intuitivamente, UPD_1^* permite actualizar un modelo $\mathcal{M}$, respecto a una fórmula $\varphi \in \Sigma$-CTL, en un conjunto de estados S'. Para actualizar en S', UPD_1^* llama a UPD_1

$\textsc{Upd}_1^*((\mathcal{M}, Q), S', \varphi)$ % Actualiza $\mathcal{M}$ en S' respecto a φ
Input: $\mathcal{M} \in \mathbf{K}_\Sigma$, $Q \subseteq S^{\mathcal{M}} \times \Sigma$-CTL, $S' \subseteq S^{\mathcal{M}}$, y $\varphi \in \Sigma$-CTL
Output: $\mathit{Modif}^*(\mathcal{M}, S', \varphi)$ % ignorando las Q'
1. **if** $S' = \varnothing$ **then** $(\mathcal{M}', Q') := (\mathcal{M}, Q)$
2. **else** {**pick** $t \in S'$;
3. $(\mathcal{M}_t, Q_t) \leftarrow \textsc{Upd}_1\big((\mathcal{M}, Q), t, \varphi\big)$;
4. $(\mathcal{M}', Q') \leftarrow \textsc{Upd}_1^*\big((\mathcal{M}_t, Q_t), S' - \{t\}, \varphi\big)\}$
5. **return** $(\mathcal{M}', Q')$

Figura 3-3: pseudocódigo de $\textsc{Upd}_1^*$.

con cada uno de los elementos de S'. Por lo tanto, $\text{XU}\textsc{pd}_1^*$ y $\text{XU}\textsc{pd}_1$ son procedimientos mutuamente recursivos. El pseudocódigo de $\textsc{Upd}_1$ se muestra en la Figura 3-4.

3.5. Actualización mediante protecciones

En contraste con el algoritmo directo de la Sección 3.4, $\text{XU}\textsc{pd}_1$, en esta sección definimos $\text{XU}\textsc{pd}_{prot}$, un algoritmo de actualización, para fórmulas Σ-XCTL, que utiliza *protecciones*. El procedimiento $\text{XU}\textsc{pd}_{prot}$, núcleo de nuestro método de actualización, se puede aplicar a fórmulas del fragmento modal Σ-XCTL para reparar modelos de $\mathbf{KP}_\Sigma$ sin modificar el vocabulario Σ, en particular, sin agregar estados. Posteriormente, mostramos cómo extender $\text{XU}\textsc{pd}_{prot}$ a fórmulas de la Σ-CTL, y mostramos cómo agregar estados en el proceso de actualización.

Intuitivamente, una llamada a $\text{XU}\textsc{pd}_{prot}((\mathcal{M}, P), s, \varphi)$ transforma gradualmente $\mathcal{M}$ intentando satisfacer las subfórmulas de φ. Los modelos $\mathcal{M}'$, producidos por $\text{XU}\textsc{pd}_{prot}$ para satisfacer una subfórmula ψ, están acompañados de una *protección* P' que contiene una parte de $\mathcal{M}'$ suficiente para satisfacer ψ. En este caso, decimos que $\mathcal{M}'$ *está protegido por* P'. Una característica clave de $\text{XU}\textsc{pd}_{prot}$ es que si $\mathcal{M}$ está protegido por P y $\text{XU}\textsc{pd}_{prot}((\mathcal{M}, P), s, \psi)$ produce $(\mathcal{M}', P')$, entonces $\mathcal{M}'$ está protegido por P' y P' es una protección *mayor que o igual a* P (Definición 3.7.1). Es decir, $\text{XU}\textsc{pd}_{prot}$ preserva la satisfacción de las subfórmulas previamente tratadas.

$\textsc{Upd}_1((\mathcal{M}, Q), s, \varphi)$ % Actualiza $(\mathcal{M}, Q)$ respecto a φ en s
Input: $\mathcal{M} \in \mathbf{K}_\Sigma$, $Q \subseteq S^{\mathcal{M}} \times \Sigma$-CTL, $s \in S^{\mathcal{M}}$, y $\varphi \in \Sigma$-CTL
Output: *Modif*$(\mathcal{M}, s, \varphi)$ % ignorando las Q'
1. **case** φ **of**
2. $\quad$ F: **fail**
3. $\quad$ T: $(\mathcal{M}', Q') \leftarrow (\mathcal{M}, Q)$
4. $\quad$ ℓ: $\{\mathcal{M}' := \mathbf{L}^u(\mathcal{M}, s, \ell)$; $Q' := Q\}$
5. $\quad$ $\alpha \vee \beta$: {**guess** $\delta \in \{\alpha, \beta\}$;
6. $\quad\quad$ $(\mathcal{M}', Q') \leftarrow \textsc{Upd}_1((\mathcal{M}, Q), s, \delta)\}$
7. $\quad$ $\alpha \wedge \beta$: $\{(\mathcal{M}_\alpha, Q_\alpha) \leftarrow \textsc{Upd}_1((\mathcal{M}, Q), s, \alpha)$;
8. $\quad\quad$ $(\mathcal{M}', Q') \leftarrow \textsc{Upd}_1((\mathcal{M}_\alpha, Q_\alpha), s, \beta)\}$
9. $\quad$ $\mathbf{EX}\,\alpha$: {**guess** $t \in A^P[s]$;
10. $\quad\quad$ $(\mathcal{M}', Q') \leftarrow \textsc{Upd}_1\big(\,(\mathbf{T}^+_\exists(\mathcal{M}, s, t), Q)\,, t, \alpha\big)\}$
11. $\quad$ $\mathbf{AX}\,\alpha$: {**guess** $S' \in \{X \subseteq A^P[s] \mid X \neq \varnothing \;\&\; E^P[s] \subseteq X\}$;
12. $\quad\quad$ $(\mathcal{M}', Q') \leftarrow \textsc{Upd}_1^*\big(\,(\mathbf{T}^u_\forall(\mathcal{M}, s, S'), Q)\,, S', \alpha\big)\}$
13. $\quad$ $\mathbf{OD}_{\leq} n$: {**guess** $S' \in \{X \subseteq A^P[s] \mid X \neq \varnothing \;\&\; E^P[s] \subseteq X \;\&\; |X| \leq n\}$;
14. $\quad\quad$ $\mathcal{M}' \leftarrow \mathbf{T}^u_\forall(\mathcal{M}, s, S')$; $Q' := Q\}$
15. $\quad$ $\mathbf{OD}_{>} n$: {**guess** $S' \in \{X \subseteq A^P[s] \mid X \neq \varnothing \;\&\; E^P[s] \subseteq X \;\&\; |X| > n\}$;
16. $\quad\quad$ $\mathcal{M}' \leftarrow \mathbf{T}^u_\exists(\mathcal{M}, s, S')$; $Q' := Q\}$
17. $\quad$ $\mathbf{E}[\alpha\,\mathbf{U}\,\beta]$: **if** $(s, \varphi) \in Q$ **then fail** % default para lfp
18. $\quad\quad$ **else** $(\mathcal{M}', Q') \leftarrow \textsc{Upd}_1((\mathcal{M}, Q \cup \{(s, \varphi)\}), s, \beta \vee (\alpha \wedge \mathbf{EX}\,\varphi))$
19. $\quad$ $\mathbf{A}[\alpha\,\mathbf{U}\,\beta]$: **if** $(s, \varphi) \in Q$ **then fail** % default para lfp
20. $\quad\quad$ **else** $(\mathcal{M}', Q') \leftarrow \textsc{Upd}_1((\mathcal{M}, Q \cup \{(s, \varphi)\}), s, \beta \vee (\alpha \wedge \mathbf{AX}\,\varphi))$
21. $\quad$ $\mathbf{E}[\alpha\,\mathbf{R}\,\beta]$: **if** $(s, \varphi) \in Q$ **then** $(\mathcal{M}', Q') \leftarrow (\mathcal{M}, Q)$ % default para gfp
22. $\quad\quad$ **else** $(\mathcal{M}', Q') \leftarrow \textsc{Upd}_1((\mathcal{M}, Q \cup \{(s, \varphi)\}), s, \beta \wedge (\alpha \vee \mathbf{EX}\,\varphi))$
23. $\quad$ $\mathbf{A}[\alpha\,\mathbf{R}\,\beta]$: **if** $(s, \varphi) \in Q$ **then** $(\mathcal{M}', Q') \leftarrow (\mathcal{M}, Q)$ % default para gfp
24. $\quad\quad$ **else** $(\mathcal{M}', Q') \leftarrow \textsc{Upd}_1((\mathcal{M}, Q \cup \{(s, \varphi)\}), s, \beta \wedge (\alpha \vee \mathbf{AX}\,\varphi))$
25. **if** $\mathcal{M}', s \models \varphi$ **then return** $(\mathcal{M}', Q')$ **else fail**

Figura 3-4: pseudocódigo de $\textsc{Upd}_1$.

3.5.1. Modificación de modelos protegidos

Construimos nuestro algoritmo para actualización de modelos protegidos mediante el uso de unas cuantas operaciones básicas que permiten cambiar gradualmente el modelo de entrada.

Definición 3.5.1 (Operaciones de actualización protegidas)**.** Sean $(\mathcal{M}, P) \in \mathbf{KP}_\Sigma$, $s, s' \in S^\mathcal{M}$, $\ell \in \mathrm{Lit}(\mathbb{V})$, $S' \subseteq S^\mathcal{M}$, y $S'' \neq \varnothing$ un conjunto finito. Las *operaciones de actualización sobre modelos protegidos* son las siguientes:

1. Actualizar ℓ y proteger ℓ.

   $\mathbf{L}^u((\mathcal{M}, P), s, \ell) = \big(\mathbf{L}^u(\mathcal{M}, s, \ell), \langle E^P, A^P, L^P[s \oplus \ell]\rangle\big)$ *si* $\overline{\ell} \notin L^P(s)$.

2. Agregar (s, s') a $\mathcal{M}$ y agregar (s, s') a E^P.

   $\mathbf{T}^+_\exists((\mathcal{M}, P), s, s') = \big(\mathbf{T}^+(\mathcal{M}, s, s'), \langle E^P \cup \{(s, s')\}, A^P, L^P\rangle\big)$ *si* $s' \in A^P[s]$.

3. Reemplazar $R^\mathcal{M}[s]$ y $A^P[s]$ por S'.

   $\mathbf{T}^u_\forall((\mathcal{M}, P), s, S') = \big(\mathbf{T}^u(\mathcal{M}, s, S'), \langle E^P, A^P - (\{s\} \times (A^P[s] - S')), L^P\rangle\big)$

   si $E^P[s] \subseteq S' \subseteq A^P[s]$ y $S' \neq \varnothing$.

4. Reemplazar $R^\mathcal{M}[s]$ y $E^P[s]$ por S'.

   $\mathbf{T}^u_\exists((\mathcal{M}, P), s, S') = \big(\mathbf{T}^u(\mathcal{M}, s, S'), \langle E^P \cup (\{s\} \times (S' - E^P[s])), A^P, L^P\rangle\big)$

   si $E^P[s] \subseteq S' \subseteq A^P[s]$ y $S' \neq \varnothing$.

5. Agregar S'' a $\mathcal{M}$, y extender A^P y L^P.

   $\mathbf{S}^+_\forall((\mathcal{M}, P), S'') = \big(\mathbf{S}^+(\mathcal{M}, S''), \langle E^P, A^P \cup (S^\mathcal{M} \times S'') \cup \big(S'' \times (S^\mathcal{M} \cup S'')\big), L^P \cup \varnothing_{S''}\rangle\big)$

   si $S'' \cap S^\mathcal{M} = \varnothing$.

Dado que las funciones anteriores son parciales, asumimos que su aplicación en los casos no cubiertos por las definiciones correspondientes produce un valor indefinido ($\bot$). Observe que todas las operaciones anteriores, excepto $\mathbf{S}^+_\forall$, preservan el vocabulario Σ.

Extendemos a modelos protegidos nuestra definición de modificación de un modelo.

Definición 3.5.2 (Modificaciones de $(\mathcal{M}, P)$ respecto a φ)**.** Si $(\mathcal{M}, P) \in \mathbf{KP}_\Sigma$ es un modelo protegido, $s \in S^{\mathcal{M}}$ y $\varphi \in \Sigma$-XCTL, definimos, mediante recursión sobre φ, *el conjunto de modificaciones de* $(\mathcal{M}, P)$ *respecto a* φ *en* s, $\mathit{Modif}((\mathcal{M}, P), s, \varphi)$:

1. $\mathit{Modif}((\mathcal{M}, P), s, \mathsf{F}) = \varnothing$
2. $\mathit{Modif}((\mathcal{M}, P), s, \mathsf{T}) = \big\{(\mathcal{M}, P)\big\}$
3. $\mathit{Modif}((\mathcal{M}, P), s, \ell) = \big\{\mathbf{L}^u((\mathcal{M}, P), s, \ell)\big\}$
4. $\mathit{Modif}((\mathcal{M}, P), s, \alpha \vee \beta) = \mathit{Modif}((\mathcal{M}, P), s, \alpha) \cup \mathit{Modif}((\mathcal{M}, P), s, \beta)$
5. $\mathit{Modif}((\mathcal{M}, P), s, \alpha \wedge \beta) = \big\{(\mathcal{M}', P') \in \mathbf{KP}_\Sigma \mid \exists (\mathcal{M}_\alpha, P_\alpha) \in \mathit{Modif}((\mathcal{M}, P), s, \alpha).$ $(\mathcal{M}', P') \in \mathit{Modif}((\mathcal{M}_\alpha, P_\alpha), s, \beta)\big\}$
6. $\mathit{Modif}((\mathcal{M}, P), s, \mathbf{EX}\,\alpha) = \big\{(\mathcal{M}', P') \in \mathbf{KP}_\Sigma \mid \exists s' \in A^P[s].$ $(\mathcal{M}', P') \in \mathit{Modif}\big(\mathbf{T}^+_\exists((\mathcal{M}, P), s, s'), s', \alpha\big)\big\}$
7. $\mathit{Modif}((\mathcal{M}, P), s, \mathbf{AX}\,\alpha) = \big\{(\mathcal{M}', P') \in \mathbf{KP}_\Sigma \mid \exists S' \subseteq A^P[s].\ E^P[s] \subseteq S' \neq \varnothing$ $\&\ (\mathcal{M}', P') \in \mathit{Modif}^*\big(\mathbf{T}^u_\forall((\mathcal{M}, P), s, S'), S', \alpha\big)\big\}$
8. $\mathit{Modif}((\mathcal{M}, P), s, \mathbf{OD}_{\leq} n) = \big\{(\mathcal{M}', P') \in \mathbf{KP}_\Sigma \mid \exists S' \subseteq A^P[s].\ E^P[s] \subseteq S' \neq \varnothing$ $\&\ |S'| \leq n\ \&\ (\mathcal{M}', P') = \mathbf{T}^u_\forall((\mathcal{M}, P), s, S')\big\}$
9. $\mathit{Modif}((\mathcal{M}, P), s, \mathbf{OD}_{>} n) = \big\{(\mathcal{M}', P') \in \mathbf{KP}_\Sigma \mid \exists S' \subseteq A^P[s].\ E^P[s] \subseteq S' \neq \varnothing$ $\&\ |S'| > n\ \&\ (\mathcal{M}', P') = \mathbf{T}^u_\exists((\mathcal{M}, P), s, S')\big\}$

donde $\mathit{Modif}^*((\mathcal{M}, P), S', \varphi)$ *extiende la definición de* $\mathit{Modif}((\mathcal{M}, P), s, \varphi)$ *a un conjunto de estados* $S' \subseteq S^{\mathcal{M}}$:

$$\mathit{Modif}^*((\mathcal{M}, P), S', \varphi) = \begin{cases} \big\{(\mathcal{M}, P)\big\} & \text{si } S' = \varnothing \\ \big\{(\mathcal{M}', P') \in \mathbf{KP}_\Sigma \mid \exists t \in S'. & \\ \quad \exists (\mathcal{M}_t, P_t) \in \mathit{Modif}\big((\mathcal{M}, P), t, \varphi\big). & \\ \quad (\mathcal{M}', P') \in \mathit{Modif}^*\big((\mathcal{M}_t, P_t), S' - \{t\}, \varphi\big)\big\} & \text{si } S' \neq \varnothing \end{cases}$$

3.5.2. XUPD$_{prot}$, actualización con protecciones para Σ-XCTL

A continuación, definimos un algoritmo para actualizar modelos protegidos. Intuitivamente, XUPD$_{prot}((\mathcal{M}, P), s, \varphi)$ encuentra modelos para φ, en el estado s, modificando $\mathcal{M}$ y respetando la protección P. La llamada inicial a XUPD$_{prot}$ puede usar la protección vacía $P_\perp$.

La Figura 3-5 muestra el pseudocódigo de XUPD$^*_{prot}$, un procedimiento auxiliar en la definición posterior de XUPD$_{prot}$. El procedimiento XUPD$^*_{prot}((\mathcal{M}, P), S', \varphi)$ es una implementación de $Modif^*((\mathcal{M}, P), S', \varphi)$ que, intuitivamente, actualiza el modelo protegido $(\mathcal{M}, P)$, respecto a una fórmula $\varphi \in \Sigma$-XCTL, en un conjunto de estados S'.

XUPD$^*_{prot}((\mathcal{M}, P), S', \varphi)$ % Actualiza $(\mathcal{M}, P)$ en S' respecto a φ
Input: $(\mathcal{M}, P) \in \mathbf{KP}_\Sigma$, $S' \subseteq S^{\mathcal{M}}$, $\varphi \in \mathbf{XCTL}$
Output: $Modif^*((\mathcal{M}, P), S', \varphi)$
1. **if** $S' = \varnothing$ **then** $(\mathcal{M}', P') := (\mathcal{M}, P)$
2. **else** {**pick** $t \in S'$ **;**
3. $(\mathcal{M}_t, P_t) \leftarrow$ XUPD$_{prot}((\mathcal{M}, P), t, \varphi)$ **;**
4. $(\mathcal{M}', P') \leftarrow$ XUPD$^*_{prot}((\mathcal{M}_t, P_t), S' - \{t\}, \varphi)$}
5. **return** $(\mathcal{M}', P')$

Figura 3-5: pseudocódigo de XUPD$^*_{prot}$.

Observe que, para actualizar en un conjunto de estados S', el procedimiento XUPD$^*_{prot}$ llama a XUPD$_{prot}$ con cada uno de los elementos de S' (línea 3). Es decir, XUPD$^*_{prot}$ y XUPD$_{prot}$ son procedimientos mutuamente recursivos.

En la Figura 3-6 mostramos el pseudocódigo del algoritmo XUPD$_{prot}$, una implementación de $Modif((\mathcal{M}, P), s, \varphi)$.

Observe que XUPD$_{prot}$ no necesita hacer una verificación similar a la verificación que se realiza en la linea (15) de XUPD$_1$. El mecanismo de protecciones garantiza que el modelo regresado, $(\mathcal{M}', P')$, cumpla el requisito $(\mathcal{M}', P'), s \models \varphi$. Primero, tal requisito se cumple cuando XUPD$_{prot}$ se aplica a fórmulas básica (T, ℓ, **OD**). Luego, las

$\text{XUPD}_{prot}((\mathcal{M}, P), s, \varphi)$ % Actualiza $(\mathcal{M}, P)$ respecto a φ en s
Input: $(\mathcal{M}, P) \in \mathbf{KP}_\Sigma$, $s \in S^{\mathcal{M}}$, y $\varphi \in \Sigma$-XCTL
Output: $\mathit{Modif}((\mathcal{M}, P), s, \varphi)$
1. **if** $(\mathcal{M}, P), s \models \varphi$ **then** $(\mathcal{M}', P') \leftarrow (\mathcal{M}, P)$
2. **else case** φ **of**
3. F: **fail**
4. T: $(\mathcal{M}', P') \leftarrow (\mathcal{M}, P)$
5. ℓ: **if** $\overline{\ell} \in L^P(s)$ **then fail**
6. **else** $(\mathcal{M}', P') := \mathbf{L}^u((\mathcal{M}, P), s, \ell)$
7. $\alpha \vee \beta$: {**guess** $\delta \in \{\alpha, \beta\}$ **;**
8. $(\mathcal{M}', P') \leftarrow \text{XUPD}_{prot}((\mathcal{M}, P), s, \delta)\}$
9. $\alpha \wedge \beta$: $\{(\mathcal{M}_\alpha, P_\alpha) \leftarrow \text{XUPD}_{prot}((\mathcal{M}, P), s, \alpha)$ **;**
10. $(\mathcal{M}', P') \leftarrow \text{XUPD}_{prot}((\mathcal{M}_\alpha, P_\alpha), s, \beta)\}$
11. **EX** α: {**guess** $s' \in A^P[s]$ **;**
12. $(\mathcal{M}', P') \leftarrow \text{XUPD}_{prot}(\mathbf{T}^+_\exists((\mathcal{M}, P), s, s'), s', \alpha)\}$
13. **AX** α: {**guess** $S' \in \{X \subseteq A^P[s] \mid X \neq \varnothing \ \& \ E^P[s] \subseteq X\}$ **;**
14. $(\mathcal{M}', P') \leftarrow \text{XUPD}^*_{prot}(\mathbf{T}^u_\forall((\mathcal{M}, P), s, S'), S', \alpha)\}$
15. $\mathbf{OD}_{\leq} n$: {**guess** $S' \in \{X \subseteq A^P[s] \mid X \neq \varnothing \ \& \ E^P[s] \subseteq X \ \& \ |X| \leq n\}$ **;**
16. $(\mathcal{M}', P') \leftarrow \mathbf{T}^u_\forall((\mathcal{M}, P), s, S')\}$
17. $\mathbf{OD}_{>} n$: {**guess** $S' \in \{X \subseteq A^P[s] \mid X \neq \varnothing \ \& \ E^P[s] \subseteq X \ \& \ |X| > n\}$ **;**
18. $(\mathcal{M}', P') \leftarrow \mathbf{T}^u_\exists((\mathcal{M}, P), s, S')\}$
19. **return** $(\mathcal{M}', P')$

Figura 3-6: pseudocódigo de XUPD_{prot}.

aplicaciones posteriores de XUPD_{prot} usan las protecciones para preservar la satisfacción de las subfórmulas previamente tratadas. En comparación con XUPD_1, el mecanismo de protecciones de XUPD_{prot} no sólo evita verificar los modelos que genera, también reduce el número posible de modelos generados, ya que sólo se generan los modelos que respetan la protección actual. En el Capítulo 4 mostramos una comparación entre XUPD_{prot} y XUPD_1 mediante ejemplos.

