El problema SAT y el algoritmo DPLL
Lógica Computacional 2023-I, Nota de clase 4

Favio Ezequiel Miranda Perea Araceli Liliana Reyes Cabello
Lourdes Del Carmen González Huesca Pilar Selene Linares Arévalo

6 de septiembre de 2022
Facultad de Ciencias UNAM

1. El problema SAT

El problema de satisfacibilidad para la lógica proposicional, denotado usualmente como SAT es de suma importancia en la teoría de la complejidad computacional y en general en Ciencias de la Computación. Este problema tiene aplicaciones tanto teóricas como prácticas. En particular resultó ser el primer problema NP-completo, lo cual fue probado por Cook en 1971. De hecho antes del Teorema de Cook, el concepto de NP completud ni siquiera existía.

En su forma más general el problema de satisfacibilidad es el siguiente: Dada una fórmula A, hallar una interpretación $\mathcal{I}$ tal que $\mathcal{I} \models A$. Es decir, hallar un modelo $\mathcal{I}$ para A. Sin embargo es más común enunciar al problema de satisfacibilidad o problema SAT como sigue:

Dado un conjunto $P = \{p_1, \ldots, p_n\}$ de variables proposicionales y
un conjunto C de cláusulas con variables en P
¿Existe una interpretación $\mathcal{I}$ que satisfaga a C?

Por supuesto que ambos problemas, el general para cualquier fórmula A, y el caso para conjuntos de cláusulas resultan equivalentes.

El área de la lógica conocida como razonamiento automatizado se encarga, entre muchas otras cosas, del desarrollo de algoritmos para resolver el problema SAT sobre todo con respecto a ciertas familias de cláusulas para los que sí existen algoritmos eficientes, como son las llamadas cláusulas de Horn. Hoy en día existen programas sumamente eficientes para resolver el problema SAT, los cuales se conocen como solucionadores SAT [1]

Es de interés observar que aun cuando se restringe el problema de forma que cada cláusula dada tenga a lo más k-variables (el llamado problema k-SAT), éste sigue siendo NP-completo para $k \geq 3$. Por otra parte, el caso $k = 2$ se resuelve en tiempo polinomial.

2. Resolución de problemas mediante SAT

La importancia práctica del problema SAT radica en que, mediante una codificación adecuada, se pueden resolver problemas de prácticamente cualquier ámbito usando los poderosos resolvedores SAT. La idea general consiste en codificar un problema dado $\mathcal{P}$ mediante una fórmula proposicional $A_{\mathcal{P}}$ de modo que $\mathcal{P}$ tiene solució si y sólo si $A_{\mathcal{P}}$ es satisfacible. Veamos un ejemplo concreto de codificación. Un ejemplo más sofisticado se encuentra en el apéndice de esta nota.

[1] En inglés *SAT solvers*.

2.1. El problema de las n-reinas

Este problema consiste en colocar n reinas en un tablero de ajedrez de $n \times n$ de modo que no se ataquen entre sí. Se trata de un problema de satisfacción de restricciones. La codificación utiliza n^2 variables proposicionales, digamos $q_{ij},\ 1 \leq i, j \leq n$ tal que q_{ij} significa que hay una reina en la casilla (i, j) del tablero. Por simplicidad vamos a codificar el caso cuando $n = 4$, dejando la generalización como ejercicio.

- Cada columna del tablero tiene al menos una reina.

$$CM =_{def} (q_{11} \vee \ldots \vee q_{14}) \wedge (q_{21} \vee \ldots \vee q_{22}) \wedge \ldots \wedge (q_{41} \vee \ldots \vee q_{44})$$

- Cada renglón del tablero tiene al menos una reina.

$$RM =_{def} (q_{11} \vee \ldots \vee q_{41}) \wedge (q_{12} \vee \ldots \vee q_{42}) \wedge \ldots \wedge (q_{14} \vee \ldots \vee q_{44})$$

- No hay más de una reina en cada columna.

$$C_1 =_{def} (q_{11} \rightarrow \neg q_{12} \wedge \neg q_{13} \wedge \neg q_{14}) \wedge (q_{12} \rightarrow \neg q_{13} \wedge \neg q_{14}) \wedge (q_{13} \rightarrow \neg q_{14})$$

  De manera análoga definimos una fórmula C_i para cada columna i y finalmente definimos

$$C =_{def} C_1 \wedge \ldots \wedge C_n$$

- No hay más de una reina en cada columna. Análoga a la anterior generando una fórmula

$$R =_{def} R_1 \wedge \ldots \wedge R_n$$

- A lo más una reina en cada diagonal. La idea es dar una fórmula D^+ para las diagonales positivas y una fórmula D^- para las diagonales negativas:

  $D^+ =_{def} (q_{13} \rightarrow \neg q_{14}) \wedge (q_{12} \rightarrow \neg q_{23} \wedge \neg q_{34}) \wedge (q_{11} \rightarrow \neg q_{22} \wedge \neg q_{33} \wedge \neg q_{44}) \wedge (q_{21} \rightarrow \neg q_{32} \wedge \neg q_{43}) \wedge (q_{31} \rightarrow \neg q_{42})$

  D^- se define de manera análoga y finalmente definimos $D =_{def} D^+ \wedge D^-$.

3. El Algoritmo DPLL

El algoritmo de Davis-Putnam-Logemann-Loveland o DPLL (1962) es un procedimiento refutacionalmente completo para decidir la satisfacibilidad de fórmulas proposicionales en forma normal conjuntiva. Se trata de un refinamiento del algoritmo original de Davis y Putnam de 1960 el cual se basa en resolución binaria. Este algoritmo sigue siendo parte importante de muchos solucionadores SAT refutacionalmente completos y se implementó por primera vez en una computadora IBM 704, y resulto muy superior al primer método de razonamiento automatizado que existío (el método de Gilmore) al resolver este en 2 minutos un problema que aquel no pudo resolver en 21 minutos antes de agotar sus recursos. Dicho problema se había resuelto a mano mediante DPLL en 30 minutos.

Las implementaciones de DPLL se basan en las siguientes reglas intuitivas que transforman un conjunto de cláusulas $\mathcal{S}$ en otro conjunto (o conjuntos) equisatisfacible[2] $\mathcal{S}'$:

[2]Decimos que $\mathcal{S}$ es equisatisfacible a $\mathcal{S}'$ si se cumple que $\mathcal{S}$ tiene un modelo syss $\mathcal{S}'$ tiene un modelo. Aquí el modelo en cada caso no tiene por que ser el mismo. Esta relación se denota con $\mathcal{S} \sim_{sat} \mathcal{S}'$

- Regla de la tautología: eliminar de $\mathcal{S}$ toda cláusula que sea tautología, es decir, toda cláusula que tenga un par de literales complementarias.
- Regla de la cláusula unitaria (RCU): Si $C \in \mathcal{S}$ es una cláusula unitaria, digamos $C = \ell$, eliminar de $\mathcal{S}$ todas las cláusulas que contengan a ℓ (incluyendo a C) y eliminar ℓ^c de las cláusulas restantes.
- Regla de la literal pura (RLP): Si existe una literal ℓ que figura en $\mathcal{S}$ y tal que ℓ^c no figura en $\mathcal{S}$ entonces eliminar de $\mathcal{S}$ todas las cláusulas que contengan a ℓ. En tal caso la literal ℓ se llama una literal pura.
- Regla de descomposición o separación (RD): Si existe una literal ℓ tal que $\mathcal{S}$ puede descomponerse como $\mathcal{S} = A \cup B \cup R$ donde:
  - todas las cláusulas de A tienen a ℓ y no a ℓ^c;
  - todas las cláusulas de B tienen a ℓ^c pero no a ℓ
  - y en R no figuran ni ℓ ni ℓ^c

  entonces reemplazar $\mathcal{S}$ por $\mathcal{S}_1$ y $\mathcal{S}_2$, donde

  $\mathcal{S}_1 = A' \cup R$
  $\mathcal{S}_2 = B' \cup R$
  $A' = \{C \setminus \ell \mid C \in A\}$
  $B' = \{C \setminus \ell^c \mid C \in B\}$
  $C \setminus \ell$ denota a la cláusula obtenida al eliminar la literal ℓ de C.

Obsérvese que si RCU no es aplicable entonces RLP sí lo es y si RLP no es aplicable entonces RD sí lo es.

Nótese que el conjunto vacío de cláusulas se representa, como es usual, mediante $\varnothing$ y no debe confundirse con la cláusula vacía denotada $\square$

Existen algunos casos particulares de aplicación de la reglas RCU que vale la pena mencionar:

- Si $\mathcal{S} = \{\ell\}$ entonces RCU nos devuelve al conjunto vacío de cláusulas $\mathcal{S} = \varnothing$
- Si todas las cláusulas de $\mathcal{S}$ tienen a la literal ℓ entonces RLP nos devuelve $\mathcal{S} = \varnothing$.
- Si $\ell^c \in \mathcal{S}$ entonces la aplicación de RCU con ℓ nos lleva a que $\square \in \mathcal{S}$. Por ejemplo si $\mathcal{S} = \{p, \neg r \vee \neg p \vee s, \neg p, p \vee \neg q \vee t\}$ entonces la aplicación de RCU con p devuelve $\mathcal{S} = \{\neg r \vee s, \square\}$

Es muy importante notar la diferencia entre el conjunto vacío de cláusulas y el conjunto $\{\square\}$ cuyo único elemento es la cláusula vacía. El primero se genera al eliminar todas las claúsulas de un conjunto mientras que el segundo se obtiene al eliminar todas las literales de una cláusula pero sin eliminar la cláusula, la cual queda vacía.

