Congr. Int. en Ing. Electrónica. Mem. ELECTRO, Vol. 39, pp. 275-281, Oct 2017, Chihuahua, Chih. México
http://electro.itchihuahua.edu.mx/memorias_electro/MemoriaElectro2017.zip
ISSN 1405-2172

El Rol de los Razonadores en Robótica y Automatización

García Mata, Carmen Leticia; Márquez Gutiérrez, Pedro Rafael; Baray Arana, Rogelio
Instituto Tecnológico de Chihuahua
División de estudios de Posgrado e Investigación del Instituto Tecnológico de Chihuahua
Ave. Tecnológico #2909, Chihuahua, Chihuahua. México, CP 31310
(614) 201-2000, Fax (614) 413-5187
e-mail: {clgarcia, pmarquez, rbaray}@itchihuahua.edu.mx

RESUMEN

En este artículo se presenta un resumen sobre el área de razonamiento automatizado. Esta investigación no pretende ser exhaustiva, pero si es su objetivo ofrecer un visión general sobre la evolución del área y en particular se hace énfasis, en cuáles aspectos teóricos del problema deben ser considerados al momento de seleccionar la metodología más adecuada para realizar el modelado y solución de cierto problema. Adicionalmente, en este trabajo se analizan también algunas características destacadas de razonadores automatizados de vanguardia, tanto de los basados en lógica clásica como de los basados en lógica no-monotónica.

Palabras clave

Razonamiento automatizado; SAT solvers; CP/CSP solvers; ASP solvers; SMT solvers.

1. INTRODUCCIÓN

Existen diversos sistemas automatizados que dadas las características del entorno dinámico en el que deben operar, necesariamente deben ser capaces de ajustar su comportamiento a los cambios que ocurren en su entorno. Un área que tradicionalmente ha lidiado con este tipo de problemas es la de robótica, donde regularmente el robot razona y ejecuta determinadas acciones basado en el conocimiento que tiene del mundo. Sin embargo, no es la única aplicación en la que se requiere que el sistema decida por sí mismo para satisfacer los objetivos del diseño. Actualmente, cada día se incrementa el número de aplicaciones que requieren de agentes inteligentes que operen robustamente en ambientes impredecibles, rápidamente cambiantes y donde existe una significante posibilidad de que las acciones fallen. Para la solución de este tipo de problemas, se requiere de metodologías y técnicas que permitan analizar, estructurar y crear bases de conocimiento y sistemas de razonamiento automático [31].

Otros ejemplos de aplicaciones que recaen fuertmente en el conocimiento son: los sistemas avanzados de manufactura, sistemas para la toma de decisiones, automóviles y drones guiados automáticamente, configuración de preferencias de clientes tales como Amazon y Netflix, diagnóstico médico asistido y muchas otras aplicaciones que ya se encuentran vigentes o que lo serán en un futuro cercano.

Como una respuesta a esta necesidad, recientemente se han desarrollado un conjunto de poderosos métodos y técnicas del área de representación del conocimiento y razonamiento (*Knowledge Representation and Reasoning*, KRR) [19] y [8]. KRR es una subárea de inteligencia artificial (AI, *Artificial Intelligence*) y su objetivo es la comprensión y automatización de varios aspectos de razonamiento. Estas tecnologías están basadas principalmente en lógica proposicional (*Propositional Logic*, LP) [35], [22] y [3]. No obstante, existen también otros razonadores basados en otro tipo de lógicas como las lógicas no-monotónicas [14], [34] y [39].

Dada la creciente necesidad de dispositivos basados en el conocimiento, este artículo se enfoca en analizar los aspectos teóricos y prácticos que engloban el desarrollo de los razonadores automatizados modernos y cómo tomar en consideración las características del problema para decidir con cuál enfoque y con cuál máquina de inferencias resulta más adecuada resolver el problema en cuestión.

2. SATISFACIBILIDAD, COMPLETEZ, VALIDEZ Y CLASES DE COMPLEJIDAD

Los primeros estudios formales para el diseño de máquinas de razonamiento automatizado pueden ser trazados al planteamiento realizado por Hillbert en 1928 del problema **Entscheindungs** [20]. Este problema básicamente consiste en: "*.... determinar si existe un procedimiento que permita decidir si: dada cualquier expresión lógica, ésta es válida o satisfacible utilizando un número finito de operaciones*..."

Turing y Church demostraron por separado que no existe un procedimiento general para verificar la validez de fórmulas planteadas con Lógica de Primer Orden (First Order Logic, FOL). Turing hizo su demostración en (1936-37), reduciendo el problema de decidibilidad para teorías lógicas, al problema de la parada (*halting problem*) para máquinas abstractas [41]. De esta forma Turing modeló el proceso de computabilidad, y probó que éste es indecidible. Mientras tanto, Church (1936) resolvió el problema planteado por Hillbert mediante Cálculo Lambda y demostró que FOL y varias de sus subclases son indecidibles [5]. No obstante, existen procedimientos de prueba que permiten verificar si una fórmula es válida, cuando efectivamente lo es. Pero no ocurre lo mismo para las fórmulas inválidas, ya que en general, estos procedimientos nunca terminan.