3.5.3. UPD_{prot}, actualización mediante protecciones para Σ-CTL

Extendemos XUPD_{prot} a fórmulas que utilizan los operadores **EU**, **AU**, **ER**, y **AR**. Estos operadores se sustituyen por sus caracterizaciones de punto fijo (ecuaciones 3-3) y luego se tratan mediante un mecanismo para detección de ciclos. Este mecanismo para detección de ciclos está controlado por un parámetro $Q \subseteq S^{\mathcal{M}} \times \Sigma\text{-CTL}$. La intuición detrás de Q es que si $(s, \psi) \in Q$, entonces el estado s ya ha fue visitado y actualizado respecto a ψ.

La Figura 3-7 muestra el pseudocódigo de UPD^*_{prot}, un procedimiento auxiliar en la definición posterior de UPD_{prot}. El procedimiento $\text{UPD}^*_{prot}((\mathcal{M}, P), S', \varphi)$ es una implementación de $Modif^*((\mathcal{M}, P), S', \varphi)$ que, intuitivamente, actualiza el modelo protegido $(\mathcal{M}, P)$, respecto a una fórmula $\varphi \in \Sigma\text{-CTL}$, en un conjunto de estados S'.

$\text{UPD}^*_{prot}((\mathcal{M}, P, Q), S', \varphi)$ % Actualiza $(\mathcal{M}, P)$ en S' respecto a φ
Input: $(\mathcal{M}, P) \in \mathbf{KP}_\Sigma$, $Q \subseteq S^{\mathcal{M}} \times \Sigma\text{-CTL}$, $S' \subseteq S^{\mathcal{M}}$, $\varphi \in$ **CTL**
Output: $Modif^*((\mathcal{M}, P), S', \varphi)$ % ignorando las Q'
1. **if** $S' = \varnothing$ **then** $(\mathcal{M}', P') := (\mathcal{M}, P)$
2. **else** {**pick** $t \in S'$ **;**
3. $(\mathcal{M}_t, P_t, Q_t) \leftarrow \text{UPD}_{prot}((\mathcal{M}, P, Q), t, \varphi)$ **;**
4. $(\mathcal{M}', P', Q') \leftarrow \text{UPD}^*_{prot}((\mathcal{M}_t, P_t, Q_t), S' - \{t\}, \varphi)\}$
5. **return** $(\mathcal{M}', P', Q')$

Figura 3-7: pseudocódigo de UPD^*_{prot}.

Observe que, para actualizar en un conjunto de estados S', UPD^*_{prot} llama a UPD_{prot} con cada uno de los elementos de S' (línea 3). Es decir, UPD^*_{prot} y UPD_{prot} son procedimientos mutuamente recursivos.

El pseudocódigo de UPD_{prot} (Figura 3-8) muestra cómo se extiende el pseudocódigo de XUPD_{prot} a los casos de **EU**, **AU**, **ER** y **AR**. La llamada inicial a UPD_{prot} utiliza un conjunto vacío Q de estados visitados y puede usar la protección vacía $P_\perp$.

$\textsc{Upd}_{prot}((\mathcal{M}, P, Q), s, \varphi)$ % Actualiza $(\mathcal{M}, P, Q)$ respecto a φ en s
Input: $(\mathcal{M}, P) \in \mathbf{KP}_\Sigma$, $Q \subseteq S^{\mathcal{M}} \times \Sigma$-CTL, $s \in S^{\mathcal{M}}$, y $\varphi \in \Sigma$-CTL
Output: $\mathit{Modif}((\mathcal{M}, P), s, \varphi)$ % ignorando las Q'
1. **if** $(\mathcal{M}, P), s \models \varphi$ **then** $(\mathcal{M}', P', Q') \leftarrow (\mathcal{M}, P, Q)$
2. **else case** φ **of**
3. F: **fail**
4. T: $(\mathcal{M}', P', Q') \leftarrow (\mathcal{M}, P, Q)$
5. ℓ: **if** $\overline{\ell} \in L^P(s)$ **then fail**
6. **else** $\{(\mathcal{M}', P') := \mathbf{L}^u((\mathcal{M}, P), s, \ell)$ **;** $Q' := Q\}$
7. $\alpha \vee \beta$: {**guess** $\delta \in \{\alpha, \beta\}$ **;**
8. $(\mathcal{M}', P', Q') \leftarrow \textsc{Upd}_{prot}((\mathcal{M}, P, Q), s, \delta)\}$
9. $\alpha \wedge \beta$: $\{(\mathcal{M}_\alpha, P_\alpha, Q_\alpha) \leftarrow \textsc{Upd}_{prot}((\mathcal{M}, P, Q), s, \alpha)$ **;**
10. $(\mathcal{M}', P', Q') \leftarrow \textsc{Upd}_{prot}((\mathcal{M}_\alpha, P_\alpha, Q_\alpha), s, \beta)\}$
11. **EX** α: {**guess** $t \in A^P[s]$ **;**
12. $(\mathcal{M}_t, P_t) \leftarrow \mathbf{T}^+_\exists((\mathcal{M}, P), s, t)$ **;**
13. $(\mathcal{M}', P', Q') \leftarrow \textsc{Upd}_{prot}((\mathcal{M}_t, P_t, Q), t, \alpha)\}$
14. **AX** α: {**guess** $S' \in \{X \subseteq A^P[s] \mid X \neq \varnothing \ \& \ E^P[s] \subseteq X\}$ **;**
15. $(\mathcal{M}_{S'}, P_{S'}) \leftarrow \mathbf{T}^u_\forall((\mathcal{M}, P), s, S')$
16. $(\mathcal{M}', P', Q') \leftarrow \textsc{Upd}^*_{prot}((\mathcal{M}_{S'}, P_{S'}, Q), S', \alpha)\}$
17. **OD**$_\leq n$: {**guess** $S' \in \{X \subseteq A^P[s] \mid X \neq \varnothing \ \& \ E^P[s] \subseteq X \ \& \ |X| \leq n\}$ **;**
18. $(\mathcal{M}', P') \leftarrow \mathbf{T}^u_\forall((\mathcal{M}, P), s, S')$ **;** $Q' := Q\}$
19. **OD**$_> n$: {**guess** $S' \in \{X \subseteq A^P[s] \mid X \neq \varnothing \ \& \ E^P[s] \subseteq X \ \& \ |X| > n\}$ **;**
20. $(\mathcal{M}', P') \leftarrow \mathbf{T}^u_\exists((\mathcal{M}, P), s, S')$ **;** $Q' := Q\}$
21. **E**$[\alpha \, \mathbf{U} \, \beta]$: **if** $(s, \varphi) \in Q$ **then fail** % default para lfp
22. **else** $(\mathcal{M}', P', Q') \leftarrow \textsc{Upd}_{prot}((\mathcal{M}, P, Q \cup \{(s, \varphi)\}), s, \beta \vee (\alpha \wedge$ **EX** $\varphi))$
23. **A**$[\alpha \, \mathbf{U} \, \beta]$: **if** $(s, \varphi) \in Q$ **then fail** % default para lfp
24. **else** $(\mathcal{M}', P', Q') \leftarrow \textsc{Upd}_{prot}((\mathcal{M}, P, Q \cup \{(s, \varphi)\}), s, \beta \vee (\alpha \wedge$ **AX** $\varphi))$
25. **E**$[\alpha \, \mathbf{R} \, \beta]$: **if** $(s, \varphi) \in Q$
26. **then** $(\mathcal{M}', P', Q') \leftarrow (\mathcal{M}, P, Q)$ % default para gfp
27. **else** $(\mathcal{M}', P', Q') \leftarrow \textsc{Upd}_{prot}((\mathcal{M}, P, Q \cup \{(s, \varphi)\}), s, \beta \wedge (\alpha \vee$ **EX** $\varphi))$
28. **A**$[\alpha \, \mathbf{R} \, \beta]$: **if** $(s, \varphi) \in Q$
29. **then** $(\mathcal{M}', P', Q') \leftarrow (\mathcal{M}, P, Q)$ % default para gfp
30. **else** $(\mathcal{M}', P', Q') \leftarrow \textsc{Upd}_{prot}((\mathcal{M}, P, Q \cup \{(s, \varphi)\}), s, \beta \wedge (\alpha \vee$ **AX** $\varphi))$
31. **return** $(\mathcal{M}', P', Q')$

Figura 3-8: pseudocódigo de $\textsc{Upd}_{prot}$.

3.5.4. XUPD$_{S+}$, actualización con adición de estados

Para actualizar un modelo con vocabulario Σ, respecto a una fórmula φ en s, la operación que agrega estados sólo es necesaria cuando no existe un modelo M', con el mismo vocabulario Σ, tal que M' satisface a φ en s. Por ejemplo, si $\Sigma = \langle\{s_0\}, \{p\}\rangle$, $(\mathcal{M}, P)$ es cualquier Σ-modelo protegido, y $\varphi = \mathbf{EX}\, p \wedge \mathbf{EX}\, \neg p$, entonces los modelos con vocabulario Σ tienen solamente un estado, pero para satisfacer a φ se requiere un modelo con al menos dos estados, uno etiquetado con p y otro etiquetado con $\neg p$. Por lo tanto, ya que XUPD$_{prot}((\mathcal{M}, P), s_0, \varphi)$ sólo produce modelos con vocabulario Σ, XUPD$_{prot}[(\mathcal{M}, P), s_0, \varphi] = \varnothing$. En este tipo de situaciones, y suponiendo que $n = |\varphi|$, podemos extender nuestro algoritmo de actualización para que sea capaz de añadir estados al modelo de entrada hasta que XUPD$_{prot}[(\mathcal{M}, P), s, \varphi] \neq \varnothing$ o $|S^{\mathcal{M}}| > n8^n$. Esta cota sobre el número de estados está justificada por un teorema que establece la propiedad de modelos finitos para CTL (*a small model theorem for CTL*) [18, p. 9]: si φ es satisfactible entonces φ es satisfactible en un modelo de tamaño menor o igual a $n8^n$, donde el *tamaño* de un modelo $\mathcal{M}$ es $|S^{\mathcal{M}}|$ [18, p. 4].

El algoritmo de la Figura 3-9, XUPD$_{S+}$, muestra cómo agregar estados en el proceso de actualización.

XUPD$_{S+}((\mathcal{M}, P), s, \varphi)$ % Actualiza $(\mathcal{M}, P)$ agregando estados si es necesario
Input: $(\mathcal{M}, P) \in \mathbf{KP}_\Sigma$, $s \in S^{\mathcal{M}} = \{s_1, \ldots, s_m\}$, $\varphi \in \Sigma$-XCTL
Output: $\mathit{Modif}(\mathbf{S}^+_\forall((\mathcal{M}, P), S^{\Sigma'} - \mathbb{S}), s, \varphi)$
Donde Σ' es el vocabulario minimal tal que $\Sigma' \supseteq \Sigma$ y:
$\mathit{Modif}(\mathbf{S}^+_\forall((\mathcal{M}, P), S^{\Sigma'} - \mathbb{S}), s, \varphi) \neq \varnothing$ o $(n = |\varphi|$ y $|S^{\Sigma'}| > n8^n)$.
1. $n \leftarrow |\varphi|$; $m' \leftarrow m$; % Recordar que $S^{\mathcal{M}} = \{s_1, \ldots, s_m\}$
2. **while** XUPD$_{prot}[(\mathcal{M}, P), s, \varphi] = \varnothing$ **and** $m' \leq n8^n$ **do**
3. $\quad\{m' \leftarrow |S^M| + 1$;
4. $\quad s' \leftarrow s_{m'}$; % s' es un estado nuevo, $s' \notin S^M$
5. $\quad(\mathcal{M}, P) \leftarrow \mathbf{S}^+_\forall((\mathcal{M}, P), \{s'\})\}$
6. $(\mathcal{M}', P') \leftarrow$ XUPD$_{prot}((\mathcal{M}, P), s, \varphi)$; % "$\leftarrow$" falla si XUPD$_{prot}$ falla
7. **return** $(\mathcal{M}', P')$

Figura 3-9: pseudocódigo de XUPD$_{S+}$.

3.6. Heurísticas y estrategias de búsqueda

Una característica significativa de $\textsc{Upd}_{prot}$ (y $\textsc{Upd}_1$) es el no determinismo. Las implementaciones deterministas directas de algoritmos no deterministas, sin embargo, pueden sufrir de ineficiencia. Para mejorar la eficiencia de $\textsc{Upd}_{prot}$, proponemos heurísticas basadas en un orden sobre la generación de elecciones no deterministas entre los elementos de: (1) un conjunto de estados, (2) un conjunto de conjuntos de estados, y (3) un conjunto de dos fórmulas de CTL. Para encontrar soluciones minimales, proponemos una estrategia de búsqueda similar a la profundización iterativa.

Para fines prácticos, hacemos algunas suposiciones. Asumimos que un multiconjunto finito A se identifica con una lista finita que contiene los elementos de A en un orden arbitrario, por ejemplo, $A = [a_1, a_2, \ldots, a_n]$. También suponemos que la ejecución de la instrucción "**guess** $x \in A$" produce elecciones no deterministas en el orden de la lista que identificada con A, es decir, la primera elección es a_1, luego a_2, y así sucesivamente. Utilizamos $+$ y $\sum$ para representar la concatenación de dos listas y la concatenación de una secuencia finita de listas, respectivamente. Bajo estos supuestos, mostramos la forma de mejorar la eficiencia de $\text{XU}\textsc{pd}_1(\mathcal{M}, s, \varphi)$ en los casos en que $\varphi = \mathbf{EX}\,\alpha$, $\varphi = \mathbf{AX}\,\alpha$, o $\varphi = \alpha \vee \beta$ (las mejoras para $\text{XU}\textsc{pd}_{prot}$ son análogas).

3.6.1. Elecciones no deterministas de estados

Para mejorar la eficiencia de $\text{XU}\textsc{pd}_1(\mathcal{M}, s, \mathbf{EX}\,\alpha)$ utilizamos un orden en la generación de elecciones no deterministas entre los elementos de un conjunto de estados. En la línea (9) de $\text{XU}\textsc{pd}_1$, sustituimos la instrucción "**guess** $s' \in S^{\mathcal{M}}$" por "**guess** $s' \in R^{\mathcal{M}}[s] + \big(S^{\mathcal{M}} - R^{\mathcal{M}}[s]\big)$". Así, las primeras elecciones no deterministas serán elementos de $R^M[s]$.

3.6.2. Elecciones no deterministas de conjuntos de estados

Para mejorar la eficiencia de $\text{XUPD}_1(\mathcal{M}, s, \mathbf{AX}\,\alpha)$ utilizamos un orden en la generación de elecciones no deterministas entre los elementos de un conjunto de conjuntos de estados.

Sea $m = |S^{\mathcal{M}}|$, y para $i = 0, \ldots, m$ sea

$Q_i(\mathcal{M}, s) = \{S' \subseteq S^{\mathcal{M}} \mid S' \neq \varnothing \ \ \& \ \ |\Delta(S', R^{\mathcal{M}}[s])| = i\}$.

Por lo tanto, si $S' \in Q_i(\mathcal{M}, s)$ entonces $d(T^u(\mathcal{M}, s, S'), \mathcal{M}) = i$.

Se puede demostrar que los elementos de $\sum_{i=0}^{m} Q_i(\mathcal{M}, s)$ son conjuntos, identificados con listas, que representan a los subconjuntos no vacíos de $S^{\mathcal{M}}$. Además, si S_1 y S_2 son dos elementos de la lista $\sum_{i=0}^{m} Q_i(\mathcal{M}, s)$ y S_1 ocurre antes que S_2 en dicha lista, entonces $d(T^u(\mathcal{M}, s, S_1), \mathcal{M}) \leq d(T^u(\mathcal{M}, s, S_2), \mathcal{M})$.

En la línea (11) de XUPD_1, sustituimos la instrucción "**guess** $S' \in \mathcal{P}(S^{\mathcal{M}}) - \{\varnothing\}$" por la instrucción "**guess** $S' \in \sum_{i=0}^{m} Q_i(\mathcal{M}, s)$".

3.6.3. Elecciones no deterministas de fórmulas

Para mejorar la eficiencia de $\text{XUPD}_1(\mathcal{M}, s, \alpha \vee \beta)$ utilizamos un orden en la generación de elecciones no deterministas entre los elementos de un conjunto de dos fórmulas de la CTL. Con este fin, identificamos el conjunto $\{\alpha, \beta\}$ con la lista $[\delta_1, \delta_2]$, donde $(\delta_1, \delta_2) = (\alpha, \beta)$ si $|\alpha| \leq |\beta|$, y $(\delta_1, \delta_2) = (\beta, \alpha)$ en caso contrario. Así, en la línea (5) de XUPD_1, sustituimos la instrucción "**guess** $\delta \in \{\alpha, \beta\}$" por la instrucción "**guess** $\delta \in [\delta_1, \delta_2]$".

3.6.4. Limitando el número de cambios

Además de mejorar la eficiencia, las heurísticas anteriores ayudan a que UPD_{prot} produzca primero modelos que son cercanos al modelo de entrada. Las heurísticas anteriores, sin embargo, no garantizan que UPD_{prot} produce primero soluciones minimales. Por lo tanto, para los casos que requieren una solución minimal, proponemos una variante de UPD_{prot}, UPD_{min}, que utiliza una estrategia de búsqueda similar a profundización iterativa.

Obtenemos una implementación de UPD_{min} modificando UPD_{prot} de la siguiente manera. Primero, añadimos a UPD_{prot} un parámetro, n, para registrar el número de *cambios aplicados* al modelo de entrada. También incluimos una variable, w, para fijar el número máximo de *cambios permitidos*. Segundo, modificamos UPD_{prot} para calcular el número de *cambios requeridos*, r, antes de aplicar una operación de actualización. Tercero, modificamos UPD_{prot} para que se detenga si $n + r > w$; es decir, UPD_{prot} se detiene si el número de cambios ya realizados, más el número de cambios que aplicará, es mayor que el número de cambios permitidos. Finalmente, para producir primero las soluciones mínimas, inicializamos w a cero e iteramos UPD_{prot}, incrementando w en uno, hasta UPD_{prot} produce una solución.

3.7. Corrección y completitud

Incluimos aquí una demostración de la corrección y completitud de XUPD_{prot}. La aplicación de XUPD_{prot} a un modelo protegido $(\mathcal{M}, P)$ debe producir modelos protegidos $(\mathcal{M}', P')$ tales que P' es *mayor que o igual a* P. Por lo tanto, empezamos por definir un orden parcial sobre protecciones adecuado para actualización de modelos.

Definición 3.7.1 (Relación $\succeq$)**.** Si $P, P' \in \mathbf{P}_\Sigma$ son dos protecciones, entonces decimos que P' es *mayor que o igual a* P, y escribimos $P' \succeq P$, si $E^{P'} \supseteq E^P$, $A^{P'} \subseteq A^P$, y $\forall t \in \mathbb{S} : L^{P'}(t) \supseteq L^P(t)$.

Teorema 3.7.2. *La relación* $\succeq$ *es un orden parcial sobre* $\mathbf{P}_\Sigma$.

Demostración. Note que la relación $\succeq$ se define mediante los ordenes parciales $\subseteq$ y $\supseteq$.

A continuación demostramos que $\succeq$ es reflexiva, antisimétrica, y transitiva:

1. Si $P \in \mathbf{P}_\Sigma$, entonces:

   $E^P \supseteq E^P$,

   $A^P \subseteq A^P$,

y $\forall t \in \mathbb{S} : L^P(t) \supseteq L^P(t)$.

Por lo tanto, $P \succeq P$, y $\succeq$ es reflexiva.

2. Si $P_1, P_2 \in \mathbf{P}_\Sigma$, $P_1 \succeq P_2$, y $P_2 \succeq P_1$, entonces

   a) $E^{P_1} \supseteq E^{P_2}$ y $E^{P_2} \supseteq E^{P_1}$

   $\therefore E^{P_1} = E^{P_2}$

   b) $A^{P_1} \subseteq A^{P_2}$ y $A^{P_2} \subseteq A^{P_1}$

   $\therefore A^{P_1} = A^{P_2}$

   c) $\forall t \in \mathbb{S} : L^{P_1}(t) \supseteq L^{P_2}(t)$, y $\forall t \in \mathbb{S} : L^{P_2}(t) \supseteq L^{P_1}(t)$.

   $\therefore \forall t \in \mathbb{S} : L^{P_1}(t) = L^{P_2}(t)$.

   $\therefore L^{P_1} = L^{P_2}$

   $\therefore P_1 = P_2$.

   Por lo tanto $\succeq$ es antisimétrica.

3. Si $P_1, P_2, P_3 \in \mathbf{P}_\Sigma$, $P_1 \succeq P_2$, y $P_2 \succeq P_3$, entonces

   a) $E^{P_1} \supseteq E^{P_2}$ y $E^{P_2} \supseteq E^{P_3}$

   $\therefore E^{P_1} \supseteq E^{P_3}$

   b) $A^{P_1} \subseteq A^{P_2}$ y $A^{P_2} \subseteq A^{P_3}$

   $\therefore A^{P_1} \subseteq A^{P_3}$

   c) $\forall t \in \mathbb{S} : L^{P_1}(t) \supseteq L^{P_2}(t)$, y $\forall t \in \mathbb{S} : L^{P_2}(t) \supseteq L^{P_3}(t)$.

   $\therefore \forall t \in \mathbb{S} : L^{P_1}(t) \supseteq L^{P_3}(t)$.

   $\therefore P_1 \succeq P_3$.

   Por lo tanto $\succeq$ es transitiva.

Por lo tanto $\succeq$ es un orden parcial sobre $\mathbf{P}_\Sigma$. □

Las operaciones de actualización protegidas, cuando son aplicables, producen un modelo protegido con una protección mayor que o igual a la protección del modelo de entrada. Es decir, estas operaciones no disminuyen protecciones.

Teorema 3.7.3 (Las operaciones no disminuyen las protecciones)**.** *Dados* $(\mathcal{M}, P) \in \mathbf{KP}_\Sigma$*,* $s, s' \in S^{\mathcal{M}}$*,* $\ell \in \mathrm{Lit}(\mathbb{V})$*, y* $S' \subseteq S^{\mathcal{M}}$*, considere las siguientes condiciones:*

1. $\overline{\ell} \notin L^P(s)$ *y* $(\mathcal{M}', P') = \mathbf{L}^u((\mathcal{M}, P), s, \ell)$
2. $s' \in A^P[s]$ *y* $(\mathcal{M}', P') = \mathbf{T}^+_\exists((\mathcal{M}, P), s, s')$
3. $E^P[s] \subseteq S' \subseteq A^P[s]$*,* $S' \neq \varnothing$*, y* $(\mathcal{M}', P') = \mathbf{T}^u_\forall((\mathcal{M}, P), s, S')$
4. $E^P[s] \subseteq S' \subseteq A^P[s]$*,* $S' \neq \varnothing$*, y* $(\mathcal{M}', P') = \mathbf{T}^u_\exists((\mathcal{M}, P), s, S')$

Si (1)*, o* (2)*, o* (3)*, o* (4)*, entonces* $(\mathcal{M}', P') \in \mathbf{KP}_\Sigma$ *y* $P' \succeq P$*.*

Demostración. (De acuerdo con las condiciones anteriores 1-4)

1. Si $\overline{\ell} \notin L^P(s)$ y $(\mathcal{M}', P') = \mathbf{L}^u((\mathcal{M}, P), s, \ell)$, entonces

   $(\mathcal{M}', P') = (\mathbf{L}^u(\mathcal{M}, s, \ell), \langle E^P, A^P, L^P[s \oplus \ell]\rangle)$.

