Desarrollo formal y verificación de sistemas proposicionales*

F.J. Martín–Mateos, J.A. Alonso, M.J. Hidalgo, y J.L. Ruiz–Reina
http://www.cs.us.es/{~fmartin,~jalonso,~mjoseh,~jruiz}

Departamento de Ciencias de la Computación e Inteligencia Artificial.
Escuela Técnica Superior de Ingeniería Informática, Universidad de Sevilla
Avda. Reina Mercedes, s/n. 41012 Sevilla, Spain

Resumen En este trabajo presentamos nuestra aproximación al desarrollo formal y verificación de procedimientos para resolver el problema de satisfacibilidad proposicional (SAT). Esta tarea es realizada en el sistema ACL2, un entorno que permite combinar ejecución con verificación. Se formalizan dos esquemas genéricos para resolver el problema SAT, verificando sus propiedades de terminación, corrección y completitud. Distintos procedimientos para resolver el problema SAT se obtenienen como versiones concretas de estos esquemas genéricos. La prueba de las propiedades de terminación, corrección y completitud de estas versiones concretas se obtiene de forma automática a partir de las de los esquemas genéricos.

1. Introducción

La lógica proposicional ha sido objeto de estudio desde hace mucho tiempo. Por un lado, el problema de satisfacibilidad proposicional (determinar si una fórmula proposicional tiene algún modelo) fue el primer problema NP-completo conocido, de ahí el gran interés en resolverlo de manera eficiente. Por otro lado, los trabajos en lógica proposicional son punto de referencia para resolver problemas similares en otras lógicas.

En la práctica, el problema SAT es fundamental en la resolución de muchos problemas que aparecen en razonamiento automático, diseño asistido por ordenador, bases de datos, robótica, diseño de circuitos integrados, etc. Es por ello que continuamente se desarrollan nuevos procedimientos para resolverlo. Entre las propiedades que se pueden exigir a estos procedimientos destacan: *terminación*, que el procedimiento termine para cualquier dato de entrada; *corrección*, que una respuesta positiva del procedimiento implique la satisfacibilidad de la fórmula de entrada; y *completitud*, que toda fórmula satisfacible se pueda caracterizar con una respuesta positiva del procedimiento.

En algunas ocasiones los procedimientos para resolver el problema SAT no verifican estas propiedades. Esto no resulta del todo inconveniente cuando se trata de la terminación, lo cual implica que el procedimiento no es capaz de

* Este trabajo ha sido financiado por el proyecto TIC2000-1368-C03-02 del MCYT

dar una respuesta para determinado tipo de fórmulas. Sin embargo, la situación es peor cuando se trata de la corrección y la completitud, esto implica que el procedimiento proporciona una respuesta incorrecta sobre cierto tipo de fórmulas. Este es el caso del sistema POSIT [3] que responde afirmativamente sobre la satisfacibilidad del conjunto de cláusulas $\{\{p, q\}, \{p, \neg q\}, \{\neg p, q\}, \{\neg p, \neg q\}\}$, y del sistema REVEAL en el que se obtuvo una primera prueba errónea de la equivalencia entre las álgebras de Robbins y las álgebras de Boole.

Este problema se agrava cuando se utilizan estos procedimientos en otras aplicaciones. Por ejemplo, los sistemas NuSMV y MACE utilizan un procedimiento proposicional para la búsqueda de modelos. De esta forma las propiedades de dichos sistemas están supeditadas a las del proposicional. Otro caso es el del algoritmo de Stålmarck para demostrar la satisfacibilidad de una fórmula proposicional, utilizado en aplicaciones comerciales. La garantía de las propiedades de este procedimiento aumenta la confianza en la corrección de las aplicaciones en las que se utiliza.

Por otro lado, el desarrollo formal y verificación de sistemas de demostración proposicional mejora y aumenta la comprensión de los mismos y de sus propiedades de corrección y completitud. De esta forma el trabajo realizado puede ser utilizado en cursos de formación sobre razonamiento automático.

En este trabajo presentamos las líneas generales de nuestra aproximación al desarrollo formal y verificación de procedimientos para resolver el problema de satisfacibilidad proposicional (SAT). Para realizar esta tarea hemos escogido el sistema ACL2, que combina la definición y ejecución de procedimientos en el lenguaje Common Lisp, con la verificación de sus propiedades. Hemos realizado una formalización de la lógica proposicional que forma el entorno en el que se definen distintos sistemas proposicionales de demostración y se verifican sus propiedades. Hemos formalizado dos esquemas genéricos para resolver el problema SAT, a partir de los que se obtienen distintos procedimientos como casos particulares. Hemos demostrado las propiedades de terminación, corrección y completitud de los esquemas genéricos. La prueba de estas propiedades para los casos particulares se obtiene de forma automática a partir de las de los esquemas genéricos. Una versión ampliada de este trabajo se puede encontrar en [5].