Después de los resultados obtenidos por Turing y Church, todo parecía indicar que la creación de sistemas de razonamiento automatizado era inviable. La situación pudo ser remontada considerando el problema desde un punto de vista diferente. Si se restringe el conjunto de modelos de alguna forma (considerando solo áquellos que satisfagan una teoría específica), es posible obtener muchos fragmentos de lógica que

son decidibles. Algunos fragmentos decidibles de FOL son: la lógica proposicional (Propositional Logic, PL), la lógica monádica de predicados (FOL restringida a símbolos unarios de predicados y sin símbolos de funciones), la lógica de dos variables, la clase Bernays-Shönfinkel y la lógica de descripciones.

2.1 Complejidad y Explosión Combinatoria

Muchos problemas de diferentes dominios pueden ser representados como problemas lógicos, y teóricamente pueden ser resueltos mediante un algoritmo general, el cual hace una búsqueda exhaustiva a través de todas las posibles combinaciones de las variables que conforman el espacio de solución. Esta clase de problemas se conoce como problemas combinatorios. Algunos problemas combinatorios típicos son: búsqueda de trayectorias más cortas/baratas, ruteo de paquetes de datos en internet, verificación de software, predicción de estructura de proteínas, planificación y calendarización de recursos.

El diseño de algoritmos eficientes para la solución de problemas combinatorios es una de las tareas más retadoras en el área de computación. Desde que se sentó el área de teoría de la computación, muchas investigaciones han dedicado esfuerzos importantes para formular problemas prácticos como generalizaciones [23]. Si bien, el concepto de *complejidad computacional*, aún no había sido desarrollado en esa época, sí se conocía que había una posibilidad de "explosión combinatoria", extrema ineficiencia y nula aplicación práctica de los algoritmos de búsqueda de fuerza bruta en la solución de problemas del tipo combinatorio, puesto que el número de posibilidades crece inmensamente conforme el número de variables se incrementa. Debido a estas características, la mayoría de los problemas combinatorios son intratables, a menos que el algoritmo utilice algún tipo de sutileza o conocimiento sobre el problema para reducir la búsqueda [40].

Teniendo como principal objetivo la solución de esta problemática, surgió la ciencia de Teoría de la Complejidad, cuyo objetivo fundamental es clasificar los problemas de acuerdo a su tratabilidad, es decir, si el problema tiene solución para instancias grandes en un tiempo que sea práctico para la aplicación [25]. Sin embargo, existen distintos grados de tratabilidad, los cuales varían de acuerdo al modelo computacional, a los recursos disponibles y a las variantes de las estructuras de datos. Muchos problemas no son tratables, y en la práctica sólo es posible resolverlos para instancias pequeñas. Aún más, todas las funciones exponenciales y factoriales son no tratables debido a que el número de posibles combinaciones para estas funciones es de muchos órdenes de magnitud mayor que el de las funciones polinomiales. La solución al problema de tratabilidad no es una solución trivial, ya que el uso de hardware más poderoso no es suficiente para su solución, sino que estos problemas deben enfocarse de distinta manera.

Aunque existen algoritmos más eficientes que los generales, todos los algoritmos conocidos para PL tienen para el peor caso, una complejidad exponencial con respecto al tamaño de la entrada. Esto no es extraño ya que LP tiene complejidad co-**NP**-completo [9].

2.2.1 El Teorema de Cook y sus Implicaciones

El Teorema de Cook tiene que ver con el problema de satisfacibilidad proposicional (SAT), el cual consiste en: *dado un conjunto de cláusulas sobre un conjunto U finito, encontrar una asignación para U que satisfaga todas las cláusulas* [4][26].

Las cláusulas deben estar en la Forma Normal Conjuntiva (*Conjuctive Normal Form*, CNF); Una cláusula está en su forma conjuntiva si y solo si las cláusulas se relacionan con operadores AND y los términos con operadores OR, por ejemplo:

$(\neg x_1 \vee \neg x_3 \vee \neg x_4) \wedge (x_2 \vee x_3 \vee \neg x_4) \wedge (x_1 \vee \neg x_2 \vee x_4) \wedge (x_1 \vee x_3 \vee x_4) \wedge (\neg x_1 \vee x_2 \vee \neg x_3)$.

En el trabajo seminal de Cook [4], éste definió formalmente las clases de los lenguajes **P** y **NP**, demostrando que el problema de satisfacibilidad (SAT) es **NP**-completo (**NP**-*complete*). Además propuso y comprobó que cualquier problema combinatorio que pueda ser reducido a SAT, es también un problema **NP**-completo. Fué Cook también quién propuso uno de los más incitantes problemas del área de computación: ¿Es **P** = **NP**? Este problema sigue abierto hasta la fecha.