El algoritmo DPLL decide si un conjunto $\mathcal{S}$ es satisfacible aplicando las reglas anteriores exhaustivamente, en tal caso los conjuntos finales arrojados por el proceso son necesariamente de la forma $\varnothing$ o $\{\square\}$. Si todos ellos son $\{\square\}$ entonces el algoritmo resuelve que $\mathcal{S}$ es insatisfacible. En otro caso, algún conjunto es $\varnothing$ y el algoritmo resuelve que $\mathcal{S}$ es satisfacible.

Veamos ahora que las reglas son correctas lo cual garantiza que su uso exhaustivo permite decidir la satisfacibilidad del conjunto original $\mathcal{S}$.

Proposición 1 *Las reglas RCU y RLP son correctas. Es decir, si $\mathcal{S}$ es un conjunto de cláusulas y $\widehat{\mathcal{S}}$ es el conjunto resultado de aplicar RCU o RLP a $\mathcal{S}$ entonces $\mathcal{S} \sim_{sat} \widehat{\mathcal{S}}$.*

Demostración. Ejercicio.

Proposición 2 *La regla RD es correcta. Es decir, si $\mathcal{S}$ es un conjunto de cláusulas y $\mathcal{S}_1$, $\mathcal{S}_2$ son los conjuntos resultantes de aplicar RD a $\mathcal{S}$ entonces $\mathcal{S}$ es satisfacible si y sólo si $\mathcal{S}_1$ es satisfacible o $\mathcal{S}_2$ es satisfacible.*

Demostración. Ejercicio.

En las versiones más modernas de DPLL no se utiliza la regla de la literal pura, y el uso de la regla de la cláusula unitaria se hace con todas las literales posibles. Este proceso se conoce como propagación de constantes booleanas BCP (boolean constant propagation). Además no se utiliza la regla de la literal pura, y el uso de la regla de la cláusula unitaria se hace con todas las literales posibles. Este proceso se conoce como propagación de constantes booleanas BCP (boolean constant propagation) . Veamos un pseudocódigo imperativo que representa a este proceso:

```
bcp(S) =  while ℓ ∈ S do
            foreach C ∈  S do
              if  ℓ ∈ C  then
                S  :=  S \ {C}
              else
               if  ℓ^c  ∈ C  then
                S  :=  (S \ {C}) ∪  C \ (ℓ^c)
```

donde $\setminus$ es la diferencia de conjuntos (recuerde que podemos considerar a las cláusulas como conjuntos). En particular $\mathcal{C} \setminus \ell$ denota a la cláusula obtenida al eliminar la literal ℓ en $\mathcal{C}$. Por ejemplo si $\mathcal{C} = \neg p \vee s \vee \neg t \vee q$ entonces $\mathcal{C} \setminus \neg t = \neg p \vee s \vee q$.

Veamos ahora un pseudocódigo para el algoritmo dplll. La entrada es un conjunto de cláusulas $\mathcal{S}$ y la salida es `true` si el conjunto es satisfacible y `false` en otro caso.

```
dpll(S) = bcp(S);
          if S = ∅ then return true ;
          if □ ∈ S then return false ;
          ℓ  = chooseLiteral(S);
          return dpll(S ∪ {ℓ}) or dpll(S ∪ {ℓ^c})
```

La función `chooseLiteral` elige una literal que figura en $\mathcal{S}$ de acuerdo a ciertas heurísticas. Por ejemplo, elegir en orden alfabético y preferir literales positivas a negativas o bien ordenar las ordinales de acuerdo al problema particular, digamos $q < s < r < p$, etc. No discutiremos la implementación de esta función.

Nótese que esta versión de dpll no implementa la regla de la literal pura y que las últimas lineas están implementando la regla de descomposición al elegir una literal, agregarla al conjunto original y llamar nuevamente al algoritmo.