   Por lo tanto, $\mathcal{M}' = \mathbf{L}^u(\mathcal{M}, s, \ell) = \langle S^{\mathcal{M}}, R^{\mathcal{M}}, L^{\mathcal{M}}[s \oplus \ell]\rangle$, y

   $P' = \langle E^P, A^P, L^P[s \oplus \ell]\rangle$.

   Es decir, la única diferencia entre $\mathcal{M}'$ y $\mathcal{M}$, y entre P' y P está en el tercer componente.

   a) Puesto que $\mathcal{M} \rhd P$, tenemos $E^P \subseteq R^{\mathcal{M}} \subseteq A^P$, y $\forall t \in S^{\mathcal{M}} : L^{\mathcal{M}}(t) \supseteq L^P(t)$.

   Así, para concluir que $\mathcal{M}' \rhd P'$, sólo falta demostrar que

   $\forall t \in S^{\mathcal{M}} : L^{\mathcal{M}}[s \oplus \ell](t) \supseteq L^P[s \oplus \ell](t)$.

   Si $t \neq s$ entonces $L^{\mathcal{M}}[s \oplus \ell](t) = L^{\mathcal{M}}(t) \supseteq L^P(t) = L^P[s \oplus \ell](t)$.

   Si $t = s$ entonces

   $L^{\mathcal{M}}[s \oplus \ell](t) = \big(L^{\mathcal{M}}(s) \cup \{\ell\}\big) - \{\overline{\ell}\} \supseteq \big(L^P(s) \cup \{\ell\}\big) - \{\overline{\ell}\} = L^P[s \oplus \ell](t)$.

   Por lo tanto, $\mathcal{M}' \rhd P'$ y $(\mathcal{M}', P') \in \mathbf{KP}_\Sigma$.

b) Además, $E^{P'} = E^P \supseteq E^P$, y $A^{P'} = A^P \subseteq A^P$.

Así, para concluir que $P' \succeq P$, sólo falta demostrar que $\forall t \in \mathbb{S} : L^{P'}(t) \supseteq L^P(t)$.

Si $t \neq s$ entonces $L^{P'}(t) = L^P[s \oplus \ell](t) = L^P(t) \supseteq L^P(t)$.

Si $t = s$ entonces

$L^{P'}(t) = L^P[s \oplus \ell](t) = (L^P(s) \cup \{\ell\}) - \{\overline{\ell}\} \supseteq L^P(s) = L^P(t)$, porque $\overline{\ell} \notin L^P(s)$.

Por lo tanto, $P' \succeq P$.

2. Si $s' \in A^P[s]$ y $(\mathcal{M}', P') = \mathbf{T}^+_{\exists}((\mathcal{M}, P), s, s')$, entonces

$(\mathcal{M}', P') = (\mathbf{T}^+(\mathcal{M}, s, s'), \langle E^P \cup \{(s, s')\}, A^P, L^P\rangle)$

Por lo tanto, $\mathcal{M}' = \mathbf{T}^+(\mathcal{M}, s, s') = \langle S^{\mathcal{M}}, R^{\mathcal{M}} \cup \{(s, s')\}, L^{\mathcal{M}}\rangle$, y

$P' = \langle E^P \cup \{(s, s')\}, A^P, L^P\rangle$.

Consecuentemente, la única diferencia entre $\mathcal{M}'$ y $\mathcal{M}$, y entre P' y P está en el primer y segundo componentes, respectivamente.

a) Ya que $\mathcal{M} \rhd P$, tenemos $E^P \subseteq R^{\mathcal{M}} \subseteq A^P$, y $\forall t \in S^{\mathcal{M}} : L^{\mathcal{M}}(t) \supseteq L^P(t)$.

Así, para concluir que $\mathcal{M}' \rhd P'$, sólo falta demostrar que $E^{P'} \subseteq R^{\mathcal{M}'} \subseteq A^{P'}$.

Puesto que $s' \in A^P[s]$,

$E^{P'} = E^P \cup \{(s, s')\} \subseteq R^{\mathcal{M}} \cup \{(s, s')\} = R^{\mathcal{M}'} \subseteq A^P \cup \{(s, s')\} = A^{P'}$.

Consecuentemente, $\mathcal{M}' \rhd P'$ y $(\mathcal{M}', P') \in \mathbf{KP}_\Sigma$.

b) Además, tenemos $A^{P'} = A^P \subseteq A^P$, y $\forall t \in \mathbb{S} : L^{P'}(t) = L^P(t) \supseteq L^P(t)$.

Así, para concluir que $P' \succeq P$, sólo falta demostrar que $E^{P'} \supseteq E^P$.

Claramente, $E^{P'} = E^P \cup \{(s, s')\} \supseteq E^P$.

Por lo tanto, $P' \succeq P$.

3. Si $E^P[s] \subseteq S' \subseteq A^P[s]$, $S' \neq \varnothing$, y $(\mathcal{M}', P') = \mathbf{T}^u_{\forall}((\mathcal{M}, P), s, S')$, entonces

$(\mathcal{M}', P') = (\mathbf{T}^u(\mathcal{M}, s, S'), \langle E^P, A^P - (\{s\} \times (A^P[s] - S'))), L^P\rangle)$.

Por lo tanto:

$\mathcal{M}' = \mathbf{T}^u(\mathcal{M}, s, S') = \langle S^{\mathcal{M}}, (R^{\mathcal{M}} - (\{s\} \times R^{\mathcal{M}}[s])) \cup (\{s\} \times S'), L^{\mathcal{M}} \rangle$, y

$P' = \langle E^P, A^P - (\{s\} \times (A^P[s] - S')), L^P \rangle$.

Consecuentemente, la única diferencia entre $\mathcal{M}'$ y $\mathcal{M}$, y entre P' y P, está en el segundo componente.

a) Ya que $\mathcal{M} \rhd P$, tenemos que $E^P \subseteq R^{\mathcal{M}} \subseteq A^P$, y $\forall t \in S^{\mathcal{M}} : L^{\mathcal{M}}(t) \supseteq L^P(t)$.

Por lo tanto, $\forall t \in S^{\mathcal{M}'} = S^{\mathcal{M}},\ L^{\mathcal{M}'}(t) = L^{\mathcal{M}}(t) \supseteq L^P(t) = L^{P'}(t)$.

Así, para concluir que $\mathcal{M}' \rhd P'$, sólo falta demostrar que $E^{P'} \subseteq R^{\mathcal{M}'} \subseteq A^{P'}$.

Como $R^{\mathcal{M}'} = (R^{\mathcal{M}} - (\{s\} \times R^{\mathcal{M}}[s])) \cup (\{s\} \times S')$, y $A^{P'} = A^P - (\{s\} \times (A^P[s] - S'))$, tenemos que $R^{\mathcal{M}'}[s] = S'$ y $A^{P'}[s] = S'$.

Por lo tanto:

1) si $t \neq s$, entonces
   $E^{P'}[t] = E^P[t] \subseteq R^{\mathcal{M}}[t] = R^{\mathcal{M}'}[t] \subseteq A^P[t] = A^{P'}[t]$,
   $\therefore E^{P'}[t] \subseteq R^{\mathcal{M}'}[t] \subseteq A^{P'}[t]$
2) si $t = s$, entonces
   $E^{P'}[t] = E^P[t] \subseteq S' = R^{\mathcal{M}'}[t] = A^{P'}[t] \subseteq A^{P'}[t]$
   $\therefore E^{P'}[t] \subseteq R^{\mathcal{M}'}[t] \subseteq A^{P'}[t]$

$\therefore E^{P'} \subseteq R^{\mathcal{M}'} \subseteq A^{P'}$.

Por lo tanto, $\mathcal{M}' \rhd P'$ y $(\mathcal{M}', P') \in \mathbf{KP}_\Sigma$.

b) Además, tenemos $E^{P'} = E^P \supseteq E^P$, y $\forall t \in \mathbb{S} : L^{P'}(t) = L^P(t) \supseteq L^P(t)$.

Así, para concluir que $P' \succeq P$, sólo falta demostrar que $A^{P'} \subseteq A^P$.

Como $A^{P'} = A^P - (\{s\} \times (A^P[s] - S'))$, tenemos que $A^{P'}[s] = S'$.

Por lo tanto:

1) si $t \neq s$, entonces $A^{P'}[t] = A^P[t] \subseteq A^P[t]$.
2) si $t = s$, entonces $A^{P'}[t] = S' \subseteq A^P[t]$.

$\therefore A^{P'} \subseteq A^P$.

Por lo tanto, $P' \succeq P$.

4. Si $E^P[s] \subseteq S' \subseteq A^P[s]$, $S' \neq \varnothing$, y $(\mathcal{M}', P') = \mathbf{T}^u_{\exists}((\mathcal{M}, P), s, S')$, entonces

   $(\mathcal{M}', P') = (\mathbf{T}^u(\mathcal{M}, s, S'), \langle E^P \cup (\{s\} \times (S' - E^P[s])), A^P, L^P \rangle)$.

   Por lo tanto:

   $\mathcal{M}' = \mathbf{T}^u(\mathcal{M}, s, S') = \langle S^{\mathcal{M}}, (R^{\mathcal{M}} - (\{s\} \times R^{\mathcal{M}}[s])) \cup (\{s\} \times S'), L^{\mathcal{M}} \rangle$, y

   $P' = \langle E^P \cup (\{s\} \times (S' - E^P[s])), A^P, L^P \rangle$.

   Consecuentemente, la única diferencia entre $\mathcal{M}'$ y $\mathcal{M}$ está en el segundo componente, y la única diferencia entre P' y P está en el primer componente.

   a) Ya que $\mathcal{M} \rhd P$, tenemos que $E^P \subseteq R^{\mathcal{M}} \subseteq A^P$, y $\forall t \in S^{\mathcal{M}} : L^{\mathcal{M}}(t) \supseteq L^P(t)$.

   Por lo tanto, $\forall t \in S^{\mathcal{M}'} = S^{\mathcal{M}},\ L^{\mathcal{M}'}(t) = L^{\mathcal{M}}(t) \supseteq L^P(t) = L^{P'}(t)$.

   Así, para concluir que $\mathcal{M}' \rhd P'$, sólo falta demostrar que $E^{P'} \subseteq R^{\mathcal{M}'} \subseteq A^{P'}$.

   Como $R^{\mathcal{M}'} = (R^{\mathcal{M}} - (\{s\} \times R^{\mathcal{M}}[s])) \cup (\{s\} \times S')$, y $E^{P'} = E^P \cup (\{s\} \times (S' - E^P[s]))$, tenemos que $R^{\mathcal{M}'}[s] = S'$ y $E^{P'}[s] = S'$.

   Por lo tanto:

   1) si $t \neq s$, entonces
      $E^{P'}[t] = E^P[t] \subseteq R^{\mathcal{M}}[t] = R^{\mathcal{M}'}[t] \subseteq A^P[t] = A^{P'}[t]$,
      $\therefore E^{P'}[t] \subseteq R^{\mathcal{M}'}[t] \subseteq A^{P'}[t]$

   2) si $t = s$, entonces
      $E^{P'}[t] = S' \subseteq S' = R^{\mathcal{M}'}[t] \subseteq A^P[t] = A^{P'}[t]$
      $\therefore E^{P'}[t] \subseteq R^{\mathcal{M}'}[t] \subseteq A^{P'}[t]$

   $\therefore E^{P'} \subseteq R^{\mathcal{M}'} \subseteq A^{P'}$.

   Por lo tanto, $\mathcal{M}' \rhd P'$ y $(\mathcal{M}', P') \in \mathbf{KP}_{\Sigma}$.

   b) Además, tenemos $A^{P'} = A^P \subseteq A^P$, y $\forall t \in \mathbb{S} : L^{P'}(t) = L^P(t) \supseteq L^P(t)$.

   Así, para concluir que $P' \succeq P$, sólo falta demostrar que $E^{P'} \supseteq E^P$.

   Como $E^{P'} = E^P \cup (\{s\} \times (S' - E^P[s]))$, tenemos que $E^{P'}[s] = S'$.

   Por lo tanto:

1) si $t \neq s$, entonces $E^{P'}[t] = E^P[t] \supseteq E^P[t]$.

2) si $t = s$, entonces $E^{P'}[t] = S' \supseteq E^P[t]$.

$\therefore E^{P'} \subseteq E^P$.

Por lo tanto, $P' \succeq P$.

□

Como consecuencia del Teorema 3.7.3, tenemos una de las características clave de la modificación de modelos protegidos: La protección de una modificación es mayor que o igual a la protección del modelo de entrada.

Teorema 3.7.4 (Las modificaciones no disminuyen las protecciones)**.** *Sean* $(\mathcal{M}, P) \in \mathbf{KP}_\Sigma, s \in S^\mathcal{M}, S' \subseteq S^\mathcal{M}$*, y sea* $\varphi \in \Sigma$*-XCTL.*

Si $(\mathcal{M}', P') \in \mathit{Modif}((\mathcal{M}, P), s, \varphi)$ *o* $(\mathcal{M}', P') \in \mathit{Modif}^*((\mathcal{M}, P), S', \varphi)$*,*

entonces $P' \succeq P$*.*

Demostración. Si $(\mathcal{M}', P') \in \mathit{Modif}((\mathcal{M}, P), s, \varphi)$ o $(\mathcal{M}', P') \in \mathit{Modif}^*((\mathcal{M}, P), S', \varphi)$, entonces o bien $(\mathcal{M}', P') = (\mathcal{M}, P)$, o $(\mathcal{M}', P')$ se obtiene a partir $(\mathcal{M}, P)$ mediante la aplicación de operaciones de actualización protegida. Por lo tanto, por el Teorema 3.7.3, $P' \succeq P$. □

Teorema 3.7.5 ($\succeq$ preserva satisfactibilidad)**.** *Sean* $(\mathcal{M}, P), (\mathcal{M}', P') \in \mathbf{KP}_\Sigma$*,* $s \in S^\mathcal{M}$*,* $y\ \varphi \in \Sigma$*-XCTL. Si* $P' \succeq P$ *y* $(\mathcal{M}, P), s \models \varphi$ *entonces* $(\mathcal{M}', P'), s \models \varphi$*.*

Demostración. Supongamos que $P' \succeq P$ y $(\mathcal{M}, P), s \models \varphi$.

Procedemos por inducción sobre la estructura de φ.

Base.

1. Los casos $\varphi = \mathsf{F}$ y $\varphi = \mathsf{T}$ son triviales.

2. Si $\varphi = \ell$, entonces $\ell \in L^P(s) \subseteq L^{P'}(s)$. Así, $(\mathcal{M}', P'), s \models \ell$.

3. Si $\varphi = \mathbf{OD}_{\leq} n$, entonces $A^{P'}[s] \subseteq A^{P}[s]$ y $|A^{P}[s]| \leq n$.

   Por lo tanto, $|A^{P'}[s]| \leq |A^{P}[s]| \leq n$.

   Así, $(\mathcal{M}', P'), s \models \mathbf{OD}_{\leq} n$.

4. Si $\varphi = \mathbf{OD}_{>} n$, entonces $E^{P'}[s] \supseteq E^{P}[s]$ y $|E^{P}[s]| > n$.

   Por lo tanto, $|E^{P'}[s]| \geq |E^{P}[s]| > n$.

   Así, $(\mathcal{M}', P'), s \models \mathbf{OD}_{>} n$.

Inducción. Supongamos que el teorema es cierto para $\varphi = \alpha, \beta$ (IH).

1. Si $\varphi = \alpha \vee \beta$, entonces, sin pérdida de generalidad $(\mathcal{M}, P), s \models \alpha$.

   Por lo tanto, por (IH), $(\mathcal{M}', P'), s \models \alpha$.

   Así, $(\mathcal{M}', P'), s \models \alpha \vee \beta$.

2. Si $\varphi = \alpha \wedge \beta$, entonces $(\mathcal{M}, P), s \models \alpha$ y $(\mathcal{M}, P), s \models \beta$.

   Por lo tanto, por (IH), $(\mathcal{M}', P'), s \models \alpha$ y $(\mathcal{M}', P'), s \models \beta$.

   $\therefore (\mathcal{M}', P'), s \models \alpha \wedge \beta$.

3. Si $\varphi = \mathbf{EX}\, \alpha$, entonces $\exists t \in E^{P}[s] \subseteq E^{P'}[s]$ tal que $(\mathcal{M}, P), t \models \alpha$.

   Por lo tanto, por (IH), $(\mathcal{M}', P'), t \models \alpha$. Así, $(\mathcal{M}', P'), s \models \mathbf{EX}\, \alpha$.

4. Si $\varphi = \mathbf{AX}\, \alpha$, entonces $\forall t \in A^{P}[s] \supseteq A^{P'}[s]$, $(\mathcal{M}, P), t \models \alpha$.

   Por lo tanto, por (IH), $\forall t \in A^{P'}[s]$, $(\mathcal{M}', P'), t \models \alpha$. Así, $(\mathcal{M}', P'), s \models \mathbf{AX}\, \alpha$.

□

A continuación, mostramos en tres pasos que *Modif* y *Modif** son correctos. Primero, dados $\varphi \in \Sigma$-XCTL, $(\mathcal{M}, P) \in \mathbf{KP}_{\Sigma}$, y $S' \subseteq S^{\mathcal{M}}$, mostramos que *Modif** es condicionalmente correcto: asumiendo que *Modif* es correcto para φ y $(\mathcal{M}, P)$, mostramos que *Modif** es correcto usando inducción sobre $|S'|$. Segundo, usando la corrección condicional de *Modif**, mostramos que *Modif* es correcto mediante inducción sobre φ. Finalmente, como corolario, obtenemos la corrección de *Modif**.

Teorema 3.7.6 (Corrección condicional de *Modif**)**.** *Sean* $(\mathcal{M}, P) \in \mathbf{KP}_\Sigma$ *y* $\varphi \in \Sigma$*-XCTL tales que*

$$\forall s \in S^{\mathcal{M}}, \text{ si } (\mathcal{M}', P') \in \mathit{Modif}((\mathcal{M}, P), s, \varphi) \text{ entonces } (\mathcal{M}', P'), s \models \varphi \tag{3-4}$$

Entonces $\forall S' \subseteq S^{\mathcal{M}}$*, si* $(\mathcal{M}', P') \in \mathit{Modif}^*((\mathcal{M}, P), S', \varphi)$ *entonces* $(\mathcal{M}', P'), S' \models \varphi$*.*

Demostración. Inducción sobre $|S'|$.

Base.

Si $|S'| = 0$, entonces $S' = \varnothing$. Por lo tanto, $(\mathcal{M}', P'), S' \models \varphi$.

Inducción.

Supongamos que $0 < m \in \mathbb{N}$ y asumamos que

$$|S'| < m \Rightarrow ((\mathcal{M}', P') \in \mathit{Modif}^*((\mathcal{M}, P), S', \varphi) \Rightarrow (\mathcal{M}', P'), S' \models \varphi) \tag{3-5}$$

A continuación demostramos que:

si $|S'| = m$ y $(\mathcal{M}', P') \in \mathit{Modif}^*((\mathcal{M}, P), S', \varphi)$,

entonces $(\mathcal{M}', P'), S' \models \varphi$.

Ya que $|S'| = m > 0$, $S' \neq \varnothing$.

Así, por la Definición 3.5.2, si $(\mathcal{M}', P') \in \mathit{Modif}^*((\mathcal{M}, P), S', \varphi)$, entonces $\exists t \in S^{\mathcal{M}}$ y $\exists (\mathcal{M}_t, P_t) \in \mathit{Modif}((\mathcal{M}, P), t, \varphi)$, tal que $(\mathcal{M}', P') \in \mathit{Modif}^*((\mathcal{M}_t, P_t), S' - \{t\}, \varphi)$.

Por la implicación (3-5), $(\mathcal{M}', P'), S' - \{t\} \models \varphi$.

Y, por la implicación (3-4), $(\mathcal{M}_t, P_t), t \models \varphi$.

Además, por el Teorema 3.7.4, $P' \succeq P_t$ y $P_t \succeq P$.

Así, $P' \succeq P$ y $(\mathcal{M}', P'), t \models \varphi$.

Consecuentemente, $(\mathcal{M}', P'), S' \models \varphi$. □

Teorema 3.7.7 (Corrección de *Modif*)**.** *Sean* $(\mathcal{M}, P) \in \mathbf{KP}_\Sigma$*, y* $s \in S^{\mathcal{M}}$*.*

Para toda $\varphi \in \Sigma$*-XCTL, si* $(\mathcal{M}', P') \in \mathit{Modif}((\mathcal{M}, P), s, \varphi)$ *entonces* $(\mathcal{M}', P'), s \models \varphi$*.*

Demostración. Inducción sobre la estructura de φ.

Sean $(\mathcal{M}, P) \in \mathbf{KP}_\Sigma$, $s \in S^\mathcal{M}$, y supongamos que $(\mathcal{M}', P') \in \mathit{Modif}((\mathcal{M}, P), s, \varphi)$.

Base.

1. Los casos $\varphi = \mathsf{F}$ y $\varphi = \mathsf{T}$ son triviales.

2. Si $\varphi = \ell$, tenemos dos casos: $\overline{\ell} \in L^P[s]$ y $\overline{\ell} \notin L^P[s]$.

   Si $\overline{\ell} \in L^P[s]$, entonces $\mathit{Modif}((\mathcal{M}, P), s, \ell) = \varnothing$ y no hay nada que demostrar en este caso.

   Si $\overline{\ell} \notin L^P[s]$, entonces $\mathit{Modif}((\mathcal{M}, P), s, \ell)$ usa $\mathbf{L}^u$ para generar exactamente un modelo, $(\mathcal{M}', P')$, añadiendo ℓ a las etiquetas protegidas de s.

   Así, $\ell \in L^{P'}(s)$, $(\mathcal{M}', P'), s \models \ell$, y $\forall s \in S^\mathcal{M}$, $Q(s, \ell)$.

3. Si $\varphi = \mathbf{OD}_{\leq} n$, entonces $\exists S' \subseteq A^P[s]$ tal que $E^P[s] \subseteq S' \neq \varnothing$, $|S'| \leq n$, y $(\mathcal{M}', P') = \mathbf{T}^u_\forall((\mathcal{M}, P), s, S')$.