Este teorema implica que si se encuentra un algoritmo polinomial para SAT, usando la poli-transformación de Cook, es posible resolver todos los problemas de la clase **NP** en tiempo polinomial, **y consecuentemente se probaría también que P=NP**.

Además, si se encuentra una poli-transformación desde SAT a otro problema *Z*, entonces *Z* también es un problema **NP**-completo, o viceversa. Adicionalmente, si se encuentra un algoritmo polinomial para *Z*, usando una poli-transformación de SAT a *Z*, sería posible resolver cualquier instancia de SAT en tiempo polinomial y por lo tanto también sería posible resolver todos los problemas de la clase **NP**.

Casi inmediatamente después del trabajo seminal de Cook, Karp [24] hizo otra importante aportación, mostrando que las versiones de decisión de muchos problemas de optimización muy conocidos son **NP**-completos, y que la única forma de encontrar algoritmos eficientes para resolver estos problemas es probar si es cierto que **P**=**NP** [12] y [43]. Estos primeros resultados en el área después conocida como Teoría de la Complejidad desalentaron la búsqueda de algoritmos eficientes, y gran parte de la investigación se enfocó a la búsqueda de algoritmos aproximados lo suficientemente rápidos para la solución de problemas prácticos.

3.RAZONADORES AUTOMATIZADOS MODERNOS

En unas cuantas décadas la investigación sobre razonamiento automatizado ha pasado a ser un área de investigación bien establecida.

Si bien el primer método para la solución del problema de Razonadores Automáticos de Teoremas fué planteado desde los años 30 del siglo pasado por Herbrand, no fué hasta el desarrollo de las primeras computadoras en los años 60 cuando Gilmore implementó computacionalmente dicho método [15]. Básicamente, el método de Herbrand consiste en encontrar una interpretación que pueda falsificar una fórmula dada. Es decir,

en lugar de probar que una fórmula es válida, el algoritmo prueba que la negación de la fórmula es inconsistente.

Aunque la implementación de Gilmore podía resolver fácilmente fórmulas simples, no ocurría lo mismo para la mayoría de otras fórmulas FOL. Apenas unos meses después de la publicación de Gilmore, Davis and Putnam publicaron un artículo donde hacían mejoras importantes al algoritmo de Gilmore [6]. Una de las aportaciones más importantes de Davis y Putnam fué la de utilizar una fórmula en su forma standard Skolem, donde las constantes y las funciones son usadas para reemplazar las variables existenciales Si bien la mejora de Davis y Putnam permitió resolver más fórmulas FOL, aún no era lo suficientemente adecuado para resolver en un tiempo adecuado otras fórmulas FOL. La principal razón por la cual el teorema de Herbrand no resulta práctico es debido a que requiere la generación de los conjuntos de todas las instancias de las cláusulas, lo que lleva a una explosión combinatoria.

Una mejora importante al teorema de Herbrand, fué el principio de resolución propuesto por Robinson [36]. Este método puede ser aplicado directamente a cualquier conjunto *S* de cláusulas (no necesariamente instanciadas) para obtener la insatisfaciblidad de *S*. El método en sí puede ser visto como una regla de inferencia que permite obtener nuevas reglas a partir de *S*. Una ventaja importante de este método de inferencia es que permite siempre confirmar o refutar una sentencia.

El **método de resolución** de Robinson, fue el primer método correcto **(sound**) y **completo (complete)** que mostró éxito como demostrador de teoremas, y marcó un hito en esta área, ya que permitió encontrar soluciones para problemas prácticos.

3.1 Razonadores SAT

Si bien, como ya se mencionó en párrafos anteriores, los primeros algoritmos para solucionar SAT fueron construidos desde los 60's, por Robinson y otros investigadores, estos algoritmos aún eran imprácticos para la solución de problemas reales. Un paso importante en esa dirección fué la mejora que Loveland and Logemann [7], realizaron sobre el algoritmo que previamente habían desarrollado Davis and Putnam [6]. Este algoritmo mejorado, es reconocido ampliamente desde entonces como el algoritmo DPLL.

A partir de los 90's, el método DPLL fue mejorado significativamente cuando se desarrolló un nuevo algoritmo que constituye la base de las máquinas de inferencias SAT modernas. Este algoritmo se llama “aprendizaje de cláusulas manejadas por conflicto” (*Conflict Drive Clause Learning*, CDCL) [29]. CDCL está basado en las heurísticas VSIDS, análisis de conflicto y dos estructuras más eficientes conocidas como “listas de literales dos veces observadas”.

El algoritmo CDCL a su vez, ha sido mejorado en las últimas décadas, de tal forma que las máquinas de inferencias SAT basadas en CDCL, ahora son capaces de resolver en unos pocos segundos problemas con cientos o miles de variables y millones de cláusulas, convirtiéndose así en los primeros razonadores en obtener buenos resultados en la resolución de problemas complejos que se presentan en distintas áreas, pero más especialmente han sido ampliamente aplicados en la industria manufacturera [22]. Las aplicaciones más importantes de SAT están relacionadas con problemas industriales en el diseño automatizado electrónico (*Electronic Design Automated*, EDA) [29].