Estos pseudocódigos corresponden a una implementación imperativa que resulta complicada e inadecuada para implementarse funcionalmente, así como para la verificación de la corrección del algoritmo. En su lugar veremos a continuación una versión de dpll mediante un sistema de transición que nos permitirá una implementación funcional directa así como una prueba de corrección por inducción estructural.

3.1. DPLL mediante búsqueda de modelos hacia atrás

Presentamos a continuación una versión del algoritmo DPLL mediante un conjunto de reglas de transición que modelan una búsqueda hacia atrás de un modelo $\mathcal{M}$ para una fórmula F que está en forma normal conjuntiva. El modelo está representado mediante un conjunto de literales (aquellas que se consideran verdaderas) y la fórmula $F = \mathcal{C}_1 \wedge \mathcal{C}_2 \ldots \wedge C_k$ se representa mediante el conjunto de cláusulas $\{\mathcal{C}_1, \ldots, C_k\}$.

Como preproceso se supone que la fórmula F no tiene tautologías, es decir cláusulas con literales complementarias y que todas las cláusulas están reducidas, es decir, no tienen literales repetidas. El proceso de búsqueda de un modelo consiste en aplicar reglas de transición que manipulan configuraciones o expresiones de la forma $\mathcal{M} \models_? F$, las cuales representan a la pregunta ¿Es $\mathcal{M}$ un modelo para F?

Es importante observar que en lo que sigue representamos a un modelo $\mathcal{M}$ como un conjunto de literales, a saber aquellas que consideramos verdaderas. Por ejemplo, $\mathcal{M} = \{p, \neg q, r\}$ representa al modelo $\mathcal{I}$ tal que $\mathcal{I}(p) = \mathcal{I}(r) = 1$ e $\mathcal{I}(q) = 0$. De hecho podemos simplificar a este modelo como un conjunto de átomos $\mathcal{M} = \{p, r\}$ entendiendo que en la interpretación correspondiente las únicas variables verdaderas son las que figuran en $\mathcal{M}$ siendo todas las demás falsas. Las transiciones son de la forma

$$\mathcal{M} \models_? F \ \rhd \ \mathcal{M}' \models_? F'$$

y representan la reducción de la búsqueda de un modelo $\mathcal{M}$ para F, a la búsqueda de un modelo $\mathcal{M}'$ para F'. La relación $\ldots \rhd \ldots$ puede leerse como "para resolver ... basta resolver ...".
Veamos las reglas de transición:

- Regla de la cláusula unitaria (propagación de una constante booleana): para que $\mathcal{M}$ sea modelo de F, ℓ basta que $\mathcal{M}, \ell$ sea modelo de F.

  $$\texttt{unit}\ \ell: \qquad \mathcal{M} \models_? F, \ell \ \rhd \ \mathcal{M}, \ell \models_? F$$

  Es importante observar que esta regla sólo puede aplicarse si $\ell^c \notin \mathcal{M}$. En otro caso el modelo requeriría que tanto ℓ como ℓ^c fueran verdaderas lo cual es imposible.

- Regla de eliminación: para que $\mathcal{M}, \ell$ sea modelo de $F, \ell \vee C$ basta que $\mathcal{M}, \ell$ sea modelo de F:

  $$\texttt{elim}: \qquad \mathcal{M}, \ell \models_? F, \ell \vee C \ \rhd \ \mathcal{M}, \ell \models_? F$$

- Regla de reducción: para que $\mathcal{M}, \ell$ sea modelo de $F, \ell^c \vee C$ basta que $\mathcal{M}, \ell$ sea modelo de F, C:

  $$\texttt{red}: \qquad \mathcal{M}, \ell \models_? F, \ell^c \vee C \ \rhd \ \mathcal{M}, \ell \models_? F, C$$

  Obsérvese que un caso particular de la regla de reducción es: $\mathcal{M}, \ell \models_? F, \ell^c \ \rhd \ \mathcal{M}, \ell \models_? F, \square$.

- Regla de separación: para que $\mathcal{M}$ sea modelo de F basta que tanto $\mathcal{M}, \ell$ como $\mathcal{M}, \ell^c$ sean modelos de F.

  $$\texttt{split}\ \ell: \qquad \mathcal{M} \models_? F \ \rhd \ \mathcal{M}, \ell \models_? F \ ; \ \mathcal{M}, \ell^c \models_? F$$

- Regla de conflicto: La búsqueda de un modelo $\mathcal{M}$ para $F, \square$ falla. Es decir, no existe modelo para $F, \square$ (puesto que no existen modelos para $\square$).

  $$\texttt{conflict}: \qquad \mathcal{M} \models_? \mathcal{F}, \square \ \rhd \ \mathsf{fail}$$