2. Sistemas proposicionales basados en transformación

En esta sección describimos un marco genérico, con respecto al cual algunos sistemas proposicionales de demostración bien conocidos pueden expresarse como casos particulares. Estos sistemas pueden interpretarse como la aplicación iterativa de un conjunto de reglas de transformación a cierto tipo de objetos construidos a partir de las fórmulas proposicionales. Para ilustrar este comportamiento, consideremos el ejemplo de la figura 1, en el que se utiliza el método de los tableros semánticos para comprobar la satisfacibilidad de la fórmula $(p \rightarrow q) \wedge p$, construyendo un modelo de la misma (Ver [2] para una descripción más detallada). Inicialmente se considera un árbol con un único nodo etiquetado con la fórmula $(p \rightarrow q) \wedge p$. En un primer paso, la fórmula es expandida,

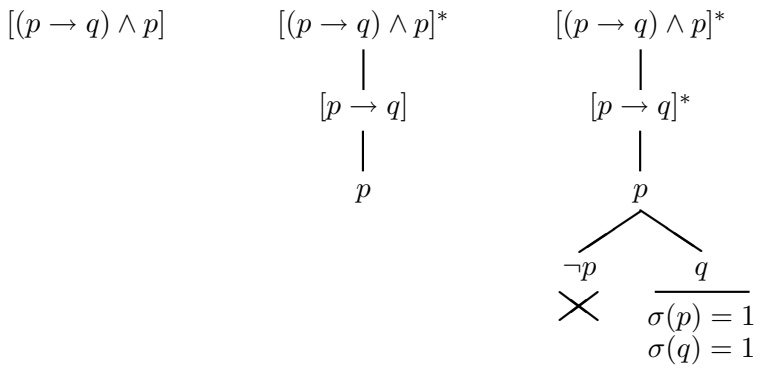

Figura 1. Un ejemplo del método de los tableros semánticos

obteniendo una extensión con las fórmulas $p \to q$ y p. En un segundo paso, la fórmula $p \to q$ es expandida obteniendo dos extensiones, una con la fórmula $\neg p$ y la otra con la fórmula q. La rama de la izquierda queda cerrada, pues contiene literales complementarios, y la rama de la derecha proporciona un modelo σ en el que los símbolos p y q son ciertos.

Si representamos una rama de un árbol como la secuencia de sus fórmulas, y un árbol como la secuencia de sus ramas, este ejemplo se puede describir como una secuencia de reglas de transformación que son aplicadas a un árbol (utilizamos la notación $\langle e_1, ..., e_k \rangle$ para representar secuencias finitas):

$$\langle\langle (p \to q) \wedge p \rangle\rangle \longmapsto \langle\langle p \to q, p \rangle\rangle \longmapsto \langle\langle p, \neg p \rangle, \langle p, q \rangle\rangle \longmapsto \langle\langle p, q \rangle\rangle$$

En cada paso se selecciona una rama, se le aplica una regla de expansión, obteniendo una lista de nuevas ramas, y se reemplaza la rama seleccionada por dicha lista. Como un caso particular, las ramas con elementos complementarios son reemplazadas por la lista vacía de ramas, es decir, son eliminadas. El proceso de transformación es aplicado hasta que se selecciona una rama sin fórmulas no literales ni elementos complementarios. A partir de esta rama se puede construir fácilmente un modelo de la fórmula original.

Este comportamiento puede ser observado en otros sistemas proposicionales de demostración, como el basado en el cálculo de secuentes (en el que los objetos sobre los que se trabaja son secuentes) o el método de Davis–Putnam (en el que los objetos sobre los que se trabaja son conjuntos de cláusulas). Estos sistemas no trabajan directamente con fórmulas, sino con unos objetos construidos a partir de ellas. Dichos objetos se modifican repetidamente usando reglas de expansión que reducen su complejidad. Finalmente, a partir de cierto tipo de objetos simples, se pueden obtener asignaciones que prueban la satisfacibilidad de la fórmula original. A estas asignaciones las llamamos *asignaciones distinguidas*. Si no se obtiene ninguno de estos objetos simples, entonces la fórmula original es insatisfacible. En las siguientes secciones se presentan las líneas generales de la formalización de este esquema de comportamiento genérico. Una versión más amplia puede encontrarse en [5,7].