   Por lo tanto, por la definición de $\mathbf{T}^u_\forall$, $A^{P'}[s] \subseteq S'$.

   Así, $|A^{P'}[s]| \leq |S'| \leq n$ y $(\mathcal{M}', P'), s \models \mathbf{OD}_{\leq} n$.

4. Si $\varphi = \mathbf{OD}_{>} n$, entonces $\exists S' \subseteq A^P[s]$ tal que $E^P[s] \subseteq S' \neq \varnothing$, $|S'| > n$, y $(\mathcal{M}', P') = \mathbf{T}^u_\exists((\mathcal{M}, P), s, S')$.

   Por lo tanto, por la definición de $\mathbf{T}^u_\exists$, $E^{P'}[s] \supseteq S'$.

   Así, $|E^{P'}[s]| > |S'| > n$ y $(\mathcal{M}', P'), s \models \mathbf{OD}_{>} n$.

Inducción. Supongamos que el teorema es cierto para $\varphi = \alpha, \beta$ (IH).

1. Si $\varphi = \alpha \vee \beta$, entonces, sin pérdida de generalidad $(\mathcal{M}', P') \in \mathit{Modif}((\mathcal{M}, P), s, \alpha)$.

   Por IH, $(\mathcal{M}', P'), s \models \alpha$.

   Por lo tanto, $(\mathcal{M}', P'), s \models \alpha \vee \beta$.

2. Si $\varphi = \alpha \wedge \beta$, entonces existe $(\mathcal{M}_\alpha, P_\alpha) \in \mathit{Modif}((\mathcal{M}, P), s, \alpha)$ tal que $(\mathcal{M}', P') \in \mathit{Modif}((\mathcal{M}_\alpha, P_\alpha), s, \beta)$.

Por IH, $(\mathcal{M}_\alpha, P_\alpha), s \models \alpha$ y $(\mathcal{M}', P'), s \models \beta$.

Puesto que $P' \succeq P_\alpha$, por el Teorema 3.7.5, $(\mathcal{M}', P'), s \models \alpha$.

Por lo tanto, $(\mathcal{M}', P'), s \models \alpha \wedge \beta$.

3. Si $\varphi = \mathbf{EX}\,\alpha$, entonces $\exists t \in A^P[s]$ tal que $(\mathcal{M}', P') \in \mathit{Modif}(\mathbf{T}^+_\exists((\mathcal{M}, P), s, t), t, \alpha)$.

   Por IH, $(\mathcal{M}', P'), t \models \alpha$.

   Además, si $(\mathcal{M}_t, P_t) = \mathbf{T}^+_\exists((\mathcal{M}, P), s, t)$, entonces $t \in E^{P_t}[s]$,

   y por el Teorema 3.7.4 $P' \succeq P_t$.

   Así, $t \in E^{P'}[s]$ y $(\mathcal{M}', P'), t \models \alpha$.

   Por lo tanto, $(\mathcal{M}', P'), s \models \mathbf{EX}\,\alpha$.

4. Si $\varphi = \mathbf{AX}\,\alpha$, entonces existe $S' \subseteq A^P[s]$ tal que $E^P[s] \subseteq S' \neq \varnothing$ y $(\mathcal{M}', P') \in \mathit{Modif}^*((\mathcal{M}_{S'}, P_{S'}), S', \alpha)$, donde $(\mathcal{M}_{S'}, P_{S'}) = \mathbf{T}^u_\forall((\mathcal{M}, P), s, S')$.

   Por IH, $\forall s \in S^{\mathcal{M}}$, si $(\mathcal{M}'', P'') \in \mathit{Modif}((\mathcal{M}_{S'}, P_{S'}), s, \alpha)$ entonces $(\mathcal{M}'', P''), s \models \alpha$.

   Por lo tanto, por el Teorema 3.7.6, $(\mathcal{M}', P'), S' \models \alpha$.

   Por la definición de $P_{S'}$, tenemos $S' = A^{P_{S'}}[s]$, y, por el Teorema 3.7.4, $P' \succeq P_{S'}$.

   Así, $S' = A^{P_{S'}}[s] \supseteq A^{P'}[s]$.

   Ya que $(\mathcal{M}', P'), S' \models \alpha$ y $A^{P'}[s] \subseteq S'$, $(\mathcal{M}', P'), A^{P'}[s] \models \alpha$.

   Por lo tanto, $(\mathcal{M}', P'), s \models \mathbf{AX}\,\alpha$.

□

Corolario 3.7.8 (Corrección de *Modif**)**.** *Sean $\varphi \in \Sigma$-XCTL y $(\mathcal{M}, P) \in \mathbf{KP}_\Sigma$.*

Para todo $S' \subseteq S^{\mathcal{M}}$, si $(\mathcal{M}', P') \in \mathit{Modif}^*((\mathcal{M}, P), S', \varphi)$ entonces $(\mathcal{M}', P'), S' \models \varphi$.

Demostración. Sean $\varphi \in \Sigma$-XCTL, $(\mathcal{M}, P) \in \mathbf{KP}_\Sigma$, $S' \subseteq S^{\mathcal{M}}$.

Supongamos que $(\mathcal{M}', P') \in \mathit{Modif}^*((\mathcal{M}, P), S', \varphi)$.

Por el Teorema 3.7.7, $\forall s \in S^{\mathcal{M}}$, $(\mathcal{M}'', P'') \in \mathit{Modif}((\mathcal{M}, P), s, \varphi) \Rightarrow (\mathcal{M}'', P''), s \models \varphi$.

Por lo tanto, por el Teorema 3.7.6, $(\mathcal{M}', P'), S' \models \varphi$. □

Para demostrar la completitud de *Modif*, es importante saber si existe una modificación de $(\mathcal{M}, P)$ para φ, es decir, saber si $(\mathcal{M}, P)$ es "φ-modificable".

Definición 3.7.9 (φ-modificable)**.** Sean $(\mathcal{M}, P) \in \mathbf{KP}_\Sigma$, $s \in S^{\mathcal{M}}$, y $\varphi \in \Sigma$-CTL. Decimos que $(\mathcal{M}, P)$ es *φ-modificable* en s si $\mathit{Modif}((\mathcal{M}, P), s, \varphi) \neq \varnothing$.

Además, tenemos que demostrar que si φ es "satisfactible", entonces XUPD_{prot} produce al menos un resultado. Por lo tanto, tenemos que aclarar el concepto de "satisfactible".

Definición 3.7.10 (Satisfactible)**.** Sean $\varphi \in \Sigma$-CTL, $P \in \mathbf{P}_\Sigma$, y $s \in \mathbb{S}$. Decimos que:

1. φ es *P-satisfactible en s* si $\exists\, (\mathcal{M}', P') \in \mathbf{KP}_\Sigma$ tal que $(\mathcal{M}', P'), s \models \varphi$ y $P' \succeq P$.

2. φ es *$\mathbf{K}_\Sigma$-satisfactible en s* si $\exists\, \mathcal{M}' \in \mathbf{K}_\Sigma$ tal que $\mathcal{M}', s \models \varphi$.

Para afirmar la completitud de *Modif* no es suficiente invertir la implicación del Teorema 3.7.7. A continuación, analizamos dos enunciados incorrectos de la completitud de *Modif*.

Primero, el enunciado

(1) $(\mathcal{M}', P'), s \models \varphi \Rightarrow (\mathcal{M}', P') \in \mathit{Modif}((\mathcal{M}, P), s, \varphi)$

no es cierto.

Por ejemplo, si $\overline{\ell}_1 \in L^P(s)$ y $\ell_1 \in L^{P'}(s)$, entonces $(\mathcal{M}', P') \models \ell_1$ pero $(\mathcal{M}', P') \notin \mathit{Modif}((\mathcal{M}, P), s, \ell_1) = \varnothing$.

Segundo, un enunciado más débil,

(2) $(\mathcal{M}', P'), s \models \varphi$ y $P' \succeq P \Rightarrow (\mathcal{M}', P') \in \mathit{Modif}((\mathcal{M}, P), s, \varphi)$,

tampoco es cierto.

Por ejemplo, si $\overline{\ell}_1, \overline{\ell}_2 \notin L^P(s)$, y $(\mathcal{M}', P')$ es el resultado de añadir ℓ_1 y ℓ_2 a $(\mathcal{M}, P)$, es decir $(\mathcal{M}_{\ell_1}, P_{\ell_1}) = \mathbf{L}^u((\mathcal{M}, P), s, \ell_1)$, y $(\mathcal{M}', P') = \mathbf{L}^u((\mathcal{M}_{\ell_1}, P_{\ell_1}), s, \ell_2)$, entonces $(\mathcal{M}', P') \models \ell_1$ y $P' \succeq P$ pero $(\mathcal{M}', P') \notin \mathit{Modif}((\mathcal{M}, P), s, \ell_1) = \{(\mathcal{M}_{\ell_1}, P_{\ell_1})\}$.

De acuerdo con nuestros propósitos (por ejemplo, el Corolario 3.7.15), un enunciado más apropiado para la completitud de *Modif* es el enunciado del Teorema 3.7.12. Para demostrar este teorema, seguimos tres pasos.

Primero, mostramos que *Modif** es condicionalmente completo. Es decir, asumiendo que *Modif* es completo para φ y $(\mathcal{M}, P)$ dados, demostramos que *Modif** es completo usando inducción sobre $|S'|$.

Segundo, usando la completitud condicional de *Modif**, demostramos que *Modif* es completo usando inducción sobre φ.

Finalmente, como corolario, obtenemos la completitud de *Modif**.

Teorema 3.7.11 (Completitud condicional de *Modif**)**.** *Sean* $(\mathcal{M}, P) \in \mathbf{KP}_\Sigma$ *y* $\varphi \in \Sigma$*-XCTL tales que*

$$\begin{aligned} \forall s \in S^{\mathcal{M}},\ (\exists (\mathcal{M}', P') \in \mathbf{KP}_\Sigma.(\mathcal{M}', P'), s \models \varphi \wedge P' \succeq P) \Rightarrow \\ (\exists (\mathcal{M}_{s,\varphi}, P_{s,\varphi}) \in \mathit{Modif}((\mathcal{M}, P), s, \varphi).\ P' \succeq P_{s,\varphi}) \end{aligned} \tag{3-6}$$

Entonces $\forall S' \subseteq S^{\mathcal{M}}$*, si* $\exists (\mathcal{M}', P') \in \mathbf{KP}_\Sigma$ *tal que* $(\mathcal{M}', P'), S' \models \varphi$ *y* $P' \succeq P$*, entonces* $\exists (\mathcal{M}_{S',\varphi}, P_{S',\varphi}) \in \mathit{Modif}^*((\mathcal{M}, P), S', \varphi)$ *tal que* $P' \succeq P_{S',\varphi}$.

Demostración. Inducción sobre $|S'|$.

Base.

Si $|S'| = 0$, entonces $S' = \varnothing$ y $\mathit{Modif}^*((\mathcal{M}, P), \varnothing, \varphi) = \{(\mathcal{M}, P)\}$.

Por lo tanto, si tomamos $(\mathcal{M}_{S',\varphi}, P_{S',\varphi}) = (\mathcal{M}, P) \in \mathit{Modif}^*((\mathcal{M}, P), \varnothing, \varphi)$, entonces $P' \succeq P = P_{S',\varphi}$.

Inducción.

Supongamos que $0 < m \in \mathbb{N}$ y asumamos que

$$\begin{aligned} (S' \subseteq S^{\mathcal{M}} \wedge |S'| < m) \Rightarrow \\ ((\exists (\mathcal{M}', P') \in \mathbf{KP}_\Sigma.(\mathcal{M}', P'), S' \models \varphi \wedge P' \succeq P) \Rightarrow \\ (\exists (\mathcal{M}_{S',\varphi}, P_{S',\varphi}) \in \mathit{Modif}^*((\mathcal{M}, P), S', \varphi).\ P' \succeq P_{S',\varphi})) \end{aligned} \tag{3-7}$$

A continuación demostramos que:

si $S' \subseteq S^{\mathcal{M}}$, $|S'| = m$, y $\exists (\mathcal{M}', P') \in \mathbf{KP}_\Sigma$ tal que $(\mathcal{M}', P'), S' \models \varphi$ y $P' \succeq P$,

entonces $\exists (\mathcal{M}_{S',\varphi}, P_{S',\varphi}) \in \mathit{Modif}^*((\mathcal{M}, P), S', \varphi)$ tal que $P' \succeq P_{S',\varphi}$.

Sea $S' \subseteq S^{\mathcal{M}}$ tal que $|S'| = m > 0$.

Supongamos que $(\mathcal{M}', P') \in \mathbf{KP}_\Sigma$ es tal que

$$(\mathcal{M}', P'), S' \models \varphi \tag{3-8}$$

$$\text{y } P' \succeq P \tag{3-9}$$

Ya que $|S'| > 0$, $S' \neq \varnothing$.

Sean $t \in S'$ y $S'_t = S' - \{t\}$.

Por un lado, por (3-8) y (3-9), $(\mathcal{M}', P'), t \models \varphi$ y $P' \succeq P$.

Así, por (3-6), $\exists(\mathcal{M}_{t,\varphi}, P_{t,\varphi}) \in \mathit{Modif}((\mathcal{M}, P), s, \varphi)$ tal que $P' \succeq P_{t,\varphi}$.

Por otra parte, por (3-8), $(\mathcal{M}', P'), S'_t \models \varphi$ y $P' \succeq P_{t,\varphi}$.

Así, por (3-7), $\exists(\mathcal{M}_{S'_t,\varphi}, P_{S'_t,\varphi}) \in \mathit{Modif}^*((\mathcal{M}_{t,\varphi}, P_{t,\varphi}), S'_t, \varphi)$ tal que $P' \succeq P_{S'_t,\varphi}$.

Consecuentemente, usando $S'_t = S' - \{t\}$ y la Definición 3.5.2, tenemos que

$(\mathcal{M}_{S'_t,\varphi}, P_{S'_t,\varphi}) \in \mathit{Modif}^*((\mathcal{M}, P), S', \varphi)$ y $P' \succeq P_{S',\varphi}$. □

Teorema 3.7.12 (Completitud de *Modif*)**.** *Sean* $(\mathcal{M}, P) \in \mathbf{KP}_\Sigma$*, y* $s \in S^{\mathcal{M}}$*. Para toda* $\varphi \in \Sigma$*-XCTL,*

si $\exists(\mathcal{M}', P') \in \mathbf{KP}_\Sigma$ *tal que* $(\mathcal{M}', P'), s \models \varphi$ *y* $P' \succeq P$

entonces $\exists(\mathcal{M}_{s,\varphi}, P_{s,\varphi}) \in \mathit{Modif}((\mathcal{M}, P), s, \varphi)$ *tal que* $P' \succeq P_{s,\varphi}$.

Demostración. Inducción sobre la estructura de φ.

Sean $(\mathcal{M}, P) \in \mathbf{KP}_\Sigma$, $s \in S^{\mathcal{M}}$, y supongamos que $(\mathcal{M}, P) \in \mathbf{KP}_\Sigma$ es tal que

$$(\mathcal{M}', P'), s \models \varphi \tag{3-10}$$

$$\text{y } P' \succeq P \tag{3-11}$$

Base.

1. Si $\varphi = \mathsf{F}$, entonces $\nexists(\mathcal{M}', P') \in \mathbf{KP}_\Sigma$ tal que $(\mathcal{M}', P'), s \models \mathsf{F}$ y $P' \succeq P$. Por lo tanto, no hay nada que demostrar en este caso.

2. Si $\varphi = \mathsf{T}$, tomemos $(\mathcal{M}_{s,\mathsf{T}}, P_{s,\mathsf{T}}) = (\mathcal{M}, P) \in \{(\mathcal{M}, P)\} = \mathit{Modif}((\mathcal{M}, P), s, \mathsf{T})$.

   Así, $P' \succeq P = P_{s,\mathsf{T}}$.

3. Si $\varphi = \ell$, entonces, por (3-10), $\ell \in L^{P'}(s)$.

   Por lo tanto, $\overline{\ell} \notin L^{P'}(s)$ y, por (3-11), $\overline{\ell} \notin L^{P}(s)$.

   Sea $(\mathcal{M}_{s,\ell}, P_{s,\ell}) = \mathbf{L}^u((\mathcal{M}, P), s, \ell)$, el único elemento de $\mathit{Modif}((\mathcal{M}, P), s, \ell)$.

   Por el Teorema 3.7.3, $(\mathcal{M}_{s,\ell}, P_{s,\ell}) \in \mathbf{KP}_\Sigma$ (y $P_{s,\ell} \succeq P$).

   Además, $P' \succeq P$, $E^{P_{s,\ell}} = E^P$, $A^{P_{s,\ell}} = A^P$, y $L^{P_{s,\ell}} = L^P[s \oplus \ell]$.

   Por lo tanto, $P' \succeq P_{s,\ell}$.

   Consecuentemente, $(\mathcal{M}_{s,\ell}, P_{s,\ell}) \in \mathit{Modif}((\mathcal{M}, P), s, \ell)$ y $P' \succeq P_{s,\ell}$.

4. $\varphi = \mathbf{OD}_{\leq} n$.

   Como $(\mathcal{M}', P'), s \models \mathbf{OD}_{\leq} n$ (por 3-10), tenemos que $|A^{P'}[s]| \leq n$.

   Sea $S' = A^{P'}[s]$.

   Como $P' \succeq P$ (por 3-11), $E^P[s] \subseteq E^{P'}[s] \subseteq R^M[s] \subseteq S' = A^{P'}[s] \subseteq A^P[s]$.

   $\therefore E^P[s] \subseteq S' \subseteq A^P[s]$ y $S' \neq \varnothing$, porque $S' \supseteq R^M[s] \neq \varnothing$.

   $\therefore \mathbf{T}^u_\forall((\mathcal{M}, P), s, S') = \big(\mathbf{T}^u(\mathcal{M}, s, S'), \big\langle E^P, A^P - (\{s\} \times (A^P[s] - S')), L^P \big\rangle\big)$.

   Sea $(\mathcal{M}_{s,\varphi}, P_{s,\varphi}) = \mathbf{T}^u_\forall((\mathcal{M}, P), s, S')$.

   Tenemos que

   $S' = A^{P'}[s] \subseteq A^P[s]$, $E^P[s] \subseteq E^{P'}[s] \subseteq A^{P'}[s] = S' \neq \varnothing$, y $|S'| = |A^{P'}[s]| \leq n$.

   Por lo tanto, $(\mathcal{M}_{s,\varphi}, P_{s,\varphi}) \in \mathit{Modif}((\mathcal{M}, P), s, \mathbf{OD}_{\leq} n)$.

   Además, $P_{s,\varphi} = \big\langle E^P, A^P - (\{s\} \times (A^P[s] - S')), L^P \big\rangle$.

   Ya que $P' \succeq P$, tenemos que:

   a) $E^{P'} \supseteq E^P = E^{P_{s,\varphi}}$,

b) $A^{P_{s,\varphi}} = A^P - (\{s\} \times (A^P[s] - S'))$,

$\therefore A^{P_{s,\varphi}}[s] = S'$, y $\forall t \in \mathbb{S}, t \neq s \Rightarrow A^{P_{s,\varphi}}[t] = A^P[t]$

$\therefore \forall t \in \mathbb{S}, t \neq s \Rightarrow A^{P'}[t] \subseteq A^P[t] = A^{P_{s,\varphi}}[t]$, y $A^{P'}[s] = S' = A^{P_{s,\varphi}}[s]$,

$\therefore A^{P'} \subseteq A^{P_{s,\varphi}}$

c) $\forall t \in \mathbb{S}, L^{P'}(t) \supseteq L^P(t) = L^{P_{s,\varphi}}(t)$

$\therefore P' \succeq P_{s,\varphi}$.

Por lo tanto, $(\mathcal{M}_{s,\varphi}, P_{s,\varphi}) \in \mathit{Modif}((\mathcal{M}, P), s, \mathbf{OD}_{\leq} n)$ y $P' \succeq P_{s,\varphi}$.

5. $\varphi = \mathbf{OD}_{>} n$. Este caso es análogo al caso $\varphi = \mathbf{OD}_{\leq} n$:

Como $(\mathcal{M}', P'), s \models \mathbf{OD}_{>} n$ (por 3-10), tenemos que $|E^{P'}[s]| > n \geq 0$, $\therefore E^{P'}[s] \neq \varnothing$.

Sea $S' = E^{P'}[s]$.

Como $P' \succeq P$ (por 3-11), $E^P[s] \subseteq E^{P'}[s] = S' \subseteq R^M[s] \subseteq A^{P'}[s] \subseteq A^P[s]$.

$\therefore E^P[s] \subseteq S' \subseteq A^P[s]$ y $S' \neq \varnothing$, porque $S' = E^{P'}[s] \neq \varnothing$.

$\therefore \mathbf{T}^u_{\exists}((\mathcal{M}, P), s, S') = \left(\mathbf{T}^u(\mathcal{M}, s, S'), \left\langle E^P \cup (\{s\} \times (S' - E^P[s])), A^P, L^P \right\rangle\right)$

Sea $(\mathcal{M}_{s,\varphi}, P_{s,\varphi}) = \mathbf{T}^u_{\exists}((\mathcal{M}, P), s, S')$.

Tenemos que

$S' = E^{P'}[s] \subseteq A^{P'}[s] \subseteq A^P[s]$, $\varnothing \neq S' = E^{P'}[s] \supseteq E^P[s]$, y $|S'| = |E^{P'}[s]| > n$.

Por lo tanto, $(\mathcal{M}_{s,\varphi}, P_{s,\varphi}) \in \mathit{Modif}((\mathcal{M}, P), s, \mathbf{OD}_{>} n)$.

Además, $P_{s,\varphi} = \left\langle E^P \cup (\{s\} \times (S' - E^P[s])), A^P, L^P \right\rangle$.

Ya que $P' \succeq P$, tenemos que:

a) $E^{P_{s,\varphi}} = E^P \cup (\{s\} \times (S' - E^P[s]))$,

$\therefore E^{P_{s,\varphi}}[s] = S'$, y $\forall t \in \mathbb{S}, t \neq s \Rightarrow E^{P_{s,\varphi}}[t] = E^P[t]$

$\therefore \forall t \in \mathbb{S}, t \neq s \Rightarrow E^{P'}[t] \supseteq E^P[t] = E^{P_{s,\varphi}}[t]$, y $E^{P'}[s] = S' = E^{P_{s,\varphi}}[s]$,

$\therefore E^{P'} \supseteq E^{P_{s,\varphi}}$

b) $A^{P'} \subseteq A^P = A^{P_{s,\varphi}}$,

c) $\forall t \in \mathbb{S}, L^{P'}(t) \supseteq L^P(t) = L^{P_{s,\varphi}}(t)$

$\therefore P' \succeq P_{s,\varphi}$.