- Regla de éxito: La búsqueda de un modelo $\mathcal{M}$ para la fórmula sin cláusulas $\varnothing$ (es decir, el conjunto vacío de cláusulas) siempre tiene éxito. Por vacuidad, cualquier modelo $\mathcal{M}$ hace verdadera a la fórmula vacía.

  $$\texttt{success}: \qquad \mathcal{M} \models_? \varnothing \ \rhd \ \checkmark$$

Para hallar un modelo de F iniciamos el proceso de búsqueda con $\cdot \models_? F$, es decir, inicialmente el modelo está vacío (no hay literales con valor de verdad asignado) hasta llegar a $\mathcal{M} \models_? \varnothing$ en cuyo caso $\mathcal{M}$ es un modelo para F.

Es muy importante observar la diferencia entre las reglas de eliminación `elim` y reducción `red`. La primera elimina cláusulas enteras del conjunto en cuestión. Mientras que la segunda nunca elimina cláusulas, sino que elimina únicamente literales dentro de cláusulas, es decir, reduce el número de literales que son complementarias a alguna del modelo. Por ejemplo, obsérvense las siguientes aplicaciones de dichas reglas:

- Por `elim` se tiene que:

$$p, \neg q \models_? s \vee t,\ r \vee \neg q \vee \neg s,\ r \vee \neg p \quad \rhd \quad p, \neg q \models_? s \vee t,\ r \vee \neg p$$

- Por `red` se tiene que:

$$p, \neg s \models_? q \vee t,\ \neg r \vee s \vee \neg t,\ \neg r \quad \rhd \quad p, \neg s \models_? q \vee t,\ \neg r \vee \neg t,\ \neg r$$

En particular se tiene por ejemplo lo siguiente:

- Por `elim` se cumple que:
$$\neg q, r, \neg t \models_? s \vee p \vee \neg t \vee q \quad \rhd \quad \neg q, r, \neg t \models_? \varnothing$$
pues la única cláusula que quedaba se eliminó.

- Por `red` se cumple que:
$$\neg r, s, \neg p \models_? \neg s \quad \rhd \quad \neg r, s, \neg p \models_? \square$$
puesto que la única cláusula que quedaba, a saber $\neg s$, se redujo a la vacía $\square$ dado que se borró la única literal presente.

Se observa también que las reglas de búsqueda son no-deterministas, en el caso de `unit` y `split` se debe elegir una literal ℓ cuya elección raramente es única. También es importante observar que la regla `split` produce dos metas o caminos a explorar, si el primero falla, se debe continuar con el segundo, esto corresponde al proceso de búsqueda en retroceso (*backtracking*). Si el segundo también falla entonces no hay modelos posibles. Por otra parte, si el primer camino tiene éxito se obtiene un modelo. En este caso el segundo camino puede omitirse si no nos interesa buscar más modelos, o bien explorarse para hallar más modelos.

El proceso para encontrar un modelo para una fórmula F o un conjunto de cláusulas $\mathcal{S}$ es exactamente el mismo puesto que se requiere que F esté en forma normal conjuntiva, F misma es un conjunto de cláusulas y viceversa un conjunto $\mathcal{S}$ puede considerarse como una sóla fórmula en FNC que consta de la conjunción de todas las cláusulas de $\mathcal{S}$.

Dado un conjunto de cláusulas $\mathcal{S}$, la búsqueda de modelos inicia con la meta $\cdot \models_? \mathcal{S}$, donde $\cdot$ denota al modelo vacío. Veamos un ejemplo:

Ejemplo 3.1 Consideremos el siguiente conjunto de cláusulas:

$$\mathcal{S} = \{\neg p \vee r \vee \neg t,\ \neg q \vee \neg r,\ p \vee \neg s,\ \neg p \vee q \vee \neg r \vee \neg s\}$$

Buscaremos los modelos de este conjunto usando DPLL mediante las reglas de transición.