2.1. Sistemas de transformación proposicionales

Un *sistema de transformación proposicional* es una tripleta $\mathcal{G} = \langle \mathcal{O}, \mathcal{R}, \mathcal{V} \rangle$, donde $\mathcal{O}$ es un conjunto de objetos de trabajo u *objetos proposicionales*, $\mathcal{R}$ es un conjunto de *reglas de expansión* y $\mathcal{V}$ determina las *asignaciones distinguidas* de los objetos de trabajo. Las reglas de expansión son de la forma $O \rightsquigarrow_{\mathcal{G}} L$, donde O es un objeto proposicional y L es una lista (posiblemente vacía) de objetos proposicionales, o de la forma $O \rightsquigarrow_{\mathcal{G}} \mathbf{t}$. Estas últimas sirven para caracterizar cierto tipo de objetos simples. Los elementos de $\mathcal{V}$ son pares de la forma (O, σ), donde σ es una asignación de valores de verdad sobre el conjunto de símbolos proposicionales. Para todo par (O, σ) de $\mathcal{V}$ diremos que σ es una asignación distinguida de O y lo notaremos $\sigma \models_{\mathcal{G}} O$.

En el caso del método de los tableros semánticos, los objetos de trabajo son las ramas, representadas como secuencias finitas de fórmulas. Las asignaciones distinguidas para una rama son aquellas en las que todas las fórmulas de la rama son ciertas. Las reglas de expansión establecen cómo se expande una rama de un tablero. Ejemplos de estas reglas se muestran a continuación:

$\langle \Gamma_1, G, \Gamma_2, \neg G, \Gamma_3 \rangle \rightsquigarrow \langle\rangle$	Ramas cerradas
$\langle \Gamma_1, F \wedge G, \Gamma_2 \rangle \rightsquigarrow \langle \langle \Gamma_1, F, G, \Gamma_2 \rangle \rangle$	Regla de la conjunción
$\langle \Gamma_1, F \rightarrow G, \Gamma_2 \rangle \rightsquigarrow \langle \langle \Gamma_1, \neg F, \Gamma_2 \rangle, \langle \Gamma_1, G, \Gamma_2 \rangle \rangle$	Regla de la disyunción
$\Gamma \rightsquigarrow_{\mathcal{T}} \mathbf{t}$	Γ no tiene fórmulas no literales ni elementos complementarios

Para completar la descripción del proceso de demostración proposicional asociado a un sistema de transformación proposicional, consideramos tres funciones: una *función de representación* i, una *regla de computación* r y una *función modelo* σ. El proceso, esquematizado en la figura 2, lo describimos a continuación.

El primer paso consiste en construir un objeto de trabajo inicial a partir de la fórmula F cuya satisfacibilidad se quiere comprobar. Para ello se utiliza la función de representación i. El proceso consiste en modificar una lista de objetos proposicionales mediante las reglas de expansión, por tanto, el punto de partida es la lista unitaria formada por el objeto inicial: $\langle i(F) \rangle$.

Partiendo de una lista cualquiera de objetos proposicionales $\langle O_1, \ldots, O_n \rangle$, se escoge un elemento cualquiera de dicha lista O_j, al que se aplica la regla de computación r. Una regla de computación no es más que un criterio que selecciona una regla de expansión aplicable a un objeto, devolviendo la lista de objetos resultante de aplicar dicha regla. De esta forma, si $r(O) = L$, entonces $O \rightsquigarrow L$ es un elemento de $\mathcal{R}$.

Si $r(O_j)$ es la lista $\langle O'_1, ..., O'_m \rangle$, con $m \geq 0$, entonces se reemplaza el objeto O_j por dicha lista, continuando el proceso con la nueva lista de objetos. Si $r(O_j) = \mathbf{t}$, entonces se el proceso termina y devuelve una asignación distinguida del objeto O_j, obtenida mediante la función modelo σ.

Obsérvese que en este proceso queda por determinar la forma en que se escoge un objeto para ser expandido. De esta forma se puede incorporar una heurística al proceso. El no considerar una función de elección concreta permite cierta generalidad tanto la formalización como la verificación de las propiedades, en el sentido de que son válidas para cualquier función de elección concreta.