Por lo tanto, $(\mathcal{M}_{s,\varphi}, P_{s,\varphi}) \in \mathit{Modif}((\mathcal{M}, P), s, \mathbf{OD}_{>}\, n)$ y $P' \succeq P_{s,\varphi}$.

Inducción.

Sea IH la hipótesis de inducción, es decir IH afirma que el teorema es cierto para $\varphi = \alpha$ y para $\varphi = \beta$.

1. $\varphi = \alpha \vee \beta$.

   Por (3-10), y sin pérdida de generalidad $(\mathcal{M}', P'), s \models \alpha$.

   Así, por IH, $\exists(\mathcal{M}_{s,\alpha}, P_{s,\alpha}) \in \mathit{Modif}((\mathcal{M}, P), s, \alpha)$ tal que $P' \succeq P_{s,\alpha}$.

   Consecuentemente, $(\mathcal{M}_{s,\alpha}, P_{s,\alpha}) \in \mathit{Modif}((\mathcal{M}, P), s, \alpha \vee \beta)$ y $P' \succeq P_{s,\alpha}$.

2. $\varphi = \alpha \wedge \beta$.

   Por (3-10), $(\mathcal{M}', P'), s \models \alpha$.

   Así, por IH, $\exists(\mathcal{M}_{s,\alpha}, P_{s,\alpha}) \in \mathit{Modif}((\mathcal{M}, P), s, \alpha)$ tal que $P' \succeq P_{s,\alpha}$.

   Por lo tanto, $P' \succeq P_{s,\alpha}$ y, por (3-10), $(\mathcal{M}', P'), s \models \beta$.

   Así, aplicando IH con $(\mathcal{M}, P) = (\mathcal{M}_{s,\alpha}, P_{s,\alpha})$ y $\varphi = \beta$, tenemos que

   $\exists(\mathcal{M}_{s,\beta}, P_{s,\beta}) \in \mathit{Modif}((\mathcal{M}_{s,\alpha}, P_{s,\alpha}), s, \beta)$ tal que $P' \succeq P_{s,\beta}$.

   Consecuentemente, $(\mathcal{M}_{s,\beta}, P_{s,\beta}) \in \mathit{Modif}((\mathcal{M}, P), s, \alpha \wedge \beta)$ y $P' \succeq P_{s,\beta}$.

3. $\varphi = \mathbf{EX}\, \alpha$.

   Por (3-10), $\exists s' \in E^{P'}[s]$ tal que $(\mathcal{M}', P'), s' \models \alpha$.

   Sea $(\mathcal{M}_{s'}, P_{s'}) = \mathbf{T}^{+}_{\exists}((\mathcal{M}, P), s, s')$.

   Ya que $s' \in E^{P'}[s]$, por (3-11), $P' \succeq P_{s'}$.

   Por lo tanto, aplicando IH con $(\mathcal{M}, P) = (\mathcal{M}_{s'}, P_{s'})$ y $\varphi = \alpha$, tenemos que

$\exists(\mathcal{M}_{s',\alpha}, P_{s',\alpha}) \in \mathit{Modif}((\mathcal{M}_{s'}, P_{s'}), s', \alpha)$ tal que $P' \succeq P_{s',\alpha}$.

Así, $s' \in E^{P'}[s] \subseteq A^{P'}[s]$ y $(\mathcal{M}_{s',\alpha}, P_{s',\alpha}) \in \mathit{Modif}(\mathbf{T}^{+}_{\exists}((\mathcal{M}, P), s, s'), s', \alpha)$.

Por lo tanto, $(\mathcal{M}_{s',\alpha}, P_{s',\alpha}) \in \mathit{Modif}((\mathcal{M}, P), s, \mathbf{EX}\,\alpha)$ y $P' \succeq P_{s',\alpha}$.

4. $\varphi = \mathbf{AX}\,\alpha$.

   Por (3-10), $\forall s' \in A^{P'}[s]$, $(\mathcal{M}', P'), s' \models \alpha$.

   Sea $S' = A^{P'}[s]$ y $(\mathcal{M}_{S'}, P_{S'}) = \mathbf{T}^{u}_{\forall}((\mathcal{M}, P), s, S')$.

   Ya que $S' \subseteq A^{P'}[s]$, por (3-11), $P' \succeq P_{S'}$.

   Así, aplicando IH con $(\mathcal{M}, P) = (\mathcal{M}_{S'}, P_{S'})$ y $\varphi = \alpha$, tenemos que

   $\exists(\mathcal{M}_{S',\alpha}, P_{S',\alpha}) \in \mathit{Modif}^*((\mathcal{M}_{S'}, P_{S'}), S', \alpha)$ tal que $P' \succeq P_{S',\alpha}$.

   Puesto que $S' = A^{P'}[s] \supseteq E^{P'}[s]$ y $A^{P'}[s] \supseteq R^{\mathcal{M}'}[s] \neq \varnothing$, tenemos que

   $S' \subseteq A^{P'}[s]$ y $E^{P'}[s] \subseteq S' \neq \varnothing$.

   Así, $S' \subseteq A^{P'}[s]$, $E^{P'}[s] \subseteq S' \neq \varnothing$,

   y $(\mathcal{M}_{S',\alpha}, P_{S',\alpha}) \in \mathit{Modif}^*(\mathbf{T}^{u}_{\forall}((\mathcal{M}, P), s, S'), S', \alpha)$.

   Por lo tanto, $(\mathcal{M}_{S',\alpha}, P_{S',\alpha}) \in \mathit{Modif}((\mathcal{M}, P), s, \mathbf{AX}\,\alpha)$ y $P' \succeq P_{S',\alpha}$.

□

Corolario 3.7.13 (Completitud de *Modif**)**.** *Sean $\varphi \in \Sigma$-XCTL y $(\mathcal{M}, P) \in \mathbf{KP}_{\Sigma}$. Para todo $S' \subseteq S^{\mathcal{M}}$,*

si $\exists(\mathcal{M}', P') \in \mathbf{KP}_{\Sigma}$ tal que $(\mathcal{M}', P'), S' \models \varphi$ y $P' \succeq P$,

entonces $\exists(\mathcal{M}_{S',\varphi}, P_{S',\varphi}) \in \mathit{Modif}^*((\mathcal{M}, P), S', \varphi)$ tal que $P' \succeq P_{S',\varphi}$.

Demostración. Sean $\varphi \in \Sigma$-XCTL, y $(\mathcal{M}, P) \in \mathbf{KP}_{\Sigma}$, $S' \subseteq S^{\mathcal{M}}$.

Supongamos que $(\mathcal{M}', P') \in \mathbf{KP}_{\Sigma}$ es tal que

$$(\mathcal{M}', P'), S' \models \varphi \tag{3-12}$$

$$\text{y } P' \succeq P. \tag{3-13}$$

Por el Teorema 3.7.12, $\forall s \in S^{\mathcal{M}}$,

si $\exists(\mathcal{M}', P') \in \mathbf{KP}_\Sigma$ tal que $(\mathcal{M}', P'), s \models \varphi$ y $P' \succeq P$,

entonces $\exists(\mathcal{M}_{s,\varphi}, P_{s,\varphi}) \in \mathit{Modif}((\mathcal{M}, P), s, \varphi)$ tal que $P' \succeq P_{s,\varphi}$.

Por lo tanto, por el Teorema 3.7.11,

si $\exists(\mathcal{M}', P') \in \mathbf{KP}_\Sigma$ tal que $(\mathcal{M}', P'), S' \models \varphi$ y $P' \succeq P$,

entonces $\exists(\mathcal{M}_{S',\varphi}, P_{S',\varphi}) \in \mathit{Modif}^*((\mathcal{M}, P), S', \varphi)$ tal que $P' \succeq P_{S',\varphi}$.

Por lo tanto, por (3-12) y (3-13),

$\exists(\mathcal{M}_{S',\varphi}, P_{S',\varphi}) \in \mathit{Modif}^*((\mathcal{M}, P), S', \varphi)$ tal que $P' \succeq P_{S',\varphi}$. □

Por lo tanto, como se indica en el siguiente teorema, "φ es P-satisfactible" es equivalente a "$(\mathcal{M}, P)$ es φ-modificable".

Teorema 3.7.14 (φ-modificable si y sólo si P-satisfactible)**.** *Sean* $(\mathcal{M}, P) \in \mathbf{KP}_\Sigma$ *y* $s \in S^{\mathcal{M}}$*. Si* $\varphi \in \Sigma$*-XCTL, entonces:*

$(\mathcal{M}, P)$ *es* φ*-modificable en* s *si y sólo si* φ *es* P*-satisfactible en* s*.*

Demostración. Sean $(\mathcal{M}, P) \in \mathbf{KP}_\Sigma$ y $s \in S^{\mathcal{M}}$, y supongamos que $\varphi \in \Sigma$-XCTL.

($\Rightarrow$). Si $(\mathcal{M}, P)$ es φ-modificable en s, entonces $\mathit{Modif}((\mathcal{M}, P), s, \varphi) \neq \varnothing$.

Sea $(\mathcal{M}', P') \in \mathit{Modif}((\mathcal{M}, P), s, \varphi)$.

Por el Teorema 3.7.7, $(\mathcal{M}', P'), s \models \varphi$, y, por el Teorema 3.7.4, $P' \succeq P$.

Por lo tanto, φ es P-satisfactible en s.

($\Leftarrow$). Si φ es P-satisfactible en s,

entonces existe $(\mathcal{M}', P') \in \mathbf{KP}_\Sigma$ tal que $(\mathcal{M}', P'), s \models \varphi$ y $P' \succeq P$.

Por lo tanto, por el Teorema 3.7.12, existe $(\mathcal{M}_\varphi, P_\varphi) \in \mathit{Modif}((\mathcal{M}, P), s, \varphi)$ tal que $P' \succeq P_\varphi$.

Así, $\mathit{Modif}((\mathcal{M}, P), s, \varphi) \neq \varnothing$, y $(\mathcal{M}, P)$ es φ-modificable en s. □

La completitud de XUPD_{prot} es consecuencia del siguiente Corolario, el cual establece que un modelo sin protección es modificable respecto a φ si, y solamente si, φ es $\mathbf{K}_\Sigma$-satisfactible.

Corolario 3.7.15 (φ-modificable si y sólo si $\mathbf{K}_\Sigma$-satisfactible)**.** *Sea $s \in S^\Sigma$. Para toda $\varphi \in \Sigma$-XCTL, y para todo $\mathcal{M} \in \mathbf{K}_\Sigma$:*

$(\mathcal{M}, P_\perp)$ es φ-modificable en s si y sólo si φ es $\mathbf{K}_\Sigma$-satisfactible en s.

Demostración. Sean $s \in S^\Sigma$, $\varphi \in \Sigma$-XCTL, y $\mathcal{M} \in \mathbf{K}_\Sigma$.

($\Rightarrow$). Si $(\mathcal{M}', P') \in \mathit{Modif}((\mathcal{M}, P_\perp), s, \varphi)$,

entonces, por el Teorema 3.7.7, $(\mathcal{M}', P'), s \models \varphi$.

Además, $P_{\mathcal{M}'} \succeq P'$.

Consecuentemente, $(\mathcal{M}', P_{\mathcal{M}'}), s \models \varphi$ y, por la Definición 3.2.6, $\mathcal{M}', s \models \varphi$.

Por lo tanto, φ es $\mathbf{K}_\Sigma$-satisfactible en s.

($\Leftarrow$). Sea $\mathcal{M}' \in \mathbf{K}_\Sigma$ tal que $\mathcal{M}', s \models \varphi$.

Entonces, por la Definición 3.2.6, $(\mathcal{M}', P_{\mathcal{M}'}), s \models \varphi$.

Además, $P_{\mathcal{M}'} \succeq P_\perp$.

Por lo tanto, φ es $P_\perp$-satisfactible en s.

Así, por el Teorema 3.7.14, $(\mathcal{M}, P_\perp)$ es φ-modificable en s. □

Observe que XUPD_{prot} es intuitivamente una *implementación* de *Modif*, y que dicha implementación se describe con un pseudocódigo *informal*. Debido a esto, sin una semántica formal del pseudocódigo, no es posible demostrar formalmente que XUPD_{prot} es correcto y completo. Sin embargo, si asumimos que los modelos computados por XUPD_{prot} son exactamente las modificaciones del modelo de entrada, es decir, si para todo $(\mathcal{M}, P) \in \mathbf{KP}_\Sigma$, $s \in S^\mathcal{M}$, y $\varphi \in \Sigma$-XCTL, $\text{XUPD}_{prot}[(\mathcal{M}, P), s, \varphi] = \mathit{Modif}((\mathcal{M}, P), s, \varphi)$, entonces el siguiente teorema formaliza, en cierto sentido, que XUPD_{prot} es correcto y completo.

Teorema 3.7.16 (XUPD_{prot} es correcto y completo)**.** *Si para todo $(\mathcal{M}, P) \in \mathbf{KP}_\Sigma$, $s \in S^\mathcal{M}$, y $\varphi \in \Sigma$-XCTL, $\text{XUPD}_{prot}[(\mathcal{M}, P), s, \varphi] = \mathit{Modif}((\mathcal{M}, P), s, \varphi)$, entonces para todo $\mathcal{M} \in \mathbf{K}_\Sigma$, $s \in S^\mathcal{M}$, y $\varphi \in \Sigma$-XCTL:*

1. *Si $(\mathcal{M}', P') \in \text{XUPD}_{prot}[(\mathcal{M}, P_\perp), s, \varphi]$, entonces $(\mathcal{M}', P'), s \models \varphi$.*

2. *Si φ es $\mathbf{K}_\Sigma$-satisfactible en s, entonces $\text{XUPD}_{prot}[(\mathcal{M}, P_\perp), s, \varphi] \neq \varnothing$.*

Demostración. Sean $\mathcal{M} \in \mathbf{K}_\Sigma$, $s \in S^{\mathcal{M}}$, y $\varphi \in \Sigma$-XCTL.

1. Si $(\mathcal{M}', P') \in \text{XUPD}_{prot}[(\mathcal{M}, P_\perp), s, \varphi] = \mathit{Modif}((\mathcal{M}, P), s, \varphi)$, entonces, por el Teorema 3.7.7, $(\mathcal{M}', P'), s \models \varphi$.

   Por lo tanto, XUPD_{prot} es correcto.

2. Si φ es $\mathbf{K}_\Sigma$-satisfactible en s, entonces, por el Corolario 3.7.15, $(\mathcal{M}, P_\perp)$ es φ-modificable en s.

   Por lo tanto, $\text{XUPD}_{prot}[(\mathcal{M}, P_\perp), s, \varphi] = \mathit{Modif}((\mathcal{M}, P_\perp), s, \varphi) \neq \varnothing$.

□

3.8. Complejidad

En esta sección se proporciona un análisis de la complejidad de nuestro método para actualización de modelos. Medimos la complejidad de UPD_{prot} con respecto a dos parámetros: el tamaño de la fórmula de entrada y el tamaño del modelo de entrada.

Para fórmulas de la Σ-CTL, utilizamos el tamaño que se define a continuación.

Definición 3.8.1 (Tamaño de fórmulas Σ-CTL)**.** Si $\varphi \in \Sigma$-CTL, el tamaño de φ, $|\varphi|$, se define recursivamente sobre la estructura de φ:

1. Si $\varphi = \mathsf{F}, \mathsf{T}$, entonces

   $|\varphi| = 0$

2. Si $\varphi = \ell, \mathbf{OD}_{\leq} n, \mathbf{OD}_{>} n$, entonces

   $|\varphi| = 1$

3. Si $\varphi = \mathbf{EX}\,\alpha, \mathbf{AX}\,\alpha$, entonces

   $|\varphi| = |\alpha| + 1$

4. Si $\varphi = \alpha \vee \beta, \alpha \wedge \beta, \mathbf{E}[\alpha\, \mathbf{U}\, \beta], \mathbf{A}[\alpha\, \mathbf{R}\, \beta], \mathbf{A}[\alpha\, \mathbf{U}\, \beta], \mathbf{E}[\alpha\, \mathbf{R}\, \beta]$, entonces

   $|\varphi| = |\alpha| + |\beta| + 1$

Para definir el tamaño de un modelo en $\mathbf{KP}_\Sigma$, observamos que es común asumir que los estados de un modelo son vectores cuyos componentes corresponden a valores de las variables. En tal caso, el número de estados es exponencial con respecto al número de variables y por lo tanto el tamaño de un modelo se puede estimar considerando solamente $|\mathbb{S} \times \mathbb{S}|$ o simplemente $|\mathbb{S}|$. En general, sin embargo, es posible que el número de variables sea mayor que el número de estados. Además, algunas partes de $\textsc{Upd}_{prot}$ dependen no sólo de los estados de modelo de entrada sino que también de las transiciones modelo. Por lo tanto, definimos el tamaño de un modelo como sigue.

Definición 3.8.2 (Tamaño de modelos protegidos)**.** Si $(\mathcal{M}, P) \in \mathbf{KP}_\Sigma$ es un modelo protegido, el *tamaño* de $(\mathcal{M}, P)$ se define como $|(\mathcal{M}, P)| = \text{máx}\{|\text{Lit}(\mathbb{V})|, |\mathbb{S} \times \mathbb{S}|\}$.

Note que todos los modelos protegidos de $\mathbf{KP}_\Sigma$ tienen el mismo tamaño. Además, si $(\mathcal{M}, P) \in \mathbf{KP}_\Sigma$, entonces las operaciones de actualización que ocurren durante la ejecución de $\text{XU}\textsc{pd}_{prot}((\mathcal{M}, P), s, \varphi)$ preservan Σ. Por lo tanto, usamos $m = |(\mathcal{M}, P)|$ como un parámetro fijo en el siguiente análisis de complejidad.

Definición 3.8.3 (Número de pasos)**.** Sean $(\mathcal{M}, P) \in \mathbf{KP}_\Sigma$, $s \in S^{\mathcal{M}}$, y $\varphi \in \Sigma$-CTL. Si $m = |(\mathcal{M}, P)|$, $n = |\varphi|$, y $\varphi \in \Sigma$-XCTL, usamos $T_m(n)$ para denotar el *número de pasos* que se requieren en una ejecución no determinista de $\text{XU}\textsc{pd}_{prot}((\mathcal{M}, P), s, \varphi)$. Si $\varphi \in \Sigma\text{-CTL} - \Sigma\text{-XCTL}$, usamos $T'_m(n)$ para denotar el *número de pasos* que se requieren en una ejecución no determinista de $\textsc{Upd}_{prot}((\mathcal{M}, P), s, \varphi)$.

Usamos $T_m(n)$ para analizar la complejidad de tiempo en el peor de los casos de $\text{XU}\textsc{pd}_{prot}$. El siguiente Teorema muestra que, en el peor de los casos, $\text{XU}\textsc{pd}_{prot}$ tiene complejidad de tiempo del orden m^n.

Teorema 3.8.4 (Complejidad de $\text{XU}\textsc{pd}_{prot}$)**.** *Sean $(\mathcal{M}, P) \in \mathbf{KP}_\Sigma$, $s \in S^{\mathcal{M}}$, y $\varphi \in \Sigma$-XCTL. Si $m = |(\mathcal{M}, P)|$ y $n = |\varphi|$, entonces $T_m(n) \in \mathcal{O}(m^n)$.*

Demostración. Analizamos $T_m(n)$ de acuerdo con la estructura de φ.

1. Si $\varphi = \mathsf{F}$ o $\varphi = \mathsf{T}$, entonces:

   La ejecución de $\text{XUPD}_{prot}((\mathcal{M}, P), s, \varphi)$, líneas (3–4), falla o regresa $(\mathcal{M}, P)$.

   En ambos casos, la ejecución requiere un solo paso.

   Por lo tanto, si $n = |\mathsf{F}| = |\mathsf{T}|$ entonces $T_m(n) = 1$.

2. Si $\varphi = \ell$, entonces:

   Ya que $|L^P(s)| \leq |\text{Lit}(\mathbb{V})| \leq m$, el cómputo de $\overline{\ell} \in L^P(s)$, línea (5), requiere un máximo de m pasos.

   Luego, para el cómputo de $\mathbf{L}^u((\mathcal{M}, P), s, \ell)$, línea (6), tenemos que calcular $L^{\mathcal{M}}[s \oplus \ell]$ y $L^P[s \oplus \ell]$.

   Así, para calcular $L^{\mathcal{M}}[s \oplus \ell]$ buscamos $L^{\mathcal{M}}(s)$ en $L^{\mathcal{M}}$, y modificamos $L^{\mathcal{M}}(s)$ agregando ℓ a $L^{\mathcal{M}}(s)$ y removiendo $\overline{\ell}$ de $L^{\mathcal{M}}(s)$.

   La búsqueda de $L^{\mathcal{M}}(s)$ en $L^{\mathcal{M}}$ se puede realizar en m pasos como máximo, porque $|S^{\mathcal{M}}| \leq m$.

   La adición (sin duplicación) de ℓ, y la eliminación de $\overline{\ell}$, se pueden realizar en $2m$ pasos a lo sumo, porque $|L^{\mathcal{M}}(s)| \leq |\text{Lit}(\mathbb{V})| \leq m$.

   Por lo tanto, el cómputo de $L^{\mathcal{M}}[s \oplus \ell]$ requiere un máximo de $3m$ pasos.

   Análogamente, el cómputo de $L^P[s \oplus \ell]$ requiere un máximo de $3m$ pasos.

   Así, $\text{XUPD}_{prot}((\mathcal{M}, P), s, \ell)$ necesita $m + 3m + 3m$ pasos como máximo.

   Por lo tanto, si $n = |\ell|$, $T_m(n) = 7m$.

3. Si $\varphi = \alpha \vee \beta$, entonces:

   Después de elegir en forma no determinista $\delta \in \{\alpha, \beta\}$ en un paso, línea (7), XUPD_{prot} hace una llamada recursiva a $\text{XUPD}_{prot}((\mathcal{M}, P), s, \delta)$, línea (8).

Sin pérdida de generalidad, supongamos que δ es la elección, entre α y β, que requiere más pasos de ejecución, y que $\delta = \alpha$.

Así, el número de pasos que requiere la ejecución de $\text{XUPD}_{prot}((\mathcal{M}, P), s, \alpha \vee \beta)$ es, a lo más, 1 más el número de pasos que necesita $\text{XUPD}_{prot}((\mathcal{M}, P), s, \delta)$.

Por lo tanto, si $n = |\alpha \vee \beta|$, $T_m(n) = 1 + T_m(n - (|\beta| + 1)) \leq 1 + T_m(n - 1)$.