$$
\begin{array}{rl}
\cdot & \models_? \neg p \vee r \vee \neg t,\ \neg q \vee \neg r,\ p \vee \neg s,\ \neg p \vee q \vee \neg r \vee \neg s \\
\rhd & \texttt{split}\ p \\
p & \models_? \neg p \vee r \vee \neg t,\ \neg q \vee \neg r,\ p \vee \neg s,\ \neg p \vee q \vee \neg r \vee \neg s\ ; \\
\neg p & \models_? \neg p \vee r \vee \neg t,\ \neg q \vee \neg r,\ p \vee \neg s,\ \neg p \vee q \vee \neg r \vee \neg s \\
\rhd & \texttt{elim} \\
p & \models_? \neg p \vee r \vee \neg t,\ \neg q \vee \neg r,\ \neg p \vee q \vee \neg r \vee \neg s\ ; \\
\neg p & \models_? \neg p \vee r \vee \neg t,\ \neg q \vee \neg r,\ p \vee \neg s,\ \neg p \vee q \vee \neg r \vee \neg s \\
\rhd^2 & \texttt{red} \\
p & \models_? r \vee \neg t,\ \neg q \vee \neg r,\ q \vee \neg r \vee \neg s\ ; \\
\neg p & \models_? \neg p \vee r \vee \neg t,\ \neg q \vee \neg r,\ p \vee \neg s,\ \neg p \vee q \vee \neg r \vee \neg s \\
\rhd & \texttt{split}\ r
\end{array}
$$

$$
\begin{array}{rl}
p,\ r & \models_? r \vee \neg t,\ \neg q \vee \neg r,\ q \vee \neg r \vee \neg s\ ; \\
p,\ \neg r & \models_? r \vee \neg t,\ \neg q \vee \neg r,\ q \vee \neg r \vee \neg s\ ; \\
\neg p & \models_? \neg p \vee r \vee \neg t,\ \neg q \vee \neg r,\ p \vee \neg s,\ \neg p \vee q \vee \neg r \vee \neg s \\
\rhd & \texttt{elim} \\
p,\ r & \models_? \neg q \vee \neg r,\ q \vee \neg r \vee \neg s\ ; \\
p,\ \neg r & \models_? r \vee \neg t,\ \neg q \vee \neg r,\ q \vee \neg r \vee \neg s\ ; \\
\neg p & \models_? \neg p \vee r \vee \neg t,\ \neg q \vee \neg r,\ p \vee \neg s,\ \neg p \vee q \vee \neg r \vee \neg s \\
\rhd^2 & \texttt{red} \\
p,\ r & \models_? \neg q,\ q \vee \neg s\ ; \\
p,\ \neg r & \models_? r \vee \neg t,\ \neg q \vee \neg r,\ q \vee \neg r \vee \neg s\ ; \\
\neg p & \models_? \neg p \vee r \vee \neg t,\ \neg q \vee \neg r,\ p \vee \neg s,\ \neg p \vee q \vee \neg r \vee \neg s \\
\rhd & \texttt{unit} \\
p,\ r,\ \neg q & \models_? q \vee \neg s\ ; \\
p,\ \neg r & \models_? r \vee \neg t,\ \neg q \vee \neg r,\ q \vee \neg r \vee \neg s\ ; \\
\neg p & \models_? \neg p \vee r \vee \neg t,\ \neg q \vee \neg r,\ p \vee \neg s,\ \neg p \vee q \vee \neg r \vee \neg s \\
\rhd & \texttt{red} \\
p,\ r,\ \neg q & \models_? \neg s\ ; \\
p,\ \neg r & \models_? r \vee \neg t,\ \neg q \vee \neg r,\ q \vee \neg r \vee \neg s\ ; \\
\neg p & \models_? \neg p \vee r \vee \neg t,\ \neg q \vee \neg r,\ p \vee \neg s,\ \neg p \vee q \vee \neg r \vee \neg s \\
\rhd & \texttt{unit} \\
p,\ r,\ \neg q,\ \neg s & \models_? \varnothing\ ; \\
p,\ \neg r & \models_? r \vee \neg t,\ \neg q \vee \neg r,\ q \vee \neg r \vee \neg s\ ; \\
\neg p & \models_? \neg p \vee r \vee \neg t,\ \neg q \vee \neg r,\ p \vee \neg s,\ \neg p \vee q \vee \neg r \vee \neg s \\
\rhd & \texttt{success} \\
p,\ \neg r & \models_? r \vee \neg t,\ \neg q \vee \neg r,\ q \vee \neg r \vee \neg s\ ; \\
\neg p & \models_? \neg p \vee r \vee \neg t,\ \neg q \vee \neg r,\ p \vee \neg s,\ \neg p \vee q \vee \neg r \vee \neg s \\
\rhd & \\
\ldots &
\end{array}
$$

La regla `success` arroja un modelo para $\mathcal{S}$ a saber: $p,\ r,\ \neg q,\ \neg s$. Dado que existen otras búsquedas se puede continuar con el proceso.