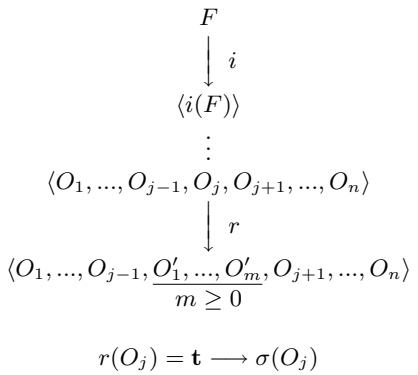

Figura 2.

En el caso del método de los tableros semánticos, la función de representación construye una lista unitaria formada por la fórmula inicial. La regla de computación establece un criterio de aplicación de las reglas de expansión, por ejemplo dar prioridad a las reglas que no producen bifurcación frente a las que sí lo hacen. La función modelo construye una asignación en la que todos los símbolos proposicionales de una rama son ciertos.

Para garantizar que el proceso descrito termina, se ha de verificar que la regla de computación (o de forma más general, las reglas de expansión) genere una lista de objetos cuya complejidad sea menor que la del objeto inicial. Para esto se utiliza una *función de medida* μ, que proporciona una medida de la complejidad de los objetos proposicionales. La propiedad que se exige a esta función es la siguiente:

$$\mathcal{P}_1 : O_i \in r(O) \Longrightarrow \mu(O_i) < \mu(O)$$

Una visión intuitiva de la terminación de este proceso es la proporcionada por Smullyan en [9]. En cada paso del proceso se dispone de una bolsa llena de bolas etiquetadas con números enteros positivos (la lista de objetos de trabajo), no importa cuántas bolas hay con la misma etiqueta ni qué números etiquetan las bolas. De esta bolsa se escoge una bola (el objeto O_j) y se reemplaza por otras bolas etiquetadas con números menores (la lista $\langle O'_1, \ldots, O'_m \rangle$). De esta forma, los números que etiquetan las bolas son cada vez más pequeños (aunque puede haber mayor cantidad) y por tanto el proceso termina.

Hemos demostrado en el sistema ACL2 que. para garantizar las propiedades de corrección y completitud del proceso, es suficiente con que la función de representación, la regla de computación y la función modelo verifiquen las siguientes propiedades:

$$\mathcal{P}_2 : F \in \mathbb{P}(\Sigma) \Longrightarrow (\sigma \models F \Longleftrightarrow \sigma \models_{\mathcal{G}} i(F))$$
$$\mathcal{P}_3 : O \in \mathcal{O} \wedge r(O) \neq \mathbf{t} \Longrightarrow (\sigma \models_{\mathcal{G}} O \Longleftrightarrow \exists O_i \in r(O), \sigma \models_{\mathcal{G}} O_i)$$
$$\mathcal{P}_4 : O \in \mathcal{O} \wedge r(O) = \mathbf{t} \Longrightarrow \sigma(O) \models_{\mathcal{G}} O$$

La propiedad $\mathcal{P}_2$ afirma que el conjunto de asignaciones que hacen cierta una fórmula F coincide con el conjunto de asignaciones distinguidas del objeto inicial $i(F)$. La propiedad $\mathcal{P}_3$ asegura que la existencia de una asignación distinguida para alguno de los objetos de trabajo considerados en cada paso del proceso, es una propiedad que se mantiene a lo largo del mismo. La propiedad $\mathcal{P}_4$ afirma que la asignación construida utilizando la función modelo sobre los objetos simples (para los que la regla de computación proporciona el valor $\mathbf{t}$), es una asignación distinguida de dichos objetos. En conclusión, la fórmula inicialmente considerada es satisfacible si y sólo si el proceso termina encontrando uno de estos objetos simples. Además, en caso de que la fórmula inicial sea satisfacible, al evaluar la función modelo sobre le objeto simple con el que termina el proceso, se obtiene una asignación que la hace cierta.

Veamos a continuación algunos detalles sobre la formalización de estos resultados en el sistema ACL2 y su utilización para obtener distintos sistemas proposicionales de demostración basados en transformación.

2.2. Formalización en la lógica de ACL2

El sistema ACL2 [4] tiene tres componentes: un lenguaje de programación (un subconjunto aplicativo de Common Lisp), una lógica que permite formular propiedades sobre las funciones escritas en este lenguaje y un demostrador automático en el que se pueden desarrollar demostraciones de estas propiedades.

ACL2 proporciona las funcionalidades necesarias para formalizar cierto tipo de resultados de segundo orden: se asume la existencia de funciones con ciertas propiedades y se realizan nuevas construcciones a partir de estas funciones, demostrando sus propiedades. Estos resultados se pueden reutilizar para cualquier conjunto concreto de funciones con las propiedades de las inicialmente asumidas.