Otra área de aplicación importante del SAT es la de Verificación de Software, y recientemente se empieza a explorar la aplicabilidad del SAT a otros problemas combinatorios como la generación de horarios (*Timetabling*), la planificación con un solo recurso y restricciones relajadas entre trabajos (*cumulative scheduling*) y la planificación de eventos deportivos (*Sports Scheduling*) [1].

Adicionalmente, los resultados obtenidos en la última competencia denominada “*SAT Competition 2016*”, nos hablan del enorme interés que se tiene en estos razonadores, puesto que participaron decenas de ellos. Los razonadores ganadores de los tres primeros lugares de las diferentes categorías son: Categoría *Random Track* (Dimetheus, CSCCSat, DCCAlm); Categoría *Incremental Track* (CryptoMiniSat, Glucose, Riss); Categoría *Parallel Track* (Treengeling, Plingeling, CryptoMiniSat); Categoría *Agile Track* (Riss, TB Glucose, CHBR Glucose); Categoría *NoLimit Trac* (BreakIDCOMiniSatPS, Lingeling, abcdSAT); Categoría *Main Track* (MapleCOMSPS, Riss, Lingeling).

3.2. Razonadores CSP

La idea básica de la programación con restricciones (*Constraint Programming*, CP) es definir un problema como un conjunto de restricciones representadas a través de un conjunto de variables. Se define un dominio de valores para cada variable y también se establecen las relaciones entre los subconjuntos de estas variables.

Por ejemplo, en un problema de planificación de tareas para un sistema de manufactura, las variables de decisión son los tamaños de los lotes de cada trabajo y los tiempos de inicio de los trabajos. En tanto, las restricciones posibles son: la capacidad de los recursos (por ejemplo, un trabajo se procesa a la vez en cada máquina), y la forma en que los trabajos son procesados (los trabajos no deben ser procesados en un orden arbitrario, sino que deben cumplir con las especificaciones de las órdenes).

Recientemente, CSP ha sido extendido para diferentes propósitos. Algunas de estas extensiones fueron diseñadas para la solución de problemas de optimización con uno ó múltiples criterios. Los razonadores de CSP (*CSP solver*s), hacen la búsqueda de la solución de una forma local o sistemática. Los razonadores basados en búsqueda sistemática usan resolución de búsqueda hacia atrás (*backtracking*), ramificación y acotamiento (*branch and bound*), o una combinación de búsqueda e inferencia. La inferencia es usada para angostar el espacio de búsqueda. Sin embargo, muchos problemas combinatorios no son solubles en tiempo polinomial y por tanto es impráctico realizar máquinas que hacen búsqueda sistemática. Es por esta causa, que aún es común el uso de razonadores basados en búsqueda no-sistemática ó búsqueda local, aún y cuando las soluciones devueltas no siempre son óptimas [17].

Se han reportado numerosos casos de éxito en la resolución de problemas de planificación/replanificación para sistemas de manufactura mediante CSP/CP, vea por ejemplo [27] y [44].

Otro caso, donde CSP/CP fue aplicada para resolver un problema de planificación de tareas en SMS, está dado en [33].

3.3 Lógica no-monotónica y ASP

Existen además otros métodos generales para la solución de problemas combinatorios basados en lógicas no-clásicas. Los más destacados en los últimos años son los basados en lógica no-monotónica.

A pesar de los enormes logros obtenidos los últimos años para razonadores automatizados basados en lógica clásica o monotónica, ésta no es lo suficientemente expresiva ni práctica para lidiar con conocimiento incompleto y contradictorio [18].

En cambio, la lógica no monotónica ha sido creada para realizar inferencias basadas en conocimiento incompleto, de forma similar al razonamiento humano en la toma de decisiones. Por ejemplo, para hacer un plan cualquiera y evitar considerar todo tipo de posibilidades, utilizamos una estrategia del "normalmente". Supongamos el caso en que nos dirigimos a nuestras actividades diarias. Habitualmente seleccionamos la trayectoria que nos lleve al destino en el menor tiempo posible, basándose en un conocimiento del "normalmente": "Normalmente tomo la avenida *X* y a la altura de *Y* volteo a la izquierda, tomando por la avenida *Z*, y desde *Z* conduzco directamente hacia el norte por 10 minutos hasta arribar al sitio deseado".

Así, simplemente cuando se presenta un evento imprevisto, reacomodamos los planes para resolver esta eventualidad. Esta estrategia es muy conveniente ya que, de otro modo, resulta imposible tener un conocimiento completo de todos los imponderables que se pueden presentar en cada situación.

Precisamente, esta forma de razonamiento basada en el "normalmente" es la que ha sido implementada en los razonadores basados en lógica no monotónica.