Se deja como ejercicio la búsqueda de modelos para los siguientes conjuntos de cláusulas:

- $\mathcal{S} = \{p \vee q,\ \neg q,\ \neg p \vee q \vee \neg r\}$

- $\mathcal{S} = \{p \vee q,\ p \vee \neg q,\ r \vee q,\ r \vee \neg q\}$
- $\mathcal{S} = \{p \vee \neg q,\ \neg p \vee q,\ q \vee \neg r,\ \neg q \vee \neg r\}$
- $\mathcal{S} = \{p \vee q,\ r \vee \neg q \vee \neg s,\ \neg p \vee s,\ \neg r\}$
- $\mathcal{S} = \{p \vee q \vee r,\ p \vee \neg q \vee \neg r,\ p \vee \neg w,\ \neg q \vee \neg r \vee \neg w, \neg p \vee \neg q \vee r,\ u \vee \neg x,\ u \vee x,\ q \vee \neg u,\ \neg r \vee \neg u\}$
- $\mathcal{S}$ es el conjunto de cláusulas para decidir la corrección del siguiente argumento: Diana, Elena y Frida son sospechosas de una fechoría y sólo una es culpable. La culpable estaba en la casa. Diana dice "*yo no fui, yo no estaba en la casa, Elena lo hizo*"; Elena dice "*Yo no fui y Frida tampoco, pero si yo lo hice, entonces o Diana lo hizo también o estaba en la casa*"; Frida dice "*Yo no fuí, Diana estaba en la casa; si Diana estaba en la casa y lo hizo entonces Elena también lo hizo*". Ninguna de las tres dijo la verdad, ¿ Quién cometió la fechoría ?

A continuación enunciamos los teoremas de correctud y completud refutacional del algoritmo cuya demostración se verá a detalle en clase. La notación $\rhd^\star$ se refiere a la cerradura transitiva y reflexiva de la relación $\rhd$.

Proposición 3 (Corrección de DPLL) *Sea $\mathcal{S}$ un conjunto de cláusulas reducidas y sin tautologias.*
Si

$$\cdot \models_? \mathcal{S} \ \rhd^\star \ \mathcal{M} \models_? \varnothing$$

entonces $\mathcal{S}$ es satisfacible y $\mathcal{M}$ es modelo de $\mathcal{S}$.
Además, si

$$\cdot \models_? \mathcal{S} \ \rhd^\star \ \mathcal{M} \models_? \mathsf{fail}$$

entonces $\mathcal{S}$ es insatisfacible.

Adicionalmente el algoritmo DPLL es refutacionalmente completo de acuerdo a la siguiente

Proposición 4 (Completud refutacional de DPLL) *Sea $\mathcal{S}$ un conjunto de cláusulas reducidas y sin tautologias. Si $\mathcal{S}$ es satisfacible entonces existe un modelo $\mathcal{M}$ tal que*

$$\cdot \models_? \mathcal{S} \ \rhd^\star \ \mathcal{M} \models_? \varnothing$$

y si $\mathcal{S}$ es insatisfacible entonces la construcción de un modelo falla, es decir, existe un modelo $\mathcal{M}$ tal que

$$\cdot \models_? \mathcal{S} \ \rhd^\star \ \mathcal{M} \models_? \mathsf{fail}$$

A. Reducción del Sudoku al problema SAT[3]

En esta sección bosquejamos una codificación de la solución de Sudokus como una instancia del problema SAT.
Consideramos aquí Sudokus que cumplen las siguientes dos propiedades:

1. Tienen una solución única.
2. Pueden resolverse con razonamiento únicamente, es decir, sin búsqueda.

La representación se sirve de 729 variables proposicionales denotadas s_{xyz} con $x, y, z \in \{1, \ldots, 9\}$, cuyo significado es que el número z está en la entrada o celda xy. Debe pensarse el tablero de Sudoku como una matriz de manera que la posición xy denota a la celda en el renglón x y columna y.

[3]Esta sección se basa en el artículo *Sudoku as a SAT Problem* de Inês Lynce y Joël Ouaknine. 9th International Symposium on Artificial Intelligence and Mathematics, January 2006.

La **especificación** es la siguiente:

- Hay al menos un número en cada celda

$$\bigwedge_{x=1}^{9} \bigwedge_{y=1}^{9} \bigvee_{z=1}^{9} s_{xyz}$$

  Esta fórmula nos dice que para cada renglón x y para cada columna y hay un número z en la celda xy.

- Cada número figura a los más una vez en cada renglón

$$\bigwedge_{y=1}^{9} \bigwedge_{z=1}^{9} \bigwedge_{x=1}^{8} \bigwedge_{i=x+1}^{9} (\neg s_{xyz} \vee \neg s_{iyz})$$

  Esta fórmula nos dice que para cada columna y, para cada número z y para cada renglón x, excepto el último, si el número z está en el renglón x (en la celda xy) entonces no sucede que el número z está en alguno de los renglones posteriores a x (los denotados por i).