De esta forma, hemos asumido la existencia de funciones que implementan las distintas componentes de un sistema de transformación proposicional y del proceso de demostración asociado, tal y como se muestra en la siguiente tabla:

Funciones ACL2	Interpretación
`gen-objeto`(O)	$O \in \mathcal{O}$
`gen-asignacion-distinguida`(σ, O)	$\sigma \models_{\mathcal{G}} O$
`gen-regla-computacion`(O)	$r(O)$
`gen-representacion`(F)	$i(F)$
`gen-medida`(O)	$\mu(O)$
`gen-modelo`(O)	$\sigma(O)$
`gen-seleccion`(L)	selecciona un elemento de L

También asumimos que estas funciones verifican las propiedades $\mathcal{P}_1$, $\mathcal{P}_2$, $\mathcal{P}_3$ y $\mathcal{P}_4$ expuestas en la sección anterior.

A continuación definimos la función `SAT-generico` que implementa el proceso de demostración proposicional descrito en la sección anterior[1]:

[1] Para presentar las funciones y sus propiedades, usamos la notación infija proporcionada por el sistema en lugar de la habitual de Common Lisp.

```
OBJ-SAT-generico(O-lst) :=
   if endp(O-lst) then nil
   else let* O be gen-seleccion(O-lst),
             resto be elimina-una(gen-seleccion(O-lst), O-lst),
             expansion be gen-regla-computacion(O)
             in
        if expansion = t then list(O)
        else OBJ-SAT-generico(expansion @ resto)

SAT-generico(F) :=
   OBJ-SAT-generico(list(gen-representacion(F)))

MOD-generico(F) :=
   gen-modelo(car(SAT-generico(F)))
```

Para estas funciones se han demostrado en ACL2 los siguientes resultados de corrección y completitud (en [5] se puede encontrar una descripción más detallada sobre las pruebas realizadas):

PROPIEDAD `correccion-SAT-generico`:
`es-proposicional`$(F) \wedge$ `SAT-generico`$(F) \rightarrow$ `modelo`(`MOD-generico`(F), F)

PROPIEDAD `completitud-SAT-generico`:
`es-proposicional`$(F) \wedge$ `modelo`$(\sigma, F) \rightarrow$ `SAT-generico`(F)

Para facilitar la reutilización de estos resultados para los distintos casos concretos hemos desarrollado una herramienta [5,6] que construye de forma automática las versiones de las funciones `OBJ-SAT-generico`, `SAT-generico` y `MOD-generico` para un caso concreto, y demuestra sus propiedades de corrección y completitud. Basta establecer la correspondencia entre las funciones concretas y las funciones asumidas en la formalización, presentadas al comienzo de esta sección.

De esta forma hemos desarrollado sistemas de demostración proposicional basados en el método de los tableros semánticos, el cálculo de secuentes y el procedimiento de Davis–Putnam, garantizando sus propiedades de corrección y completitud. Estos procedimientos son ejecutables en cualquier sistema Common Lisp junto con un pequeño conjunto de funciones adicionales incorporadas por el sistema ACL2.

En la figura 3 presentamos los tiempos (en segundos) invertidos por estos procedimientos para demostrar la satisfacibilidad de una versión proposicional del problema de las N-reinas. El procedimiento de Davis–Putnam trabaja con cláusulas proposicionales y, por tanto, es necesario un proceso de transformación de fórmulas a conjunto de cláusulas. En este caso no se incluyen los tiempos de transformación. Todas las pruebas han sido realizadas en un sistema con dos procesadores Pentium III a 800 MHz.

N	Tableros	Secuentes	Davis–Putnam
2	0.010	0.000	0.000
3	0.060	0.020	0.010
4	0.530	0.180	0.040
5	2.370	0.820	0.140
6	212.070	72.600	0.250
7	750.540	255.640	0.570