4. Si $\varphi = \alpha \wedge \beta$, entonces:

   La línea (9) de XUPD_{prot} equivale a elegir en forma no determinista $(\mathcal{M}_\alpha, P_\alpha)$ entre los elementos de $\text{XUPD}_{prot}[(\mathcal{M}, P), s, \alpha]$.

   La linea (10) es otra llamada recursiva para calcular $\text{XUPD}_{prot}((\mathcal{M}_\alpha, P_\alpha), s, \beta)$.

   Por lo tanto, el número de pasos requeridos para calcular $\text{XUPD}_{prot}((\mathcal{M}, P), s, \alpha \wedge \beta)$ está dado por la suma de:

   $T_m(n - (|\beta| + 1))$ pasos para calcular $\text{XUPD}_{prot}((\mathcal{M}, P), s, \alpha)$,

   un paso para elegir $(\mathcal{M}_\alpha, P_\alpha)$ entre los elementos de $\text{XUPD}_{prot}[(\mathcal{M}, P), s, \alpha]$,

   y $T_m(n - (|\alpha| + 1))$ pasos para calcular $\text{XUPD}_{prot}((\mathcal{M}_\alpha, P_\alpha), s, \beta)$.

   Por lo tanto, si $n = |\alpha \wedge \beta|$,

   $T_m(n) = 1 + T_m(n - (|\beta| + 1)) + T_m(n - (|\alpha| + 1)) \leq 1 + T_m(n - 1) + T_m(n - 1) = 1 + 2T_m(n - 1)$.

5. Si $\varphi = \mathbf{EX}\,\alpha$, entonces:

   En un paso, la línea (11) elige $s' \in A^P[s]$ en forma no determinista.

   Para computar $\mathbf{T}^+_\exists((\mathcal{M}, P), s, s')$ en la línea (12), tenemos que agregar (s, s') a $R^\mathcal{M}$ y proteger (s, s') en E^P.

   La línea (12) agrega la transición (s, s') a $(\mathcal{M}, P)$ en $2m$ pasos y hace una llamada recursiva a $\text{XUPD}_{prot}(\mathbf{T}^+_\exists((\mathcal{M}, P), s, s'), s', \alpha)$ en $T_m(n - 1)$ pasos.

   Así, el número de pasos necesarios para calcular $\text{XUPD}_{prot}((\mathcal{M}, P), s, \mathbf{EX}\,\alpha)$ es uno más la suma de $2m$ y $T_m(n - 1)$.

Por lo tanto, si $n = |\mathbf{EX}\,\alpha|$, $T_m(n) = 1 + 2m + T_m(n-1)$.

6. Si $\varphi = \mathbf{AX}\,\alpha$, entonces:

   En un paso, la línea (13) elige $S' \in \{X \subseteq A^P[s] \mid E^P[s] \subseteq X \text{ y } X \neq \varnothing\}$ en forma no determinista.

   Para computar $\mathbf{T}^u_\forall((\mathcal{M}, P), s, S')$ en línea (14), tenemos que calcular $\mathbf{T}^u(\mathcal{M}, s, S')$ y $(A^P - s \times (A^P[s] - S'))$.

   El cálculo de $\mathbf{T}^u(\mathcal{M}, s, S')$ requiere un máximo de $3m$ pasos: m pasos para calcular $R^{\mathcal{M}}[s]$, m pasos para eliminar $\{s\} \times R^{\mathcal{M}}[s]$ de $R^{\mathcal{M}}$, y m pasos para agregar $\{s\} \times S'$ a $R^{\mathcal{M}}$.

   El cálculo de $(A^P - \{s\} \times (A^P[s] - S'))$ requiere $3m$ pasos como máximo: m pasos para calcular $A^P[s]$, m pasos para calcular $A^P[s] - S'$, y m para eliminar $\{s\} \times (A^P[s] - S'))$ de A^P.

   Por lo tanto, el cálculo de $\mathbf{T}^u_\forall((\mathcal{M}, P), s, S')$ en la línea (14) requiere un máximo de $6m$ pasos.

   Por otro lado, la llamada a $\text{XUPD}^*_{prot}(\mathbf{T}^u_\forall((\mathcal{M}, P), s, S'), S', \alpha)$ en la línea (14) equivale a $|S'|$ llamadas consecutivas a $\text{XUPD}_{prot}(\_, s', \alpha)$ con cada $s' \in S'$.

   Ya que $|S'| \leq m$ y $|\alpha| = n - 1$, el número máximo de pasos requeridos por todas estas llamadas a XUPD_{prot} es $mT_m(n-1)$.

   Por lo tanto, si $n = |\mathbf{AX}\,\alpha|$, $T_m(n) = 1 + 6m + mT_m(n-1)$.

El peor caso de $\text{XUPD}_{prot}((\mathcal{M}, P), s, \varphi)$ es cuando $\varphi = \mathbf{AX}\,\alpha$.

Por lo tanto, una cota inferior para la complejidad, en el peor de los casos, de XUPD_{prot} está dado por una solución de la relación de recurrencia definida por

$T_m(n) = 1 + 6m + mT_m(n-1)$, si $n > 1$, y $T_m(1) = 7m$.

El factor m en $T_m(1)$ y el término $mT_m(n-1)$ en $T_m(n)$ implican que una solución de esta relación de recurrencia tiene un factor m^n.

Por lo tanto, $T_m(n) \in \mathcal{O}(m^n)$. □

A continuación, mostramos cómo se extiende el teorema anterior al resto de las fórmulas de Σ-CTL.

Teorema 3.8.5 (Complejidad de UPD_{prot})**.** *Sean* $(\mathcal{M}, P) \in \mathbf{KP}_\Sigma$, $s \in S^{\mathcal{M}}$, *y* $\varphi \in \Sigma\text{-}CTL - \Sigma\text{-}XCTL$. *Si* $m = |(\mathcal{M}, P)|$, *y* $n = |\varphi|$, *entonces* $T'_m(n) \in \mathcal{O}(m^{n+1})$.

Demostración. Sea ψ una subfórmula de φ tal que $\psi \in \mathbf{CTL}$ tiene la forma $\mathbf{E}[\alpha\,\mathbf{U}\,\beta]$ o $\mathbf{A}[\alpha\,\mathbf{U}\,\beta]$ o $\mathbf{E}[\alpha\,\mathbf{R}\,\beta]$ o $\mathbf{A}[\alpha\,\mathbf{R}\,\beta]$.

La evaluación de $\text{UPD}_{prot}((\mathcal{M}, P), s, \psi)$ se realiza reemplazando ψ con la fórmula de punto fijo correspondiente ψ_{fix}.

Por un lado, el tamaño de ψ_{fix} es lineal respecto al tamaño de ψ, es decir $|\psi_{fix}| = c * |\psi|$ para alguna $c \in \mathbb{N}$.

Por otro lado, la aplicación de UPD_{prot} a ψ_{fix} implica la aplicación recursiva de XUPD_{prot} a ψ en un estado s', donde s' es un sucesor de s que se obtiene de $R^{\mathcal{M}}$ con un costo de m pasos.

En cada llamada recursiva a XUPD_{prot} el par (ψ, s') se agrega a una lista de estados visitados con la fórmula ψ. La ejecución de XUPD_{prot} continúa de esta manera siempre que el par (ψ, s') no represente un estado que ya fue visitado con ψ.

Por lo tanto, el número máximo de llamadas recursivas a XUPD_{prot} está acotado por m.

Así, por el Teorema 3.8.4,

$T'_m(n) \leq m * (m + T_m(n)) \leq m^2 + m * d * m^n$, para alguna $d \in \mathbb{N}$.

Por lo tanto, $T'_m(n) \leq k * m^{n+1}$, para alguna $k \in \mathbb{N}$, y $T'_m(n) \in \mathcal{O}(m^{n+1})$. □

Capítulo 4

Pruebas de desempeño

En este capítulo, utilizando dos implementaciones de UPD_{prot} y de UPD_1, mostramos los resultados de varias pruebas de desempeño. Estas dos implementaciones se programaron utilizando dos lenguajes de programación que facilitan la implementación de operaciones no deterministas, Prolog y Curry [21].

A continuación, Ilustramos el comportamiento de UPD_{prot} con varios ejemplos, incluyendo un problema clásico de exclusión mutua y la actualización de modelos con cientos de estados. En el sitio

```
http://turing.iimas.unam.mx/ctl_upd3/form1.prl          (4-1)
```

está disponible una aplicación web que permite ejecutar una implementación de UPD_{prot}, con ejemplos preconstruidos, con ejemplos definidos mediante una interfaz sencilla, o cargando ejemplos construidos fuera de línea.

4.1. Síntesis de un modelo de exclusión mutua

Emerson y Clarke [17] presentan un método para sintetizar automáticamente *esqueletos de sincronización*, a partir de una especificación CTL, mediante un procedimiento de

decisión que construye un modelo de la especificación. Usamos el ejemplo de exclusión mutua de Emerson y Clarke [17] para mostrar que nuestro método de actualización CTL también se puede utilizar como un procedimiento de decisión para el problema de satisfactibilidad de fórmulas CTL.

La especificación del problema de exclusión mutua descrita por Emerson y Clarke utiliza una variante de CTL *con procesos* [17]. Adaptamos esta especificación a una especificación en Σ-CTL, Γ_{mutExc}, que utiliza las variables n_i, t_i, c_i, y $turn_i$, $i \in \{1, 2\}$, con el siguiente significado intuitivo:

- n_i: "El proceso i está en su *región no crítica*"
- t_i: "El proceso i *intenta entrar* en su región crítica"
- c_i: "El proceso i está en su *región crítica*"
- $turn_i$: "*Es el turno* del proceso i"

La especificación Γ_{mutExc} es el resultado de factorizar **AG** en la conjunción de las siguientes fórmulas, donde $i, j \in \{1, 2\}$ y $j \neq i$:

1. Estado de inicio. Ambos procesos están en su región no crítica:

   $(n_1 \wedge n_2)$

2. Cada proceso i está siempre exactamente en una de las tres regiones de código:

   $\mathbf{AG}\,((n_i \vee t_i \vee c_i) \wedge (n_i \rightarrow \neg(t_i \vee c_i)) \wedge (t_i \rightarrow \neg(n_i \vee c_i)) \wedge (c_i \rightarrow \neg(n_i \vee t_i)))$

3. Cualquier movimiento que el proceso i hace desde su región no crítica (crítica) es hacia el interior de su región de intento (no crítica), y tal movimiento siempre es posible:

   $$\mathbf{AG}\,(\quad (n_i \rightarrow ((\mathbf{AX}\,(t_i \vee n_i)) \wedge (\mathbf{EX}\,t_i))) \wedge \\ \qquad (c_i \rightarrow ((\mathbf{AX}\,(n_i \vee c_i)) \wedge (\mathbf{EX}\,n_i)))\quad)$$

4. Cualquier movimiento que el proceso i hace desde su región de intento es hacia el interior de su región crítica, y tal movimiento es posible cuando es el turno del proceso i:

   $$\mathbf{AG}\,((t_i \rightarrow (\mathbf{AX}\,(c_i \vee n_i))) \wedge ((t_i \wedge turn_i) \rightarrow (\mathbf{EX}\,c_i))\ )$$

5. Una transición mediante un proceso no puede causar que otro proceso se mueva. Si el proceso i está en la región $r_i \in \{n_i, t_i, c_i\}$ y el proceso j se mueve, entonces el proceso i permanece en la región r_i:

   $$\begin{aligned}\mathbf{AG}\,(\quad & ((r_i \wedge n_j) \rightarrow (\mathbf{AX}\,(t_j \rightarrow r_i))) \wedge ((r_i \wedge t_j) \rightarrow (\mathbf{AX}\,(c_j \rightarrow r_i))) \\ & \wedge\ ((r_i \wedge c_j) \rightarrow (\mathbf{AX}\,(n_j \rightarrow r_i)))\ )\end{aligned}$$

6. Algún proceso siempre se puede mover. Si algún proceso está en su región no crítica entonces ambos procesos se pueden mover; de lo contrario sólo un proceso se puede mover:

   $$\mathbf{AG}\,(((n_1 \vee n_2) \rightarrow (turn_1 \wedge turn_2)) \wedge ((\neg n_1 \wedge \neg n_2) \rightarrow (turn_1 \veebar turn_2))\ )$$

7. Cada transición se debe al movimiento de exactamente un proceso:

   $$\begin{aligned}\mathbf{AG}\,(\quad & ((turn_1 \wedge turn_2) \rightarrow (\mathbf{OD}_{\leq}\,2)) \wedge \\ & ((\neg turn_1 \vee \neg turn_2) \rightarrow (\mathbf{OD}_{\leq}\,1))\ \ )\end{aligned}$$

8. El estado $s = (t_1, t_2, turn_1, turn_2)$ se divide en los estados $(t_1, t_2, turn_1, \neg turn_2)$ y $(t_1, t_2, \neg turn_1, turn_2)$, y las transiciones que van hacia s se separan. Este requisito refleja una preferencia de Emerson y Clarke para distinguir todos los estados mediante sus etiquetas proposicionales [17, p. 258]:

   $$\mathbf{AG}\,((t_i \wedge n_j) \rightarrow (\mathbf{EX}\,(t_j \wedge turn_i)))$$

Observe que, en la especificación Γ_{mutExc}, sólo usamos fórmulas para especificar la *estructura local* del sistema, a través de los operadores $\mathbf{AX}$, $\mathbf{EX}$, y un operador más exterior $\mathbf{AG}$. No incluimos fórmulas para especificar *comportamiento global* del sistema mediante el uso de los operadores $\mathbf{F}, \mathbf{G}, \mathbf{U}$, o $\mathbf{R}$. Para las fórmulas de comportamiento

global, por ejemplo $\mathbf{AG}\,(t_i \rightarrow \mathbf{EF}\,c_i)$ y $\mathbf{AG}\,\neg\mathbf{EF}\,(c_1 \wedge c_2)$, se espera generalmente que su satisfacción sea una consecuencia de las fórmulas que especifican estructura local. Intuitivamente, es relativamente más fácil sintetizar un modelo a partir de fórmulas que especifican estructura local que sintetizar un modelo a partir de fórmulas que especifican comportamiento global. En lugar de usarse para síntesis, las fórmulas de comportamiento global se pueden utilizar para actualizar un modelo defectuoso que es hipotéticamente *cercano* a un modelo correcto. Una diferencia entre la síntesis de modelos y la actualización de modelos está relacionada con la diferencia entre estos dos tipos de fórmula.

Más adelante mostramos los resultados de pruebas realizadas para comparar $\textsc{Upd}_1$ y $\textsc{Upd}_{prot}$ en la reparación de varios modelos defectuosos de exclusión mutua respecto a la especificación Γ_{mutExc}. Los modelos que usamos como entrada en estas pruebas se obtuvieron eliminando diversas transiciones y etiquetas del modelo de exclusión mutua $\mathcal{M}_{mutExc}$ (Figura 4-1, a la derecha), donde las variables $turn_i$ se omiten para mayor claridad. El modelo $\mathcal{M}_{mutExc}$ satisface a Γ_{mutExc} en s_0, pero suprimiendo algunas transiciones y etiquetas en $\mathcal{M}_{mutExc}$ se obtienen modelos defectuosos que no satisfacen Γ_{mutExc}.

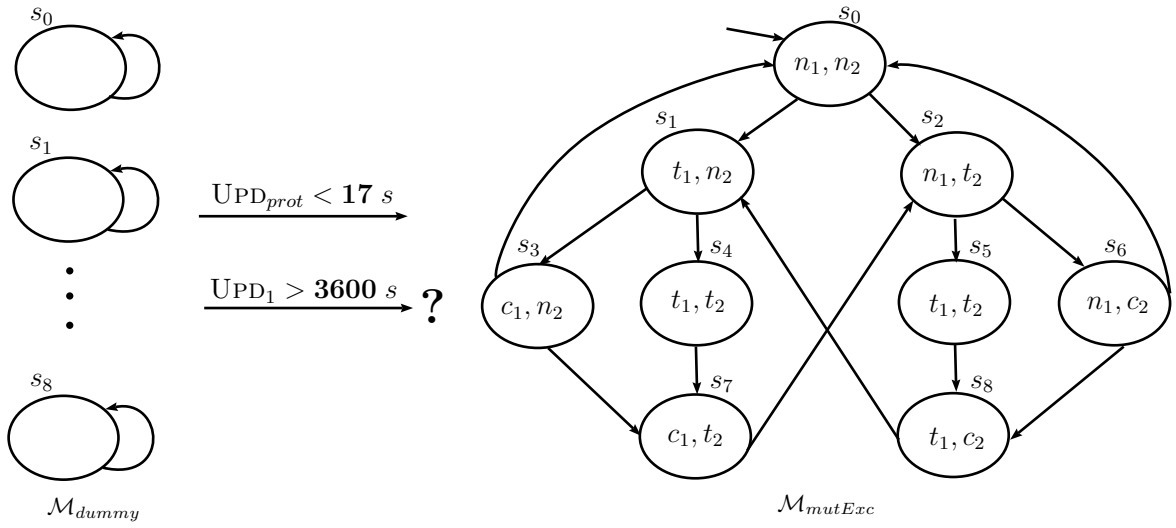

Figura 4-1: $\textsc{Upd}_1$ vs. $\textsc{Upd}_{prot}$ *reparando* $\mathcal{M}_{dummy}$ respecto a Γ_{mutExc}.

En el proceso para obtener modelos defectuosos a partir de $\mathcal{M}_{mutExc}$, *suprimimos* solamente etiquetas positivas, es decir transformamos literales positivos en literales negativos. Además, si la eliminación de transiciones en $\mathcal{M}_{mutExc}$ provoca que un estado s no tenga sucesores, agregamos una transición (s, s) para que el modelo obtenido sea total. Así, al suprimir en $\mathcal{M}_{mutExc}$ todas las transiciones y todas las etiquetas obtenemos el modelo ficticio $\mathcal{M}_{dummy}$ (Figura 4-1, a la izquierda).

En la Tabla 4-1, comparamos $\textsc{Upd}_1$ y $\textsc{Upd}_{prot}$ en la reparación de modelos defectuosos de exclusión mutua respecto a Γ_{mutExc}. La columna N indica el número de cambios necesarios para reparar el modelo. Los modelos de entrada se obtienen mediante la eliminación, de acuerdo con las marcas ✗, de transiciones (s_i, s_j) y etiquetas n_i en s_j, en el modelo $\mathcal{M}_{mutExc}$ que se muestra en la Figura 4-1. Una marca ✗ en la columna D significa que el modelo de entrada es el modelo ficticio $\mathcal{M}_{dummy}$ (Figura 4-1, a la izquierda). Las columnas -cN restringen la búsqueda a un máximo de N cambios. Las columnas -c? registran búsquedas con un número ilimitado de cambios. Las entradas "—" significan que no hubo respuesta después de una hora de proceso. Las cuatro últimas columnas muestran el tiempo, en segundos, necesario para producir la primera solución usando una PC con un procesador de doble núcleo a 2.0 GHz y 2 GB de RAM.

Tabla 4-1: $\textsc{Upd}_1$ vs. $\textsc{Upd}_{prot}$ en la reparación de modelos defectuosos de exclusión mutua.

Cambios necesarios	Partes removidas (✗)							Tiempo (segundos)			
								Con ayuda (-cN)		Sin ayuda (-c?)	
N	s_0s_1	s_1s_3	s_3s_7	s_2s_5	n_1s_0	n_2s_1	D	$\textsc{Upd}_1$	$\textsc{Upd}_{prot}$	$\textsc{Upd}_1$	$\textsc{Upd}_{prot}$
1	✗							1.9	0.6	—	1.79
2	✗	✗						5.7	0.6	—	1.81
3	✗	✗	✗					67.0	0.8	—	1.83
4	✗	✗	✗	✗				1161.4	0.9	—	8.9
5	✗	✗	✗	✗	✗			1274.8	0.9	—	8.9
6	✗	✗	✗	✗	✗	✗		—	1.2	—	9.1
55	✗	✗	✗	✗	✗	✗	✗	—	14.9	—	16.4

Las pruebas, cuyos resultados muestra Tabla 4-1, se realizaron sobre modelos defectuosos que se obtuvieron removiendo gradualmente algunas partes del modelo $\mathcal{M}_{mutExc}$ de la Figura 4-1. Las pruebas consistieron en reparar los modelos defectuosos respecto a la especificación Γ_{mutExc}. Observe que mientras más partes se remueven, más tiempo requieren $\textsc{Upd}_1$, y $\textsc{Upd}_{prot}$, para producir una primera solución. La opción -cN ayuda a que $\textsc{Upd}_1$ y $\textsc{Upd}_{prot}$ limiten la búsqueda de soluciones a modelos que resultan de aplicar un máximo de N cambios al modelo de entrada.

Entre los resultados que se muestran en la Tabla 4-1, *podemos destacar* que:

1. Sin la ayuda del límite de cambios (opción -cN), $\textsc{Upd}_\mathbf{1}$ *no puede* reparar, en menos de una hora, un modelo que requiere solamente un cambio ($N = 1$).

   En contraste, $\textsc{Upd}_{prot}$ realiza esta reparación en *menos de dos segundos*.

2. Con la ayuda del límite de cambios (opción -cN), $\textsc{Upd}_\mathbf{1}$ necesita *1274 segundos* para reparar un modelo que requiere cinco cambios ($N = 5$) para satisfacer Γ_{mutExc}.

   En contraste, $\textsc{Upd}_{prot}$ realiza la misma reparación en *menos de un segundo*.

3. Para reparar el modelo $\mathcal{M}_{dummy}$ se requieren 55 cambios ($N = 55$). En este caso, $\textsc{Upd}_\mathbf{1}$ *no puede* producir ninguna solución después de una hora de ejecución (aun con ayuda del límite de cambios).

   En contraste, $\textsc{Upd}_{prot}$ produce una primera solución en *menos de 17 segundos* sin la ayuda del límite de cambios.

   En este caso, $\textsc{Upd}_{prot}$ produce el modelo $\mathcal{M}_{mutExc}$ (Figura 4-1). Es decir $\textsc{Upd}_{prot}$ *produce el modelo esperado*, el que describen Emerson y Clarke [17, Fig. 11, p. 259].

4. La fórmula que utilizamos, Γ_{mutExc}, *tiene tamaño 174* (1$\mathbf{AG}$, 8$\mathbf{EX}$, 24$\mathbf{AX}$, 2$\mathbf{OD}$, 90$\vee$, y 49$\wedge$). Γ_{mutExc} no es una formula tan simple como ρ_1 (fórmula 4-2). ρ_1 tiene tamaño 3 y es la fórmula que utilizan Zhang y Ding [26] para ilustrar su método (ver la sección 4.2).