Actualmente, las soluciones semánticas más exitosas, en las cuáles se basan la mayoría de las máquinas de inferencia no monotónica son la semántica del modelo estable (*Stable Model Semantic*, SME) [14] y la semántica bien fundada (*Well-Founded Semantics*, WFS) [42]. Ambas semánticas tienen numerosas extensiones.

La Semántica del Modelo Estable se basa en la idea de aceptar múltiples modelos mínimos (*answer sets*) como una descripción del significado de un programa, a diferencia de la lógica clásica, en la cual se acepta un único modelo.

Esto ha llevado al desarrollo de un novedoso paradigma de la programación, comúnmente referido como *Answer Set Programming* (ASP) [28]. ASP es un paradigma de computación, en el cual las teorías lógicas (cláusulas de Horn con negación por default) sirven como especificaciones, y las soluciones de los problemas son representadas por una colección de modelos. ASP es un lenguaje completamente declarativo y ha sido diseñado también para resolver problemas combinatorios difíciles. La característica más sobresaliente de ASP es que permite representar fácilmente cláusulas por defecto (*defalut*), con las cuáles es posible describir reglas del tipo "normalmente".

Por ejemplo, durante el diseño de un planificador de tareas para un robot mediante ASP, es posible representar la incerteza epistémica asumiendo que normalmente el robot podrá desplazarse por la trayectoria planificada previamente. El "normalmente" de esta regla se refiere al caso en el cual no existe ningún obstáculo en el camino del robot. Pero si un objeto se interpone en su camino, el cambio en el entorno percibido por el robot, dispara la regla del plan que permite considerar este evento aleatorio. El planificador, de acuerdo a ciertas políticas establecidas previamente, pondera si es necesario replanificar la trayectoria

No menos importante resulta destacar el amplio rango de aplicaciones prácticas que han sido resueltas con ASP. Entre las más destacadas se encuentran el sistema para toma de decisiones de una nave espacial, planificación, robótica, configuración, reconfiguración, verificación asistida por computadora, configuración de producto basado en Web, enrutamiento protocolos de seguridad, análisis de inhibición de redes, y diagnóstico [2].

Además, es importante destacar que la investigación en esta área se ha expandido en gran parte gracias a que existen numerosas y eficientes máquinas de inferencia para la solución de problemas representados en ASP. Destacadamente, en la Sexta Competencia de ASP- 2015, participaron 13 razonadores ASP. Los ganadores por categoría son: Categoría *Main Track* (ME-ASP, WASP+DLV, LP2NORMAL+CLASP); Categoría Maratón (WASP + DLV, ME-ASP, LP2NORMAL + CLASP).

3.4. Razonadores SMT

Actualmente existen diversos campos de estudio de KRR que tradicionalmente se habían desarrollado de manera separada, tales como: Answer Set Programming (ASP) [2], Satisfacibilidad Proposicional (SAT) [16], Programación con Restricciones (Constraint Logic Progamming), CSP/CLP [37]. Todas estas subáreas son representativas de las diferentes direcciones de investigación que han surgido en el área de razonamiento automatizado.

Aunque estas metodologías son de propósito general, en la práctica se ha mostrado que cada una de ellas resulta más conveniente para diferentes tareas de razonamiento.

Por ejemplo, ASP ha mostrado ser una herramienta muy efectiva para la formalización de tareas compolejas de planificación, mientras que CSP es muy eficiente para la solución de problemas de calendarización de tareas (*scheduling*).

Sin embargo, existen muchos procesos de producción en la industria en las que es necesaria la elaboración de planes complejos que estén profundamente entrelazados, como por ejemplo, el plan general con el programador de tareas (*scheduler*). En esta situación ni ASP ni CSP por sí solos son suficientes para resolver este tipo de problemas.

Por lo que hace a los resolvedores SAT, éstos han sido aplicados exitosamente en tareas de análisis, prueba y verificación. Además, los razonadores SAT han incrementado dramáticamente su poder de resolución los últimos veinte años y pasaron de resolver problemas "simples" a principios de los 90's, hasta los complejos problemas prácticos que pueden

resolver en la actualidad, con capacidad para manejar millones de variables y millones de reglas.

Es importante destacar también que la aplicabilidad de SAT en la resolución de diversos problemas, sobre todo industriales, proviene no solo de los algoritmos avanzados que subyacen en el resolvedor SAT, sino que es también el resultado de la integración de SAT con algoritmos especializados que soportan otras teorías tales como: Diferencia Aritmética, Aritmética Lineal, Arreglos, Vectores de Bits, Funciones no interpretadas, etc.

Es decir, actualmente existe una tendencia hacia la solución de problemas difíciles mediante enfoques híbridos, combinando algoritmos y sistemas provenientes de diferentes subcampos de AI. Un ejemplo muy conocido de esta tendencia son las Teorías de Satisfacibilidad Modulo (*Satisfiability Modulo Theories, SMT*) [32] y [30].