Figura 3. Tiempos de evaluación para el problema de las N-reinas

soporte	*usables*	C	Resolventes
$\langle\langle p,q\rangle,\langle p,\neg q\rangle,\langle\neg p,q\rangle\rangle$	$\langle\rangle$	$\langle p,q\rangle$	
$\langle\langle p,\neg q\rangle,\langle\neg p,q\rangle\rangle$	$\langle\langle p,q\rangle\rangle$	$\langle p,\neg q\rangle$	$\langle p\rangle$
$\langle\langle p\rangle,\langle\neg p,q\rangle\rangle$	$\langle\langle p,\neg q\rangle,\langle p,q\rangle\rangle$	$\langle p\rangle$	
$\langle\langle\neg p,q\rangle\rangle$	$\langle\langle p\rangle,\langle p,\neg q\rangle,\langle p,q\rangle\rangle$	$\langle\neg p,q\rangle$	$\langle q\rangle$
$\langle\langle q\rangle\rangle$	$\langle\langle\neg p,q\rangle,\langle p\rangle,\langle p,\neg q\rangle,\langle p,q\rangle\rangle$	$\langle q\rangle$	$\langle p\rangle$
$\langle\rangle$	$\langle\langle q\rangle,\langle\neg p,q\rangle,\langle p\rangle,\langle p,\neg q\rangle,\langle p,q\rangle\rangle$		

Figura 4. Un ejemplo de saturación por resolución binaria

3. Sistemas proposicionales basados en saturación

Los sistemas proposicionales de demostración basados en resolución no tienen el mismo comportamiento que el tratado en la sección anterior. Estos sistemas se basan en la aplicación reiterada de una operación de resolución elemental sobre las cláusulas de un conjunto. El proceso termina cuando se genera la cláusula vacía o se *satura* el conjunto, es decir, el resultado de aplicar la operación de resolución elemental sobre cualesquiera cláusulas del conjunto es una cláusula que ya pertenece a dicho conjunto (Ver [2] para una descripción más detallada). En estos sistemas, el resultado de aplicar la operación de resolución elemental no siempre es de menor complejidad que las cláusulas a las que se aplica dicha operación. Además, después de cada aplicación de la operación de resolución elemental, el conjunto de cláusulas se ve incrementado con el resultado de dicha operación (si es que no pertenecía ya al conjunto de cláusulas original). De esta forma, no se puede considerar un objeto de trabajo que sea transformado a lo largo del proceso, dando lugar a objetos de trabajo más simples.

Una forma de realizar el proceso de saturación se muestra en el ejemplo de la figura 4 para la resolución binaria. A lo largo del proceso se consideran dos conjuntos: *soporte* y *usables*. El valor inicial del *soporte* es el conjunto de cláusulas para las que se quiere realizar una saturación por resolución. El conjunto *usables* es un acumulador donde se obtendrá el conjunto saturado por resolución al final del proceso. En cada paso se toma un elemento C del conjunto *soporte* y se calculan todas las resolventes entre C y los elementos del conjunto *usables*. Estas resolventes se añaden al conjunto *soporte* y la cláusula C al conjunto *usables*. El proceso termina cuando se obtiene al cláusula vacía al realizar una operación de resolución o cuando el conjunto *soporte* queda vacío. En el primer caso se puede afirmar que el conjunto de cláusulas original es

insatisfacible. En el segundo caso, el conjunto *usables* es un conjunto saturado por resolución.

El proceso descrito se puede aplicar a las distintas variantes de resolución: binaria, positiva, negativa, semántica, soporte, etc. Para ello basta considerar la variante correspondiente como operación de resolución elemental. En las siguientes secciones presentamos las líneas generales de la formalización de un proceso genérico de saturación por resolución, en el que, para poder aplicar la operación de resolución elemental, se exige que las cláusulas verifiquen cierta condición no especificada. De esta forma, al considerar distintas condiciones, se obtienen distintas variantes de resolución. Una versión ampliada de este trabajo se puede encontrar en [5].

3.1. Saturación por resolución condicionada

En lo sucesivo consideraremos un predicado binario conmutativo P. La *resolvente condicionada* de dos cláusulas C_1 y C_2 con respecto a un literal L, tales que L aparece en C_1, $\overline{L}$ aparece en C_2 y $P(C_1, C_2)$, es la cláusula $(C_1 - \{L\}) \cup (C_2 - \{\overline{L}\})$. Diremos que P es la condición de resolución. Llamamos *saturación por resolución condicionada* al proceso de saturación descrito al comienzo de esta sección, en el que la operación de resolución elemental considerada es la resolución condicionada.

Como se ha indicado antes, considerando distintas condiciones de resolución se obtienen variantes del proceso de resolución. La resolución binaria se obtiene al considerar una condición de resolución que sea siempre cierta. La resolución positiva (resp. negativa) se obtiene al considerar como condición de resolución el hecho de que una de las dos cláusulas C_1 o C_2 sea positiva (resp. negativa). La resolución semántica se obtiene al considerar como condición de resolución el hecho de que una de las dos cláusulas C_1 o C_2 sea cierta en una asignación prefijada, y la otra falsa. Al considerar el proceso de saturación por resolución condicionada, se obtiene cierta generalidad en la formalización, de forma que los resultados obtenidos son válidos para cualquier condición de resolución y, por tanto, para distintas variantes de resolución.