4.2. Actualización del modelo de un horno

En la Sección 4.1, mostramos que la aplicación de $\textsc{Upd}_{prot}$ a un modelo de entrada ficticio (dummy) es capaz de construir un modelo que satisface a una fórmula dada. Suponemos ahora que el modelo de entrada es el modelo no ficticio, $\mathcal{M}_{oven}$, que se muestra en la Figura 4-2.

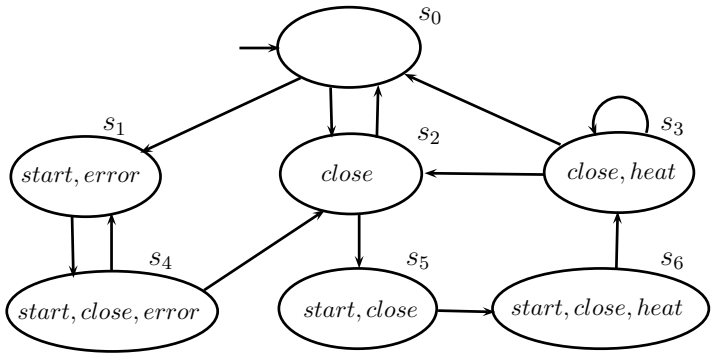

Figura 4-2: $\mathcal{M}_{oven}$, un modelo de Kripke para un horno de microondas.

Clarke et al. [12] usan $\mathcal{M}_{oven}$ para ilustrar la técnica de verificación de modelos (model checking) y, siguiendo a Zhang y Ding [26], usamos $\mathcal{M}_{oven}$ para mostrar la aplicación de $\textsc{Upd}_{prot}$ en la actualización de (modificaciones de) $\mathcal{M}_{oven}$ respecto a algunas fórmulas dadas.

La intuición detrás de $\mathcal{M}_{oven}$ está en la siguiente especificación informal de la operación de un horno de microondas *no muy parecido* a los hornos actuales. Entre paréntesis indicamos las transiciones de $\mathcal{M}_{oven}$ relacionadas con la respectiva instrucción de operación.

1. Abra la puerta y coloque los alimentos en el interior ($\rightarrow s_0$).

2. Cierre la puerta ($s_0 \rightarrow s_2$).

3. Presione START ($s_2 \rightarrow s_5$) para iniciar el calentamiento del horno ($s_5 \rightarrow s_6$).

4. Después del calentamiento, la cocción se inicia automáticamente ($s_6 \rightarrow s_3$).

5. Cuando finaliza la cocción, el calor se apaga ($s_3 \to s_2$).

6. Durante la cocción, la puerta se puede abrir ($s_3 \to s_0$).

7. Presionar START mientras la puerta está abierta produce un error ($s_0 \to s_1$).

8. En caso de error, cierre la puerta ($s_1 \to s_4$) y presione RESET ($s_4 \to s_2$).

A continuación, formalizamos la especificación informal anterior mediante una fórmula CTL, Ψ_{oven}, que utiliza las variables *start*, *close*, *heat* y *error*, con el siguiente significado intuitivo:

- *start*: "El botón START fue presionado".
- *close*: "La puerta del horno está cerrada".
- *heat*: "El calor del horno está apagado".
- *error*: "Se produjo un error".

La especificación Ψ_{oven} es la fórmula CTL que resulta al factorizar **AG** en la conjunción de las siguientes fórmulas:

1. Estado inicial:

   $\Psi_1 = (\neg start \wedge \neg close \wedge \neg heat \wedge \neg error)$

2. En cualquier momento, si la puerta está abierta, se puede cerrar:

   $\Psi_2 = \mathbf{AG}\,(\neg close \to \mathbf{EX}\, close)$

3. Si el calor está apagado, se puede presionar START:

   $\Psi_3 = \mathbf{AG}\,((\neg start \wedge \neg heat) \to \mathbf{EX}\, start)$

4. Si se presiona START y no hay ningún error, el calentamiento puede comenzar:

   $\Psi_4 = \mathbf{AG}\,((start \wedge close \wedge \neg heat \wedge \neg error) \to \mathbf{EX}\,(start \wedge heat))$

5. Después del calentamiento, la cocción puede comenzar:

   $\Psi_5 = \mathbf{AG}\,((start \wedge heat) \rightarrow \mathbf{EX}\,(\neg start \wedge heat))$

6. Durante la cocción, el calor se puede apagar (por tiempo transcurrido):

   $\Psi_6 = \mathbf{AG}\,((\neg start \wedge heat) \rightarrow \mathbf{EX}\,(close \wedge \neg heat))$

7. Durante la cocción, la puerta se puede abrir:

   $\Psi_7 = \mathbf{AG}\,((\neg start \wedge heat) \rightarrow \mathbf{EX}\,\neg close)$

8. Si se presiona START y la puerta está abierta, se produce un error inmediatamente:

   $\Psi_8 = \mathbf{AG}\,((start \wedge \neg close) \rightarrow error)$

9. En caso de error, si la puerta está abierta, el error persiste:

   $\Psi_9 = \mathbf{AG}\,((error \wedge \neg close) \rightarrow \mathbf{AX}\,error)$

10. En caso de error, si la puerta está cerrada, el horno se puede reiniciar:

    $\Psi_{10} = \mathbf{AG}\,((error \wedge close) \rightarrow \mathbf{EX}\,(\neg start \wedge close))$

El modelo $\mathcal{M}_{mutExc}$ de la Sección 4.1 está caracterizado por la especificación Γ_{mutExc}. A diferencia $\mathcal{M}_{mutExc}$, el modelo $\mathcal{M}_{oven}$ (Fig. 4-2) no está caracterizado por la especificación anterior Ψ_{oven}, pero se puede demostrar que $\mathcal{M}_{oven}$ satisface a Ψ_{oven} en s_0.

Por otro lado, si *definimos* ρ_1 mediante

$$\rho_1 = \mathbf{AG}\,(start \rightarrow \mathbf{AF}\,heat) \tag{4-2}$$

entonces ρ_1 expresa que *"si se presiona START, entonces el calor se enciende eventualmente"*. En tal caso, se puede demostrar que $\mathcal{M}_{oven}$ no satisface a ρ_1 en s_0.

Observe que ρ_1 expresa una *propiedad indeseable*. Si $\mathcal{M}_{oven}$ satisficiera ρ_1, entonces el calor podría encenderse con la puerta abierta y por tanto dañar al usuario. Incluimos ρ_1 en nuestras pruebas solamente con fines de comparación, siguiendo a Zhang y Ding [26].

En la Tabla 4-2, mostramos resultados de pruebas que comparan $\textsc{Upd}_1$ y $\textsc{Upd}_{prot}$ en la reparación de modificaciones del modelo $\mathcal{M}_{oven}$. El significado de las columnas en esta tabla es como sigue:

1. Los modelos de entrada para las pruebas se obtuvieron mediante modificaciones a $\mathcal{M}_{oven}$ de acuerdo a los cambios que se indican en la columna "Modificaciones".

   Usamos t, c, h, y e como abreviaturas de las variables *start*, *close*, *heat* y *error*, respectivamente. Dada $x \in \{t, c, h, e\}$, una modificación ix complementa la etiqueta x en el estado s_i. Una modificación ij agrega la transición (s_i, s_j) a $\mathcal{M}_{oven}$ si $(s_i, s_j) \notin R^{\mathcal{M}_{oven}}$, en caso contrario ij elimina (s_i, s_j) de $\mathcal{M}_{oven}$.

2. La columna φ muestra la fórmula utilizada para actualizar el modelo de entrada.

   Para $j \geq 0$, $\varphi_j = \rho_1 \wedge (\bigwedge_{i=1}^{j} \Psi_i)$.

   Por diseño, la modificación de $\mathcal{M}_{oven}$ en el renglón φ_j no satisface a $\bigwedge_{i=1}^{j} \Psi_i$.

3. La columna m, calculada mediante $\textsc{Upd}_{min}$, indica el número mínimo de cambios necesarios para reparar el modelo de entrada.

4. Las columnas -cm y -c? muestran el tiempo (en segundos) necesario para producir una primera solución.

   Las columnas -cm muestran soluciones con una búsqueda restringida a un máximo de m cambios.

   Las columnas -c? registran soluciones de búsquedas con un número ilimitado de cambios, es decir, sin usar m.

5. Las columnas n_1 y n_{prot} indican el número de cambios que aplicaron $\textsc{Upd}_1$ y $\textsc{Upd}_{prot}$, respectivamente, para generar una primera solución con la opción -c?.

6. Las entradas "—" indican que no se produjo ninguna respuesta antes de 600 segundos de proceso.

La protección inicial en la ejecución de UPD_{min} y UPD_{prot} es $P_{\perp}$. Estas pruebas se realizaron en una PC con un procesador de cuatro núcleos a 2.0 GHz y 8 GB de RAM.

Tabla 4-2: UPD_1 vs. UPD_{prot} en la reparación de modificaciones de $\mathcal{M}_{oven}$.

Input		UPD_{min}		Con ayuda (-cm)		Sin ayuda (-c?)		Cambios	
Modificaciones	φ	-c?	m	UPD_1	UPD_{prot}	UPD_1	UPD_{prot}	n_1	n_{prot}
Ninguna	ρ_1	0.1	1	0.1	0.1	0.1	0.1	4	4
$0h, 2c$	φ_2	0.1	3	0.1	0.1	—	0.1	—	7
$0h, 2c, 5t$	φ_3	0.4	4	2.0	0.1	—	0.1	—	10
$0h, 6h, 1h, 02, 3h$	φ_7	2.2	5	36.5	0.3	—	0.1	—	11
$6t$,$0t$,$2c$,$4t$,$4e$,$3t$,$6e$	φ_{10}	2.0	5	72.5	0.1	—	0.1	—	15

Los reglones de la Tabla 4-2 muestran la eliminación gradual de algunas partes de $\mathcal{M}_{oven}$ y el aumento gradual de la complejidad de la fórmula de entrada. Observe que mientras más partes se remueven, UPD_1 puede requerir más tiempo para producir una solución. La opción -cm ayuda a que UPD_1 y UPD_{prot} limiten la búsqueda de soluciones a modelos que resultan de aplicar un máximo de m cambios al modelo de entrada.

De las pruebas con el modelo del horno de microondas podemos destacar lo siguiente:

1. Observe que ambos procedimientos, UPD_1 y UPD_{prot}, son capaces de actualizar $\mathcal{M}_{oven}$ respecto a ρ_1. Con el input $(\mathcal{M}_{oven}, \rho_1)$, UPD_1 y UPD_{prot} producen una primera solución con cuatro cambios en menos de un segundo, incluso sin la ayuda que limita el número de cambios (Tabla 4-2).

   Sin embargo, la actualización realizada por UPD_1 en este caso puede causar que $\mathcal{M}_{oven}$ pierda algunas de sus propiedades, por ejemplo, la satisfacción de (partes de) la especificación Ψ_{oven}.

   De hecho, la primera solución de $\text{UPD}_1(\mathcal{M}_{oven}, s_0, \rho_1)$ no satisface a Ψ_{oven}. Peor aún, muchas de las primeras soluciones de $\text{UPD}_1(\mathcal{M}_{oven}, s_0, \rho_1)$ no satisfacen Ψ_{oven}, como se puede deducir de el tiempo de ejecución de $\text{UPD}_1(\mathcal{M}_{oven}, s_0, \varphi_{10})$ (Tabla 4-2).

2. En la práctica, la fórmula que se utiliza para actualizar un modelo determinado suele ser más compleja que ρ_1. Una fórmula para actualizar un modelo puede incluir subfórmulas que expresan propiedades que motivaron el diseño del modelo, es decir, una especificación del modelo (por ejemplo, Ψ_{oven}).

3. En las columnas "Sin ayuda (-c?)" de la Tabla 4-2, podemos observar que, sin la ayuda del mínimo número de cambios, $\textsc{Upd}_1$ puede producir una respuesta en menos de 600 segundos *solamente* cuando el input es $(\mathcal{M}_{oven}, \rho_1)$, es decir, el modelo sin modificaciones y la fórmula más simple. En contraste, $\textsc{Upd}_{prot}$ produce una primera respuesta en menos de un segundo con todos los modelos y fórmulas de la columna "Input".

4.3. Actualización del modelo de un contador

El siguiente ejemplo muestra el comportamiento de $\textsc{Upd}_{prot}$ en un problema fácilmente escalable: un contador síncrono. El no determinismo para este ejemplo se refleja con una entrada binaria externa (*halt*) que permite hacer que el contador cuente (*halt*=0) o se detenga (*halt*=1). Cada estado del contador tiene exactamente dos sucesores (uno para cada valor de *halt*), con la intención de representar un sistema con dos sucesores por estado en promedio. En lugar de hacer que el contador se detenga inmediatamente, la entrada *halt*=1 causa que el contador se detenga eventualmente. Este comportamiento de la entrada *halt* hace que nuestro ejemplo sea más representativo de los casos típicos en verificación de modelos, donde se requiere que una propiedad se cumpla eventualmente.

A continuación definimos una especificación Σ-CTL para un contador de n-bits. Esta especificación utiliza la variable h, y n variables x_i, donde $i \in \{0, \ldots, n-1\}$, con el siguiente significado intuitivo:

- h: "El contador se detendrá eventualmente".
- $\neg h$: "El contador seguirá contando".

- x_i: "El bit i-esimo del contador es 1" (x_0 es el bit menos significativo).

La especificación de un contador síncrono de n-bits, Ψ_c, es la conjunción de las fórmulas siguientes, donde $i \in \{0, \ldots, n-1\}$.

1. Cada estado tiene exactamente dos sucesores:

   $\mathbf{OD}_{\leq} 2 \wedge \mathbf{OD}_{>} 1$.

2. Si la entrada es "cuenta" ($\neg h$) y, o bien el bit i está en 1 o los i bits menos significativos están en 1, entonces el bit i estará próximamente en 1. En caso contrario el bit i estará próximamente en 0 (dos fórmulas para cada i):

   $(\neg h \wedge (x_i \veebar (x_0 \wedge \ldots \wedge x_{i-1}))) \rightarrow \mathbf{AX}\, x_i$

   $\neg(\neg h \wedge (x_i \veebar (x_0 \wedge \ldots \wedge x_{i-1}))) \rightarrow \mathbf{AX}\, \neg x_i$

   donde $\veebar$ es el operador *or exclusivo*. Por convención, $(x_0 \veebar (x_0 \wedge \ldots \wedge x_{0-1}))) = \neg x_0$.

3. Si, al no contar (h), se llega a un estado que tiene todos sus bits en 0, entonces el contador permanece allí:

   $(h \wedge \neg x_0 \wedge \ldots \wedge \neg x_{n-1}) \rightarrow \mathbf{AX}\, (\neg x_0 \wedge \ldots \wedge \neg x_{n-1})$

4. Cuando no está contando, el contador eventualmente llega a un estado que tiene todos sus bits en 0:

   $h \rightarrow \mathbf{A}[h\, \mathbf{U}\, \neg x_0 \wedge \ldots \wedge \neg x_{n-1}]$

En este ejemplo, asumimos que las etiquetas de los estados permanecen fijas, y actualizaremos solamente las transiciones. Esto se puede lograr fácilmente en $\textsc{Upd}_{prot}$ con una protección inicial que protege todas las etiquetas. En $\textsc{Upd}_1$, podemos simplemente inhibir el fragmento de código que modifica las etiquetas. Observe también que la especificación se debe satisfacer en todos los estados.

Los modelos de entrada se obtienen mediante la modificación intencional de un modelo que satisface la especificación, ya sea con la eliminación de transiciones correctas o con la

adición de transiciones incorrectas. Las transiciones incorrectas que se agregan son lazos, ciclos de un estado, de modo que si se eliminan todas las transiciones correctas y todos los estados tienen un lazo, obtenemos un modelo ficticio (dummy) como modelo de entrada, lo cual representa el peor de los casos.

La Tabla 4-3 muestra el tiempo, en segundos, que tarda UPD_{prot} para obtener un contador síncrono actualizando diferentes modelos defectuosos respecto a la especificación anterior, Ψ_c. (UPD_1 tarda más de una hora, incluso para las instancias más simples de la tabla.)

Tabla 4-3: UPD_{prot} aplicado a modelos defectuosos de contadores síncronos.

Num. estados	Trans. rem.	Lazos agreg.	UPD_{prot}
64	64	0	1
	96	32	7
	128	**64**	**13**
128	128	0	8
	192	64	77
	256	**128**	**155**

Num. estados	Trans. rem.	Lazos agreg.	UPD_{prot}
256	256	0	55
	384	128	1,355
	512	**256**	**2,662**
512	512	0	140
	576	64	950
	640	128	3,776

Las columnas etiquetadas “Num. estados” tienen el número de estados que resultan al considerar los bits del contador y el bit de control h. Las columnas etiquetadas “Trans. rem.” indican el número de transiciones removidas del modelo correcto. Las columnas etiquetadas “Lazos agreg.” indican el número de lazos añadidos. Las columnas “UPD_{prot}” muestran el tiempo que utiliza UPD_{prot} para obtener el modelo del contador mediante la reparación del modelo defectuoso respectivo. Las líneas en negrita se refieren a los modelos ficticios (dummy). Las pruebas se realizaron en una PC con un procesador de doble núcleo a 2.0 GHz y 2 GB de RAM.

Podemos resaltar los siguientes puntos de las pruebas realizadas con los modelos de contadores síncronos:

1. Con contadores de hasta 128 estados, UPD_{prot} puede reconstruir los modelos defectuosos, incluso los modelos ficticios (dummy), en menos de tres minutos.

2. Con 256 estados, UPD_{prot} puede obtener modelo del contador correspondiente, a partir del modelo ficticio, en menos de 45 minutos.

3. Si el número de estados es 512, y el modelo defectuoso se obtiene removiendo del modelo correcto 640 transiciones, más de la mitad del total, entonces UPD_{prot} puede reparar dicho modelo defectuoso en menos de 63 minutos.

4. Los modelos de estas pruebas tiene una estructura particular: contienen trayectorias relativamente largas que recorren todos los estados del modelo. Las trayectorias largas representan una dificultad especial para los algoritmos que recorren trayectorias.

5. Aunque la práctica actual de verificación de modelos emplea con frecuencia modelos con cientos de miles de estados, el hecho de que UPD_{prot} pueda manejar algunos ejemplos con cientos de estados muestra que nuestro método podría al menos ser útil para actualizar modelos que resultan al aplicar un proceso de abstracción a modelos grandes.

4.4. Actualización de modelos aleatorios

Para complementar las pruebas de la Sección 4.3, realizadas sobre modelos con una estructura particular, aquí mostramos el comportamiento de UPD_{prot} cuando se aplica a modelos y fórmulas *sin una estructura particular*, generados aleatoriamente.

La Tabla 4-4 (izquierda) muestra la aplicación de UPD_{prot} a un modelo aleatorio (fijo), M, generado con 10 variables, 1024 estados y 2048 transiciones.

El significado de las columnas de la Tabla 4-4 (izquierda) es como sigue:

Tabla 4-4: UPD$_{prot}$ aplicado a un modelo aleatorio M y una fórmula aleatoria φ.

M: 10 var, 1,024 sts, 2,048 trn		
Fórmula	Cam.	UPD$_{prot}$
$10\vee, 10\wedge, 20X$	4	7
$+2U$	2	5
$+2U, 2R$	2	23
$+2U, 2R, 2F$	3	411
$+2U, 2R, 2F, 2G$	—	—

φ: 10var,5$\vee$,5$\wedge$,2X,2U,2R,2F,2G			
Modelo edos-trans.		Cam.	UPD$_{prot}$
128	256	2	1
256	512	22	4
512	1,024	49	36
1,024	2,048	48	157
2,048	3,072	—	—

1. Los renglones de la columna "Fórmula" son formulas generadas aleatoriamente. Todas las fórmulas de esta columna son fórmulas con 10 variables y un conjunto de operadores que *contiene* 10 $\vee$, 10 $\wedge$, 10 **EX**, y 10 **AX**.

   La fórmula del primer renglón se generó con un conjunto de operadores que *tiene exactamente* 10 $\vee$, 10 $\wedge$, 10 **EX**, y 10 **AX**.

   La fórmula en el segundo renglón se generó usando, además de los 10 $\vee$, 10 $\wedge$, 10 **EX**, y 10 **AX**, dos operadores **U**: un **EU** y un **AU**.

   Las fórmulas de los demás renglones se generaron de manera similar. Por ejemplo, la fórmula del tercer renglón se generó usando, (además de los 10 $\vee$, 10 $\wedge$, 10 **EX**, y 10 **AX**) dos operadores **U** y dos operadores **R**: un **EU**, un **AU**, un **ER**, un **AR**.

2. La columna "Cam." muestra el número de cambios que UPD$_{prot}$ aplicó al modelo de entrada para actualizarlo.

3. La columna "UPD$_{prot}$" muestra el tiempo de ejecución (en segundos) requerido por UPD$_{prot}$ para actualizar el modelo de entrada.

4. Las entradas "—" significan que UPD$_{prot}$ no produjo un respuesta en 600 segundos (o menos).

   Descartamos tiempos de ejecución superiores a 600 segundos porque:

   a) la fórmula de entrada puede ser *insatisfactible*, o bien

b) el modelo y la fórmula pueden representar un caso de complejidad *difícil* (la complejidad de $\textsc{Upd}_{prot}$, en el peor de los casos, es exponencial, ver la Sección 3.8)

Los resultados de la Tabla 4-4 (izquierda) se obtuvieron aplicando $\textsc{Upd}_{prot}$ para actualizar el modelo M respecto a fórmulas aleatorias que se generaron con los operadores indicados en la primera columna, "Fórmula".

La Tabla 4-4 (derecha) muestra la aplicación de $\textsc{Upd}_{prot}$ a una fórmula aleatoria (fija), φ, generada con 10 variables, cinco operadores $\vee$, cinco $\wedge$, y uno de todos los demás operadores (**EX**, **AX**, **EU**, **AU**, **ER**, **AR**, **EF**, **AF**, **EG**, **AG**). Es decir, φ tiene 10 variables, 10 operadores proposicionales, y 10 operadores temporales.

El significado de las columnas de la Tabla 4-4 (izquierda) es como sigue:

1. Los renglones de las columnas "Modelo edos-trans." indican el número de estados y transiciones de modelos generados aleatoriamente utilizando 10 variables.

   Por ejemplo, el primer renglón de esta columna es un modelo generado aleatoriamente con 10 variables, 128 estados, y 256 transiciones.

2. El significado de las columnas "Cam." y "$\textsc{Upd}_{prot}$" es igual al significado de las respectivas columnas en la Tabla 4-4 (izquierda)

Los resultados de la Tabla 4-4 (derecha) se obtuvieron aplicando $\textsc{Upd}_{prot}$ para actualizar, respecto a la fórmula φ, los modelos de la columna "Modelo edos-trans." que se generaron aleatoriamente utilizando 10 variables y el número de estados y transiciones indicados en dicha columna.