El objetivo de SMT consiste en determinar la satisfacibilidad de fórmulas lógicas de una o más teorías diferentes entre sí.

Desafortunadamente, los problemas SMT pertenecen a una clase de complejidad muy alta y por ello resulta inconveniente construir una máquina de inferencias que pueda resolver arbitrariamente cualquier problema SMT. Para resolver problemas SMT, se ha adoptado una estrategia que consiste en resolver eficientemente problemas que existen en la práctica, asumiendo que, aunque potencialmente alta, de la mayoría de las fórmulas producidas por los algoritmos de análisis de las teorías subyacentes, solo un conjunto pequeño de éstas resulta crítica para establecer la satisfacibilidad del problema. De esta manera, en SMT, la interpretación de los símbolos se restringe a alguna teoría base.

Una estrategia que ha dado un fuerte impulso al área de investigación de SMT es la creación de un lenguaje común para la descripción de la sintaxis y la semántica de las teorías, y la otra es la realización de competencias anuales. El lenguaje de descripción es llamado SMT-lib. Posteriormente se empezaron a utilizar los formatos de descripción, como por ejemplo DIMACS.

Actualmente existen numerosos razonadores SMT, cada uno de ellos soporta diferentes teorías y diferentes lenguajes de descripción de las teorías. La siguiente tabla resume la información de algunos destacados razonadores SMT.

Tabla 3.4.1. Teorías y Lenguajes soportados por algunos Razonadores SMT

Nombre	AD	AL	ANL	Q	Lenguaje
CVC4	Sí	Sí	Sí*	Sí	SMT-LIB v1.0/v2.0, lenguaje nativo
MathSAT 5	No	Sí	No	No	SMT-LIB v1.2/v2.0, formato DIMACS, lenguaje nativo
SMTInterpol	No	Sí	No	No	SMT-LIB v1.2/v2.0, formato DIMACS
veriT	Sí	Sí	No	Sí	SMT-LIB v1.2/v2.0, formato DIMACS
Yices 2	Sí	Sí	No	No	SMT-LIB v1.2/v2.0, lenguaje nativo
Z3	Sí	Sí	Sí	Sí	SMT-LIB v2.0, formato Simplify, formato DIMACS

*soporte limitado a través de conversión a aritmética lineal. (AD : Aritmética de Diferencia; AL: Aritmética Lineal; ANL: Aritmética No-Lineal; Q: Cuantificadores)

4. CONCLUSIONES

La aplicación del SAT ha sido remarcablemente exitosa en muy diversos problemas. Gran parte de su éxito es debido al gran número de eficientes máquinas de inferencias que han sido desarrolladas en la última década. Esto resulta bastante sorprendente si consideramos que SAT es NP-completo. Aunque actualmente no se tiene una respuesta única para esta paradoja, si es conocido que los problemas tomados del mundo real tienen ciertas características que permiten la solución del problema de manera eficiente. Entre las características más destacadas están:

- Las instancias de los problemas 3-SAT solo son difíciles si la relación entre cláusulas y variables se encuentran en una angosta región cercana a 4.25.
- Los problemas del mundo real tienen estructura del tipo estático y/o dinámico. Además, este tipo de problemas son modulares.
- Los problemas reales a menudo tienen demostraciones que son cortas.

Para más detalles sobre este tema, vea [16] y [13].

Al igual que SAT, el problema CSP y ASP son **NP**-completo, pero como ambos han adoptado muchas de las estrategias y técnicas que han sido desarrolladas por la comunidad de SAT, los razonadores CSP y ASP también obtienen soluciones a problemas reales que teóricamente pertenecen a la clase NP-completo, pero que frecuentemente muestran algunas propiedades estructurales que pueden ser explotadas para reducir el espacio de búsqueda y encontrar la solución en un tiempo conveniente.

Por ejemplo, en todos estos razonadores para SAT, CSP Y ASP, una de las propiedades estáticas es el grado de aciclicidad de los grafos de restricciones. Lo que significa que una instancia de problemas de optimización combinatoria es soluble en tiempo polinomial si el ancho del árbol (*tree-width*) de su grafo de restricciones está acotado por una constante [38]. Por cuanto hace a las propiedades estructurales dinámicas, en este enfoque se tiene como estrategia la búsqueda de estructuras escondidas, las que se determinan en tiempo de ejecución mediante el análisis de los datos generados durante la instanciación. Estas estructuras son conocidas como puertas traseras (*backdoor set*) relativas a las variables, cuya instanciación transforma un problema **NP**-completo a uno tratable [10], [16] y [13].

Si bien, los razonadores de CSP han sido aplicados exitosamente en la solución de problemas reales, también tienen algunos inconvenientes. Uno de ellos es que frecuentemente los problemas reales se encuentran sobre-restringidos y no existe una solución para ese conjunto dado de restricciones. Una forma en que este problema ha sido resuelto, es usar preferencias en lugar de restricciones, o modificar ligeramente las restricciones de forma que se encuentre una solución sin modificar demasiado el problema original [11] y [21].