El proceso de saturación por resolución condicionada es correcto, es decir, si se obtiene la cláusula vacía a lo largo del proceso de resolución, entonces el conjunto original de cláusulas es insatisfacible. Esto se debe al hecho de que una resolvente condicionada de dos cláusulas es satisfacible si y sólo si alguna de dichas cláusulas también lo es.

Una forma de demostrar la completitud del proceso de saturación por resolución condicionada es la desarrollada por Bezem en [1]. Una condición P es segura si existe una asignación σ_P tal que para todo par de cláusulas C_1 y C_2, si σ_P es modelo de C_1 y no lo es de C_2, entonces C_1 y C_2 cumplen la condición de resolución. Si P es una condición segura entonces el proceso de saturación por resolución condicionada es completo. Es decir, si el proceso termina sin generar la cláusula vacía, entonces el conjunto original es satisfacible. Además, la prueba de Bezem de este hecho proporciona una asignación que es modelo del conjunto de cláusulas inicial.

A continuación presentamos algunos detalles de la formalización de estos resultados en el sistema ACL2.

3.2. Formalización en la lógica de ACL2

Al igual que en la formalización presentada en la sección 2, asumimos la existencia de dos funciones, `condicion` y `asignacion`, verificando las siguientes propiedades:

1. `condicion` es un predicado binario conmutativo:

$$\texttt{condicion}(C_1, C_2) \Longleftrightarrow \texttt{condicion}(C_2, C_1)$$

2. `asignacion` es una función constante que verifica la siguiente propiedad con respecto a `condicion`:

$$\texttt{(asignacion)} \models C_1 \text{ y } \texttt{(asignacion)} \not\models C_1 \Longrightarrow \texttt{condicion}(C_1, C_2)$$

La función `condicion` hace el papel de la condición de resolución P, y la constante `(asignacion)` el papel de la asignación σ_P.

A continuación definimos la función `saturacion-resolucion` que implementa el proceso de saturación por resolución condicionada siguiendo la descripción dada al comienzo de esta sección.

```
saturacion-resolucion(S) :=
   resolucion-cond-saturacion(S,nil)

resolucion-cond-saturacion(S1,S2) :=
   if endp(S1) then S2
   else let resolventes be
                resolucion-cond-clausula-conjunto(car(S1),S2,nil,S1@S2)
        in
        if contiene-clausula-vacia(resolventes) then t
        else resolucion-cond-saturacion(resolventes@cdr(S1),
                                        cons(car(S1),S2))
```

La función `resolucion-cond-clausula-conjunto` calcula las nuevas resolventes entre la cláusula que se le pasa como primer argumento y cualquier cláusula perteneciente al conjunto que se pasa como segundo argumento. El tercer argumento se utiliza como acumulador del conjunto de cláusulas construido y el cuarto para comprobar que efectivamente se están generando cláusulas nuevas.

Obsérvese que, cuando en el proceso de saturación por resolución se obtiene la cláusula vacía, se devuelve el valor **t**, mientras que en otro caso se devuelve un conjunto de cláusulas.

Para estas funciones hemos demostrado en ACL2 los siguientes resultados (en [5] se puede encontrar una descripción más detallada sobre las pruebas realizadas):

Tamaño del problema	5	6	7	8	9	10
Resolución binaria	8.720	250.380	—	—	—	—
Resolución positiva	0.000	0.000	0.010	0.000	0.010	0.000
Resolución negativa	0.040	0.220	0.420	0.810	2.770	3.810

Figura 5. Tiempos de evaluación para las fórmulas de Plaisted

PROPIEDAD `correccion-saturacion-resolucion`:
`es-forma-clausal`(S) $\wedge$ `saturacion-resolucion`(S)
$\rightarrow$ `not`(`modelo-forma-clausal`(σ,S))

PROPIEDAD `completitud-saturacion-resolucion`:
`es-forma-clausal`(S) $\wedge$
`not`(`contiene-clausula-vacia`(S)) $\wedge$
`not`(`saturacion-resolucion`(S))
$\rightarrow$ `modelo-forma-clausal`(δ,S)

En el resultado de completitud, la asignación δ es el modelo que se construye en la demostración de Bezem de la completitud de la resolución condicionada.

