Capítulo 5

Cómputo de Safe Beliefs

El método propuesto por Davis-Putnam para resolver el problema de satisfiabilidad (SAT) en lógica clásica proposicional es uno de los más prácticos. Hantao Zhang (y otros) desarrollaron una implementación muy eficiente para el método de Davis-Putnam llamado SATO [27]. Investigaciones recientes han demostrado que este tipo de programas visto como un módulo independiente puede ser utilizado, bajo ciertas condiciones, para extraer eficientemente modelos estables de programas lógicos. Se ha demostrado por medio de algunos ejemplos que el tiempo de ejecución por parte de SATO es diez veces más rápido que el tiempo utilizado por SMODELS para encontrar modelos estables de un programa.

Una alternativa, sugerida en la Sección 4.2, considera la posible reducción de una teoría arbitraria, utilizando una transformacíon en tiempo polinomial con respecto al tamaño del programa, en progrmas disyuntivos mas sencillos para despues encontrar los answer sets del programa a través de software ya existente (como DLV o SMODELS) computando así sus safe beliefs.

El principal problema con este tipo de transformaciones, es que estas pueden introducir más átomos para poder realizar reducciones " *de tal forma que el espacio de búsqueda para el procedimiento de DP para encontrar las soluciones crece con gran*

amplitud ", como E. Giunchiglia and R. Sebastiani [9] han señalado.

Se presenta un método para computar safe beliefs de teorias arbitrarias, basado en el metodo de Davist-Putnam. Evitaremos introducir nuevos átomos utlizando el conjunto de reducciones presentadas en la Sección 4.1, en lugar de tratar de reducir la estructura del programa, que son de ayuda para evlauar el efecto de las decisiones realizadas por el método de DP reduciendo el tamaño del programa evaluado.

Nuestro algoritmo, descrito en la función GetSafeBeliefs toma como entrada un programa y 2 conjuntos de átomos $M_{\mathcal{W}}$ y $M_{\mathcal{N}}$ que contienen átomos que pueden suponerse como positivos o negativos respectivamente. El algorimto primeramente reduce el programa utlizando la información contenida en $M_{\mathcal{W}}$ y $M_{\mathcal{N}}$, y después revisa algunas condiciones sencillas que pueden rechazar tempranamente un modelo, evitando realizar ramificaciones no necesarias. Por otro lado, la función es llamada recursivamente adivinando el valor de un átomo nuevo devolviendo un conjunto de todos los safe beliefs encontrados. Como en [9], nosotros aplicamos la partición dentro del algoritmo solo para los átomos que ocurren en el programa original.

función GetSafeBeliefs$(P, M_{\mathcal{W}}, M_{\mathcal{N}})$

$P \longleftarrow$ Reduce$(P, M_{\mathcal{W}}, M_{\mathcal{N}})$.

Si $P = \{\bot\}$ **o** $\mathcal{L}_P \subset M_{\mathcal{W}}$ **entonces**

return $\emptyset$.

en otro caso Si $\mathcal{L}_P = M_{\mathcal{W}}$ **entonces**

return CheckPositive(P).

en otro caso

Sea x un átomo en $\mathcal{L}_P \setminus M_{\mathcal{W}}$.

return

GetSafeBeliefs$(P, M_{\mathcal{W}} \cup \{x\}, M_{\mathcal{N}})$

$\cup$ GetSafeBeliefs$(P, M_{\mathcal{W}}, M_{\mathcal{N}} \cup \{x\})$.

Fin de condión

Fin función

La función GetSafeBeliefs esta tratando de adivniar, mediante el método de DP, para cada átomo cuando puede ser este asumido como positivo o negativo. Cuando no es posible realizar más suposiciones, la función CheckPositive es utilizada para finalmente determinar si el conjunto $M_{\mathcal{W}}$ es o no un safe belief de la teoría reducida T.

Entonces la función CheckPositive toma como entrada una teoría T. No hay que olvidar que las propiedades de las reducciones aplicadas hacen que T sea siempre positivo. La función debe devolver $\{\mathcal{L}_T\}$ sí $T \vdash \mathcal{L}_T$ y $\{\}$ en otro caso.

El problema de determinar, para una teoría positiva dada T, si $T \vdash \mathcal{L}_T$ es al parecer un problema NP. Es sencillo encontrar reducciones en el problema *Check Positive* para transformalrlo a otro problema NP-completo conocido como encontrar el modelo mínimo, o incluso a un problema de tipo SAT. Un método similar al DP puede ser utilizado nuevamente para resolver este problema tratando de encontrar un modelo $M \subset \mathcal{L}_T$

(inclusión propia) para la teoría T.

Si tal modelo existe entonces es falso que $T \vdash \mathcal{L}_T$, en otro caso significa que se tiene una prueba para hecho de que $T \vdash \mathcal{L}_T$.

En el momento en que el algoritmo requiere seleccionar un atomo de $\mathcal{L}_P \setminus M_{\mathcal{W}}$, es posible utilizar heuristicas para mejorar su comportamiento. De hecho, es posible seleccionar el átomo con mas ocurrencias en el programa P.

Teorema 5.1 (Correctez). *La función* GetSafeBeliefs$(T, \emptyset, \emptyset)$ *termina y devuelve el conjunto de safe beliefs de la teoría* T.

Demostración. (Sketch) Es inmediato que el algoritmo termina ya que el lenguaje de P es finito. La función CheckPositive se asume que es correcta y devuelve el correspondiente conjunto de safe beliefs. El caso final resuelve el problema recursivamente agregando nuevos átomos al safe belief parcial computado en $M_{\mathcal{W}}$ y $M_{\mathcal{N}}$. □