En conclusión, todos los tipos de razonadores analizados han experimentado un incremento en eficiencia de uno ó más órdenes de magnitud con respecto a los razonadores de la década pasada. En consecuencia, cada día más y más procesos y servicios serán automatizados en base a técnicas de representación del conocimiento y razonamiento. Algunos ejemplos son, robots totalmente independientes, operando en el mundo real y fábricas completamente automatizadas.

5. REFERENCIAS

1. Abío, I.: Solving hard industrial combinatorial problems with SAT. Ph.D. thesis, Technical University of Catalonia (UPC) (2013).

2. Brewka, G., Niemel¨a, I., Truszczy´nski, M.: Answer set programming at a glance. Communications of the ACM 54(12), 92–103 (2011)

3. Bruynooghe, M., Blockeel, H., Bogaerts, B., De Cat, B., De Pooter, S., Jansen, J., Labarre, A., Ramon, J., Denecker, M., Verwer, S.: Predicate logic as a modeling language: modeling and solving some machine learning and data mining problems with IDP3. Theory and Practice of Logic Programming (TPLP), pp. 783–817, November 2015.

4. Cook, S. "The Complexity of Theorem Proving Procedures," Proc. Third Annual ACM STOC Symp. (1971), 151-158.

5. Church, A., "A note on the Entscheidungsproblem", Journal of Symbolic Logic, 1 (1936), pp 40–41.

6. Davis, Martin; Putnam, Hilary (1960) A Computing Procedure for Quantification Theory. Journal of the ACM. 7 (3): 201–215. doi:10.1145/321033.321034.

7. Davis, M., Logemann, G. and Loveland, D. A machine program for theorem-proving. Communications of the ACM, 5(7):394–397, 1962.

8. Davis, E., and Marcus, G. (2015). Commonsense Reasoning and Commonsense Knowledge in Arti- ficial Intelligence. Communications of the ACM , 58 (9), 92–103.

9. Delgrande J.P. and Gupta, A. Two results in negation-free logic. Applied Mathematics Letters, 6(6):79–83, 1993.

10. Dilkina, B., Gomes, C. and Sabharwal, A.: Tradeoffs in the complexity of backdoor detection for combinatorial problems. In: CP-07, 13th International Conference on Principles and Practice of Constraint Programming, Vol. 4741 of LNCS, (pp. 256-270), Providence, RI, USA (2007).

11. Downing, N., Feydy T. and Stuckey P.: Unsatisfiable cores for constraint programming. CoRR arxiv.org/abs/1305.1690. (2013) Accessed 01 Sept 2016.

12. Fortnow, L., The status of the P Problema P versus NP Problema. Communications of the ACM 52 (9): 78–86. 2009. *doi*:*10.1145/1562164.1562186*.

13. Gaspers, S., Misra, N., Ordyniak, S., Szeider, S., and Zivny, S. (2014). Backdoors into heterogeneous classes of SAT and CSP. In C.E. Brodley, and P. Stone (Eds.), *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, July 27 -31, 2014, Québec City, Québec, Canada* (pp. 2652–2658). AAAI Press.

14. Gelfond, M, and Lifschitz, V.: The stable model semantics for logic programming. In: ICLP 1988, Proceedings of the Fifth International Conference on Logic Programming, pp.1070-1080, Seattle, Washington, USA (1988).

15. Gilmore, P.C.: A Proof Method for Quantification Theory, IBM J. Res. Develop. 4 (1960), 28-35.

16.Gomes, C.P., Kautz, H., Sabharwal, A., Selman, B.: Satisfiability solvers. In: van Harmelen, F., Lifschitz, V., Porter, B. (eds.) Handbook of Knowledge Representation, pp. 89–134. Elsevier (2008)

17. Goultiaeva, A. and Bacchus, F. Off the trail: Re-examining the CDCL algorithm. In Proceedings of Theory and Applications of Satisfiability Testing (SAT), pages 30–43, 2012.

18. Hanks, S. and McDermott, D. Default reasoning, nonmonotonic logics, and the frame problem. Proc. AAAI-86, Philadelphia, PA (1986).

19. Harmelen, F. V., Lifschitz, V. and Porter, B.: Handbook of Knowledge Representation (Foundations of Artificial Intelligence), Elsevier, Amsterdam, The Netherlands (2008).

20. Hillbert, D. and Ackerman, W. 1928. Grundzüge der theoretischen Logik (Principles of Mathematical Logic). Springer-Verlag, ISBN 0-8218-2024-9.

21. Hoeve, W.: Over-constrained problems. In Milano, M. and Van Hentenryck, P. (Eds.): Hybrid Optimization: the 10 Years of CPAIOR, pp.191-225, Springer (2011).

22. Hölldobler, S., Manthey, N., Philipp, T., Steinke. P. Generic CDCL - A formalization of modern propositional satisfiability solvers. In Daniel Le Berre, editor, 5th Workshop on Pragmatics of SAT — POS 2014, volume 27 of EPiC Series, pages 89–102. EasyChair, 2014.