Para obtener las distintas variantes del proceso de resolución, utilizamos una herramienta [5,6] que construye de forma automática las versiones de la función `saturacion-resolucion` para una condición de resolución concreta, y demuestra sus propiedades de corrección y completitud. De esta forma hemos desarrollado sistemas de demostración proposicional basados en resolución binaria, positiva, negativa, semántica y soporte. Estos procedimientos son ejecutables en cualquier sistema Common Lisp junto con un pequeño conjunto de funciones adicionales incorporadas por el sistema ACL2.

En la figura 5 presentamos los tiempos (en segundos) invertidos por los procedimientos obtenidos para la resolución binaria, positiva y negativa en algunos de los problemas propuestos por Plaisted en [8]. De nuevo, las pruebas han sido realizadas en un sistema con dos procesadores Pentium III a 800 MHz.

4. Conclusiones y trabajo futuro

Hemos presentado una aplicación del demostrador ACL2 para razonar sobre sistemas proposicionales de demostración. Hemos formalizado dos esquemas genéricos de sistemas proposicionales de demostración, el primero está basado en reglas de transformación y el segundo en un proceso de saturación. Hemos demostrado las propiedades de terminación, corrección y completitud de los procedimientos de demostración desarrollados en cada uno de los dos esquemas genéricos. Finalmente, hemos obtenido procedimientos ejecutables y verificados para demostrar la satisfacibilidad de una fórmula proposicional como casos particulares de los esquemas genéricos. Este último proceso ha sido realizado de una forma automática.

La metodología que hemos seguido resulta muy adecuada para la verificación automática. Primero se razona sobre un procedimiento genérico, lo cual nos permite concentrarnos en los aspectos esenciales del proceso, facilitando la tarea de verificación. Mediante un proceso automático de instanciación se obtienen versiones concretas verificadas del procedimiento genérico, sin necesidad de repetir el esfuerzo de prueba. Un último paso en esta metodología es el refinamiento. Podemos definir procedimientos más eficientes y demostrar sus propiedades probando que son equivalentes a los iniciales. Esta es una línea de trabajo futuro.

En este sentido, hemos implementado en ACL2 el método de Davis–Putnam basado en [10], dando lugar a un procedimiento más eficiente que el presentado aquí. La figura 6 presenta los tiempos empleados por este procedimiento mejorado para el problema de las N-reinas. Aún no hemos abordado la verificación de este procedimiento, aunque esta tarea será más fácil si demostramos previamente propiedades de equivalencia entre la versión mejorada y la versión verificada obtenida a partir del esquema genérico.

N	Davis–Putnam mejorado	N	Davis–Putnam mejorado
6	0.010	10	0.180
7	0.010	11	0.160
8	0.020	12	0.330
9	0.020	13	0.300

Figura 6. Tiempos de evaluación para el problema de las N-reinas

Referencias

1. Bezem, M. Completeness of resolution revisited. *Theoretical Computer Science*, vol. 74, no. 2, pp. 227–237, 1990.
2. Fitting, M.C. *First–Order Logic and Automated Theorem Proving.* Springer–Verlag, New York, 1990.
3. Freeman, J.W. *Improvements to Propositional Satisfiability Search Algorithms* Tesis doctoral, 1995.
4. Kaufmann, M., Manolios, P. y Moore, J S. *Computer–Aided Reasoning: An Approach.* Kluwer Academic Publishers, 2000.
5. Martín–Mateos, F.J. *Teoría computacional (en ACL2) sobre cálculos proposicionales.* Tesis doctoral, Septiembre 2002.
6. Martín–Mateos, F.J., Alonso, J.A., Hidalgo, M.J. y Ruiz–Reina, J.L. A Generic Instantiation Tool and a Case Study: A Generic Multiset Theory. Third International Workshop on the ACL2 Theorem Prover, Grenoble, 2002.
7. Martín–Mateos, F.J., Alonso, J.A., Hidalgo, M.J. y Ruiz–Reina, J.L. Verification in ACL2 of a generic framework to synthesize SAT–provers. LOPSTR–2002.
8. Plaisted, D.A. A simplified problem reduction format. *Artificial Intelligence*, vol. 18, pp. 227–261, 1982.
9. Smullyan, R.M. Trees and ball games. *Annals of the New York Academy of Sciences*, no. 321, pp. 86–90, 1979.
10. Zhang, H. y Stickel, M.E. Implementing the Davis–Putnam method *Journal of Automated Reasoning*, vol. 24(1–2), pp. 277–296, 2000.