- Cada número figura a lo más una vez en cada columna

$$\bigwedge_{x=1}^{9} \bigwedge_{z=1}^{9} \bigwedge_{y=1}^{8} \bigwedge_{i=y+1}^{9} (\neg s_{xyz} \vee \neg s_{xiz})$$

  Esta fórmula nos dice que para cada renglón x, para cada número z y para cada columna y, excepto la última, si el número z está en la columna y (en la celda xy) entonces no sucede que el número z está en alguna de las columnas posteriores a y (las denotadas por i).

- Cada número figura a lo más una vez en cada cuadrante de 3×3:

  - Dentro de cada cuadrante, si un número z está en un renglón entonces no está en las celdas siguientes en el mismo renglón. Aquí i, j delimitan las celdas válidas del cuadrante y k se encarga de verificar las columnas siguientes a la de la celda en consideración. Todo dentro del mismo cuadrante.

$$\bigwedge_{z=1}^{9} \bigwedge_{i=0}^{2} \bigwedge_{j=0}^{2} \bigwedge_{x=1}^{3} \bigwedge_{y=1}^{3} \bigwedge_{k=y+1}^{3} (\neg s_{(3i+x)(3j+y)z} \vee \neg s_{(3i+x)(3j+k)z})$$

  - Dentro de cada cuadrante, si un número z está en una columna entonces no está en los renglones superiores ni en la misma columna ni en las columnas siguientes. Aquí i, j delimitan las celdas válidas del cuadrante, k se encarga de verificar los renglones superiores y l verifica la columna actual y las columnas siguientes.

$$\bigwedge_{z=1}^{9} \bigwedge_{i=0}^{2} \bigwedge_{j=0}^{2} \bigwedge_{x=1}^{3} \bigwedge_{y=1}^{3} \bigwedge_{k=x+1}^{3} \bigwedge_{l=1}^{3} (\neg s_{(3i+x)(3j+y)z} \vee \neg s_{(3i+k)(3j+l)z})$$

Las cláusulas anteriores conforman una codificación mínima. Un bonito cálculo combinatorio lleva a concluir que a partir de las fórmulas anteriores la fórmula resultante de la transformación a FNC tendrá 8829 cláusulas de las cuales 81 cláusulas tienen 9 literales, y las restantes 8748 son binarias, es decir, tienen 2 literales.

Adicionalmente se deben considerar las cláusulas unitarias correspondientes a las entradas del Sudoku que ya tienen un número asignado. Al agregar las siguientes cláusulas modelamos la *unicidad* de un número en cada celda:

- Hay a lo más un numero en cada celda.

$$\bigwedge_{x=1}^{9}\bigwedge_{y=1}^{9}\bigwedge_{z=1}^{8}\bigwedge_{i=z+1}^{9}(\neg s_{xyz}\vee\neg s_{xyi})$$

  Esta fórmula nos dice que para cada renglón x, para cada columna y, para cada número no 9, si la celda xy tiene al numero z entonces no tiene a los números mayores que z.

- Cada número figura al menos una vez en cada columna

$$\bigwedge_{y=1}^{9}\bigwedge_{z=1}^{9}\bigvee_{x=1}^{9}s_{xyz}$$

  Está fórmula nos dice que para cada columna y y cada número z, existe un renglón x, tal que z está en la celda xy.

- Cada número figura al menos una vez en cada renglón

$$\bigwedge_{x=1}^{9}\bigwedge_{z=1}^{9}\bigvee_{y=1}^{9}s_{xyz}$$

  Está fórmula nos dice que para cada renglón x y cada número z, existe una columna y, tal que z está en la celda xy.

- Cada número figura al menos una vez en cada cuadrante de 3×3.

$$\bigvee_{z=1}^{9}\bigwedge_{i=0}^{2}\bigwedge_{j=0}^{2}\bigwedge_{x=1}^{3}\bigwedge_{y=1}^{3}s_{(3i+x)(3j+y)z}$$

  Está fórmula nos dice que hay un número z en cada uno de los cuadrantes, los cuales qse generan por las celdas $(3i+x)(3j+y)$.

La fórmula resultante de transformar todas las anteriores consta de 11988 cláusulas, 324 de ellas con 9 literales y las restantes 11664 binarias. Nuevamente debemos agregar cláusulas unitarias para las entradas previamente asignadas del Sudoku.