23. Johnson, D. A brief history of NP-completeness, 1954–2012. Grötschel M, ed. Documenta Mathematica—Optim. Stories. 21 St Internat. Sympos. Math. Programming (Journal der Deutschen Mathematiker-Vereinigung, Berlin), 359–376.

24. Karp, R. M. Reducibility Among Combinatorial Problemas. En R. E. Miller and J. W. Thatcher (editors). Complexity of Computer Computations. New York: Plenum. pp.85-103, 1972.

25. Karp, R.M. Combinatorics, Complexity and Randomness. Com of ACM 29, 98-109, 1986.

26. Kautz, H., Selman, B. (2007)The state of SAT. Discrete Appl. Math. 155:1514–1524.

27. Lombardi, M. and Milano, M.: Optimal Methods for Resource Allocation and Scheduling: a Cross-Disciplinary Survey, Constraints, 17(1), 51-85 (2012).

28.Marek, V. W. and Truszczynski, M. (1999). Stable models and an alternative logic programming paradigm. In Apt, K., Marek, V. W., Truszczynski, M., and Warren, D. S., editors, The Logic Programming Paradigm. A 25-Year Perspective, pages 375-398. Springer Verlag.

29. Marques-Silva J.P. and Sakallah, K.A. "Boolean satisfiability in electronic design automation," in Proc. Design Automation Conf., Los Angeles, CA, Jun. 2000, pp. 675–680.

30. Michel, R., Hubaux, A., Ganesh, V. and Heymans, P. An SMT-based approach to automated configuration. In Pascal Fontaine and Amit Goel, editors, Proc. 10th International Workshop on Satisfiability Modulo Theories (SMT), pages 107–117, 2012.

31. Negri, E., Fumagalli, L., Garetti, M., Tanca, L. Requirements and languages for the semantic representation of

manufacturing systems. Computers in Industry. 2015. Article in Press.

32. Nieuwenhuis, R., Oliveras, A., Tinelli, C.: Solving SAT and SAT modulo theories: From an abstract Davis-Putnam-Logemann-Loveland procedure to DPLL(T). Journal of the ACM 53(6), 937–977 (2006).

33. Novas, J.M. and Henning, G.P.: Reactive scheduling framework based on domain knowledge and constraint programming, Computers and Chemical Engineering, 34(12), 2129-2148 (2010).

34. Pelov, N., Denecker, M. and Bruynooghe, M. Well-founded and stable semantics of logic programs with aggregates.

TPLP, 7(3):301–353, 2007.

35. Pham, D.N., Thornton, J.R. and Sattar, A. Modelling and solving temporal reasoning as propositional satisfiability. In Proceeding of the 4th International Workshop on Modelling and Reformulating, Sitges, Spain, 2005.

36. Robinson, J.A., A Machine-oriented Logic Based on the Resolution Principle. J. ACM 12, 1 (Jan, 1965), 23-41.

37. Rossi, F., van Beck, P., Walsh, T.: Constraint porgramming. In: van Harmelen, F., Lifschitz, V., Porter, B. (eds.) Handbook of Knowledge Representation, pp. 181–212. Elsevier (2008).

38. Samer, M. and Szeider, S.: Constraint satisfaction with bounded treewidth revisited, Journal of Computer and System Sciences, 76(2), pp. 103-114 (2010).

39. Shen.Y.D. and Eiter, T. (2016) Evaluating Epistemic Negation in Answer Set Programming. Artificial Intelligence. 237: 115-135.

40. Szeider, S. The parameterized complexity of constraintsatisfaction and reasoning. In H. Tompits, S. Abreu, J. Oetsch, J. Pührer, D. Seipel, M. Umeda, and A. Wolf, editors, INAP 2011, and WLP 2011, Vienna, Austria, September 28-30, 2011, Revised Selected Papers, volume 7773 of Lecture Notes in Computer Science, pages 27–37. Springer, 2011.

41. Turing, A. "On computable numbers, with an application to the Entscheidungsproblem", Proceedings of the London Mathematical Society, Series 2, 42 (1936-7), pp 230–265.

42. Van Gelder, A., Ross, A. K. and Schlipf, J. S.: The well-founded semantics for general logic programs, Journal of ACM, 38(3), 620-650 (1991).

43. Wigderson, A. The Gödel Phenomena in Mathematics: A Modern View. In: Kurt Gödel and the Foundations of Mathematics: Horizons of Truth; Baaz, M.; Papadimitriou, C.H.; Putnam, H.W.; Scott, D.S., Eds.; Cambridge University Press: Cambridge, 2010.

44. Zeballos, L. J., Castro, P. M. and Méndez, C. A.: Integrated constraint programming scheduling approach for automated wet-etch stations in semiconductor manufacturing, Industrial and Engineering Chemistry Research, 50(3), 1705-1715 (2011).