De las pruebas realizadas con modelos y fórmulas generados aleatoriamente, podemos resaltar lo siguiente:

1. En la Tabla 4-4 (izquierda, cuarto renglón) podemos observar que, *en algunos casos*, $\textsc{Upd}_{prot}$ es capaz de actualizar un modelo aleatorio (con 10 variables, 1,024 estados, y 2,048 transiciones), respecto a una formula aleatoria (con 10 variables, 10 $\vee$, 10 $\wedge$,

10 **EX**, 10 **AX**, dos operadores **U**, dos **R**, y dos **F**), aplicando 3 cambios al modelo de entrada en 411 segundos.

2. En la Tabla 4-4 (derecha, cuarto renglón) podemos observar que, *en algunos casos*, UPD_{prot} es capaz de actualizar, respecto a una fórmula aleatoria (con 10 variables, cinco $\vee$, cinco $\wedge$, dos operadores **X**, dos **U**, dos **R**, dos **F**, y dos **G**), un modelo aleatorio (con 10 variables, 1,024 estados, y 2,048 transiciones), aplicando 48 cambios al modelo de entrada en 157 segundos.

3. Observe que los tiempos de ejecución de UPD_{prot} en las Tablas 4-4 son para *instancias especificas* del problema de actualización abordado por UPD_{prot}. Los tiempos de ejecución no pueden ser tan buenos para *todas las instancias* porque la complejidad de UPD_{prot}, en el peor de los casos, es exponencial respecto al tamaño de φ (ver la Sección 3.8).

4. A pesar de la complejidad exponencial, los resultados de las Tablas 4-4 muestran que UPD_{prot} puede tener un desempeño aceptable cuando se aplica a modelos con cientos de estados, *sin una estructura particular*, respecto a fórmulas CTL no simples.

4.5. Actualización de modelos con pocos estados

Por último, mencionamos un ejemplo que compara UPD_1 y UPD_{prot} con un modelo de un solo estado. Este ejemplo se puede encontrar en el sitio web de UPD_{prot}. La fórmula utilizada en este ejemplo se construyó de la siguiente manera.

Primero, usando 25 variables construimos una fórmula proposicional φ que resulta de la conjunción de 105 cláusulas con tres literales. Por construcción, φ es satisfactible por una única asignación de verdad.

Luego, transformamos φ en una fórmula temporal φ' prefijando cada literal de φ con el operador **EX**.

Los puntos que podemos destacar de las pruebas realizadas en las condiciones descritas previamente son:

1. Observamos que $\textsc{Upd}_{prot}$ actualiza un modelo ficticio de un solo estado, respecto a φ', en menos de 30 segundos.

   En contraste, con el mismo input, $\textsc{Upd}_1$ no produce ningún modelo después de más de una hora de ejecución.

2. Si cambiamos el modelo ficticio de un solo estado por un modelo ficticio de dos estados, entonces $\textsc{Upd}_{prot}$ produce una solución en menos de un segundo.

3. Por supuesto, una herramienta para la resolución de instancias de SAT puede resolver de manera eficiente el problema de este ejemplo usando φ en lugar de φ'. Sin embargo, este ejemplo muestra dos puntos:

   - Primero, incluso instancias del problema de actualización CTL con modelos de pocos estados pueden resultar difíciles para un actualizador CTL.
   - Segundo, un aumento en el número de estados puede causar una reducción en el tiempo requerido por un actualizador CTL. Es decir, la intuición de que un aumento en el número de estados aumenta la dificultad del problema de actualización CTL no es correcta en todos los casos.

4. Parte de la intuición detrás del ejemplo de esta sección es que el problema de actualización de modelos respecto a fórmulas de la CTL *contiene* al problema SAT, un problema NP-completo. Por lo tanto, no es posible que $\textsc{Upd}_{prot}$ produzca soluciones, para todas las entradas, en forma eficiente. Sin embargo, como ocurre con las herramientas para la resolución de instancias de SAT, puede resultar muy útil que $\textsc{Upd}_{prot}$, en casos prácticos, produzca soluciones con tiempos de ejecución aceptables.

Capítulo 5

Conclusiones

Dividimos las concusiones de este trabajo de acuerdo a los siguientes puntos que corresponden a etapas de nuestra investigación:

- Identificar el problema de actualización de modelos, respecto a fórmulas de la CTL.
- Definir una especificación formal de dicho problema.
- Demostrar formalmente que dicha especificación es correcta y completa.
- Implementar nuestra especificación del problema y observar el desempeño.
- Establecer el trabajo futuro.

Concluimos que el problema de actualización CTL de modelos es un problema importante, y que los métodos existentes para la solución de este problema tienen inconvenientes teóricos y prácticos (Sec. 5.1). Considerando dichos inconvenientes, concluimos que la propuesta de un método de actualización, basado en protecciones, aportaba conocimiento a la solución del problema (Sec. 5.2). Mediante demostraciones formales, concluimos que nuestra especificación es correcta y completa (Sec. 5.3). Después de implementar dicha especificación, y de realizar pruebas con varios ejemplos, concluimos que nuestro método tiene un desempeño aceptable (Sec. 5.4). Finalmente, concluimos que, entre las líneas de investigación futura, la incorporación de técnicas simbólicas en nuestro método tendría consecuencias importantes en términos de escalabilidad (Sec. 5.5).

5.1. Identificación del problema

Ubicamos la solución del problema de actualización de modelos como una consecuencia importante para las herramientas de verificación de modelos (Sec. 1.1). Además, *señalamos que la actualización de modelos es un problema importante* porque esta estrechamente relacionada con un área de investigación en Inteligencia Artificial, el razonamiento sobre acciones y cambio (Sec. 1.2).

Debido a que la CTL es una lógica con cuantificadores universales y existenciales, *pudimos señalar que los métodos para actualización CTL enfrentan dos dificultades:*

- La eliminación de una transición puede causar que se pierda la satisfacción de las subfórmulas existenciales previamente tratadas.
- La adición de una transición puede causar que se pierda la satisfacción de las subfórmulas universales previamente tratadas.

Revisamo1s los métodos existentes para actualización de modelos respecto a fórmulas de la CTL, *y concluimos que:*

- *Algunos de los métodos existentes se restringen a ACTL*, el fragmento universal de la CTL (Sec. 2.1, 2.6). Es decir, *estos métodos son incompletos* porque simplemente evitan la dificultad que implica tratar conjuntamente cuantificadores universales y existenciales.
- *Los métodos existentes que no se restringen a ACTL, tienen varios inconvenientes:*
  - Algunos métodos *utilizan heurísticas* para procesar la combinación de las fórmulas universales y existenciales (Sec. 2.2).

    Un inconveniente de las heurísticas es que éstas son frecuentemente incompletas y dependen de un dominio de aplicación particular.
  - Otros métodos *tienen una definición imprecisa e informal* (Sec. 2.3, 2.5, 2.8), o recurren a un método con una definición imprecisa (Sec. 2.7).

Entre otras consecuencias, la definición imprecisa e informal (o la heurística) de estos métodos impide hacer demostraciones formales de corrección y completitud.

- En general, estos métodos son muy ineficientes porque utilizan un procedimiento similar a los métodos de *generar-y-probar*, sobre un espacio de búsqueda muy grande.

  Estos métodos modifican un modelo dado para *generar un modelo candidato* que satisfaga fórmulas universales (existenciales) sin tener en cuenta que, antes de la modificación, el modelo dado satisfacía algunas fórmulas existenciales (universales).

  Entre otras consecuencias, esta forma de actualización provoca que se deshagan cambios previamente hechos, y que sea necesario *probar (test) si el modelo candidato satisface* las fórmulas previamente satisfechas.

5.2. Especificación formal de una solución

Considerando que los métodos existentes para la actualización de modelos, respecto a fórmulas de la CTL, tienen los inconvenientes señalados previamente, *propusimos un método de actualización* basado en un concepto nuevo, protecciones (Def. 3.2.3).

Intuitivamente, una protección P consta de tres componentes, $\langle E, A, L\rangle$. En E, la protección existencial, se registran las transiciones que no se pueden remover. En A, la protección universal, están registradas las transiciones que se pueden agregar. En L, la protección de etiquetas, están los literales que no se pueden modificar.

Las protecciones permiten superar las dificultades inherentes a la actualización respecto a fórmulas con cuantificadores universales y existenciales. Esto se logra preservando la satisfacción de fórmulas a través del proceso de actualización.

En lugar de *probar* (verificar) que un modelo *generado* satisface fórmulas previamente satisfechas, nuestro mecanismo de protecciones *protege la satisfacción* de las fórmulas al

realizar las modificaciones. Consecuentemente, el método de actualización que propusimos, basado en protecciones, es más eficiente que los métodos que utilizan mecanismos similares a generar-y-probar.

Desarrollamos el concepto de protección mediante la definición *formal* de:

- Modelos protegidos, protección vacía, y protección completa (Def. 3.2.4).
- Una semántica para la CTL mediante modelos protegidos (Def. 3.2.5).
- Una especificación de las modificaciones aceptables (Def. 3.5.2).
- Un orden parcial sobre el conjunto de todas las protecciones (Def. 3.7.1).
- Las nociones φ-modificable (Def. 3.7.9), y $\mathbf{K}_\Sigma$-satisfactible (Def. 3.7.10).

Usando la formalización de las modificaciones de un modelo protegido, *diseñamos el algoritmo* XUPD_{prot}, para actualizar de modelos respecto a fórmulas XCTL, el fragmento modal de CTL (Fig. 3-6).

Después, usando un mecanismo para detección de ciclos, *extendimos* XUPD_{prot} *a* UPD_{prot} (Fig. 3-8), un algoritmo para actualizar modelos respecto a fórmulas CTL. *También diseñamos* XUPD_{S+}, un algoritmo de actualización que considera una operación para agregar estados (Fig. 3-9).

Para mejorar la eficiencia de UPD_{prot}, *propusimos heurísticas y estrategias de búsqueda* basadas en un orden sobre la generación de elecciones no deterministas (Sec. 3.6).

5.3. Corrección, completitud y complejidad

Demostramos que las operaciones de actualización, y las modificaciones, se comportan apropiadamente respecto al orden parcial de las protecciones (Teo. 3.7.3, y Teo. 3.7.4). Así, *concluimos que el orden de las protecciones preserva satisfactibilidad* (Teo. 3.7.5).

Usando las propiedades del orden sobre las protecciones, *demostramos que la especificación formal de las modificaciones, Modif , es correcta* (Teo. 3.7.7).

Recíprocamente, usando que siempre que un modelo es φ-modificable tenemos que φ es $\mathbf{K}_\Sigma$-satisfactible (Cor. 3.7.15), *concluimos que la especificación Modif es completa.*

Combinando ambos resultados, y asumiendo que XUPD_{prot} es una implementación de *Modif*, *demostramos que* XUPD_{prot} *es correcto y completo* (Teo. 3.7.16).

También dimos una demostración de que si m es el tamaño de $(\mathcal{M}, P)$, n es el tamaño de φ, y $T'_m(n)$ denota el número máximo de pasos, que requiere la ejecución no determinista de $\text{UPD}_{prot}((\mathcal{M}, P), s, \varphi)$, para producir una primera respuesta, entonces $T'_m(n) \in \mathcal{O}(m^{n+1})$ (Teo. 3.8.5). Es decir, *concluimos que la complejidad de* UPD_{prot}*, con tiempo no determinista, es exponencial respecto a* $|\varphi|$*, pero la base de dicha exponencial es* $|(\mathcal{M}, P)|$.

5.4. Implementación y pruebas de desempeño

Implementamos UPD_{prot} y realizamos pruebas de desempeño con varios ejemplos. El acceso a esta implementación está disponible mediante una aplicación web (Url: 4-1).

Para medir el desempeño de UPD_{prot} realizamos pruebas con los siguientes modelos y fórmulas:

- Un modelo de exclusión mutua (Sec. 4.1).

  Mostramos que UPD_{prot} es capaz de sintetizar un modelo de exclusión mutua a partir de un modelo ficticio de nueve estados, con una fórmula CTL de tamaño 174 (1**AG** , 8**EX** , 24**AX** , 2**OD**, 90$\vee$, y 49$\wedge$), en menos de 17 segundos.

  Comparado con UPD_1, un método generar-y-probar, UPD_{prot} resultó dramáticamente superior.

- Un modelo de un horno (Sec. 4.2).

  Mostramos que, si usamos un método generar-y-probar, UPD_1, para reparar el modelo de un horno de microondas, respecto a una fórmula utilizada por Zhang y

Ding [26] (ρ_1 4-2), el desempeño es engañoso. Al requerir 0.1 segundos de proceso en este caso, UPD_1 aparenta eficiencia.

El desempeño real de UPD_1 surge al aumentar el número de cambios necesarios para reparar el modelo, o al aumentar el tamaño de la fórmula. Con un aumento en la complejidad de los datos, UPD_1 ya no produce una respuesta inmediata, no responde en menos de 10 minutos.

En contraste, UPD_{prot} responde, en todas las pruebas con este modelo, en 0.1 segundos (Tabla 4-2).

- Modelos con cientos de estados y una estructura particular (Sec. 4.3).

  En esta prueba utilizamos modelos que representan contadores de diferentes tamaños con una señal externa que les permite o bien contar o reiniciarse.

  UPD_{prot} fue capaz de sintetizar completamente (a partir de un modelo ficticio) casos con hasta 256 estados en menos de una hora. UPD_{prot} también pudo reparar un modelo defectuoso de 512 estados, haciendo 768 cambios, en alrededor de una hora de tiempo de proceso con una PC común.

  Menos cambios significarían la posibilidad de manejar modelos más grandes.

- Modelos con cientos de estados y sin una estructura particular (Sec. 4.4).

  Pudimos observar que, *en algunos casos*, UPD_{prot} es capaz de actualizar un modelo aleatorio fijo (con 10 variables, 1,024 estados, y 2,048 transiciones), respecto a formulas aleatorias (con 10 variables, 10 $\vee$, 10 $\wedge$, 10 **EX**, 10 **AX**, dos operadores **U**, dos **R**, y dos **F**), aplicando 3 cambios al modelo de entrada en 411 segundos.

  También pudimos observar que, *en algunos casos*, UPD_{prot} es capaz de actualizar, respecto a una fórmula aleatoria fija (con 10 variables, cinco $\vee$, cinco $\wedge$, dos operadores **X**, dos **U**, dos **R**, dos **F**, y dos **G**), modelos aleatorios (con 10 variables, 1,024 estados, y 2,048 transiciones), aplicando 48 cambios al modelo de entrada en 157 segundos.

Los tiempos de ejecución no pueden ser tan buenos para *todos los casos* debido a la complejidad del problema de actualización de modelos respecto a fórmulas CTL.

- Modelos con un estado (Sec. 4.5).

  En esta prueba utilizamos un modelo con un estado y una fórmula CTL con 25 variables, φ', que obtuvimos prefijando con **EX** los literales de una conjunción de 105 cláusulas proposicionales con tres literales. Por construcción, φ' es satisfactible por un único etiquetamiento del modelo.

  Con este ejemplo, mostramos que incluso modelos de un estado pueden resultar difíciles para un actualizador CTL (recordando que el problema de actualización CTL de modelos contiene a SAT).

  También mostramos que la intuición de que un aumento en el número de estados aumenta la dificulta el problema de actualización CTL no es correcta en todos los casos.

  Por ejemplo, $\textsc{Upd}_{prot}$ pudo actualizar un modelo de un solo estado, respecto a φ', en 30 segundos. Mientras que, aumentando el número de estados a dos, $\textsc{Upd}_{prot}$ realizó la misma actualización, respecto a φ', en menos de un segundo.

5.5. Trabajo futuro

Identificamos varias vías posibles de investigación futura.

Primero, creemos que nuestras protecciones podrían beneficiarse de ideas utilizadas en otros métodos de actualización, incluyendo la actualización respecto a fórmulas de la LTL (Linear time logic). Por ejemplo, Jobstmann *et al.* [22] emplean juegos para realizar actualización de modelos respecto a fórmulas de la LTL. El modelo a ser corregido se transforma en un juego infinito donde tienen interacción el sistema (el protagonista) y el medio ambiente (el antagonista). De manera similar, la idea de incorporar a nuestro método de actualización una interacción con el usuario es interesante porque una disminución

del grado de automatización, con la ayuda del usuario, podría disminuir los tiempos de ejecución.

Segundo, la estrecha relación entre la actualización de modelos y razonamiento acerca de las acciones y el cambio (por ejemplo, [16]) sugieren investigar la posibilidad de adaptar nuestro enfoque a métodos desarrollados para este tipo de razonamiento.

Finalmente, nuestro método podría alcanzar un mejor escalamiento mediante la codificación de modelos protegidos con técnicas simbólicas, como los diagramas de decisión binarios ordenados (por ejemplo, [1]).

Bibliografía

[1] Bryant, R.E. Symbolic Boolean Manipulation with Ordered Binary-decision Diagrams. *ACM Comput. Surv.* **24**(3):293–318 (1992)

[2] Buccafurri, F., Eiter, T., Gottlob, G., y Leone, N. Enhancing model checking in verification by AI techniques. *Artificial Intelligence* **112**(1–2):57–104 (1999)

[3] Calzone, L., Chabrier-Rivier, N., Fages, F., y Soliman, S. Machine learning biochemical networks from temporal logic properties. En *Transactions on Computational Systems Biology VI*, *Lecture Notes in Bioinformatics*, tomo 4220, págs. 86–94. Springer (2006)

[4] Carrillo, M. y Rosenblueth, D.A. Learning Models from Temporal-Logic Properties via Explanations. En T.R. Roth-Berghofer, S. Schulz, y D.B. Leake (editores), *AAAI-07 workshop on Explanation-aware Computing ExaCt 2007*, Technical Report WS-07-06, págs. 10–13. AAAI Press (2007)

[5] Carrillo, M. y Rosenblueth, D.A. A Method for CTL Model Update, Representing Kripke Structures as "Table Systems". *International Journal of Pure and Applied Mathematics* **52**(3):401–431 (2009)

[6] Carrillo, M. y Rosenblueth, D.A. Nondeterministic Update of CTL Models by Preserving Satisfaction through Protections. En T. Bultan y P.A. Hsiung (editores),

Automated Technology for Verification and Analysis (ATVA 2011), *Lecture Notes in Computer Science*, tomo 6996, págs. 60–74. Springer (2011)

[7] Carrillo, M. y Rosenblueth, D.A. A source of incompleteness in CTL update methods. Manuscrito disponible en `https://sites.google.com/site/miguelmcb/home/publicaciones/incompleteness-CTL-update-AIJ-2012.pdf` (2012). Consulta: mayo 2014

[8] Carrillo, M. y Rosenblueth, D.A. CTL update of Kripke models through protections. *Artificial Intelligence* **211**(0):51–74 (2014)

[9] Chabrier-Rivier, N., Chiaverini, M., Danos, V., Fages, F., y Schächter, V. Modeling and querying biomolecular interaction networks. *Theor. Comput. Sci.* **325**(1):25–44 (2004)

[10] Chatzieleftheriou, G., Bonakdarpour, B., Smolka, S.A., y Katsaros, P. Abstract Model Repair. En A.E. Goodloe y S. Person (editores), *Proc. 4th NASA Formal Methods (NFM 2012)*, *Lecture Notes in Computer Science*, tomo 7226, págs. 341–355. Springer (2012)

[11] Cimatti, A., Clarke, E.M., Giunchiglia, F., y Roveri, M. NuSMV: a new Symbolic Model Verifier. En N. Halbwachs y D. Peled (editores), *Proc. Eleventh Conference on Computer-Aided Verification (CAV '99)*, *Lecture Notes in Computer Science*, tomo 1633, págs. 495–499. Springer (1999)

[12] Clarke, E.M., Grumberg, O., y Peled, D.A. *Model Checking*. The MIT Press (1999)

[13] Clarke, E., Jha, S., Lu, Y., y Veith, H. Tree-Like Counterexamples in Model Checking. En *17th IEEE Symposium on Logic in Computer Science (LICS 2002)*, págs. 19–29. Springer (2002)

[14] Ding, Y. *Model Update for System Modifications*. Tesis Doctoral, School of Computing and Mathematics, University of Western Sydney, Australia (2007)

[15] Ding, Y. y Hemer, D. An optimised algorithm to tackle the model explosion problem in CTL model update. En B.T. Zhang y M.A. Orgun (editores), *11th Pacific Rim International Conference on Artificial Intelligence (PRICAI 2010)*, *Lecture Notes in Artificial Intelligence*, tomo 6230, págs. 589–594. Springer (2010)

[16] Eiter, T., Erdem, E., Fink, M., y Senko, J. Updating action domain descriptions. *Artificial Intelligence* **174**:1172–1221 (2010)

[17] Emerson, E.A. y Clarke, E.M. Using branching time temporal logic to synthesize synchronization skeletons. *Science of Computer Programming* **2**:241–266 (1982)

[18] Emerson, E.A. y Halpern, J.Y. Decision Procedures and Expressiveness in the Temporal Logic of Branching Time. *Journal of Computer and System Sciences* **30**(1):1–24 (1985)

[19] Forouzan, B. (editor). *Data Communications and Networking*. McGraw-Hill (2003)

[20] Guerra, P.T. y Wassermann, R. Revision of CTL models. En *Ibero-American Conference on Artificial Intelligence (IBERAMIA 2010)*, *Lecture Notes in Artificial Intelligence*, tomo 6433, págs. 153–162. Springer (2010)

[21] Hanus, M. Functional Logic Programming: From Theory to Curry. En *Programming Logics - Essays in Memory of Harald Ganzinger*, págs. 123–168. Springer LNCS 7797 (2013)

[22] Jobstmann, B., Staber, S., Griesmayer, A., y Bloem, R. Finding and fixing faults. *Journal of Computer and System Sciences* **78**:441–460 (2012)

[23] Kelly, M. y Zhang, Y. Local model update with an application to sliding window protocol. En *Proc. 14th Knowledge-Based and Intelligent Information and*

Engineering Systems, Part IV (KES 2010), *Lecture Notes in Artificial Intelligence*, tomo 6279, págs. 11–21. Springer (2010)

[24] Kelly, M., Pu, F., Zhang, Y., y Zhou, Y. ACTL local model update with constraints. En *Proc. 14th Knowledge-Based and Intelligent Information and Engineering Systems, Part IV (KES 2010)*, *Lecture Notes in Artificial Intelligence*, tomo 6279, págs. 135–144. Springer (2010)

[25] Wing, J.M. y Vaziri-Farahani, M. A case study in model checking software. *Science of Computer Programming* **28**:273–299 (1997)

[26] Zhang, Y. y Ding, Y. CTL model update for system modifications. *Journal of Artificial Intelligence Research* **31**:113–155 (2008)