[Tema 3-Representación del conocimiento]

Conectivas lógicas

P	Q	$\neg P$	$P \wedge Q$	$P \vee Q$	$P \Rightarrow Q$	$P \Leftrightarrow Q$
false	*false*	*true*	*false*	*false*	*true*	*true*
false	*true*	*true*	*false*	*true*	*true*	*false*
true	*false*	*false*	*false*	*true*	*false*	*false*
true	*true*	*false*	*true*	*true*	*true*	*true*

Equivalencias lógicas

$$
\begin{aligned}
(\alpha \wedge \beta) &\equiv (\beta \wedge \alpha) && \text{commutativity of } \wedge \\
(\alpha \vee \beta) &\equiv (\beta \vee \alpha) && \text{commutativity of } \vee \\
((\alpha \wedge \beta) \wedge \gamma) &\equiv (\alpha \wedge (\beta \wedge \gamma)) && \text{associativity of } \wedge \\
((\alpha \vee \beta) \vee \gamma) &\equiv (\alpha \vee (\beta \vee \gamma)) && \text{associativity of } \vee \\
\neg(\neg\alpha) &\equiv \alpha && \text{double-negation elimination} \\
(\alpha \Rightarrow \beta) &\equiv (\neg\beta \Rightarrow \neg\alpha) && \text{contraposition} \\
(\alpha \Rightarrow \beta) &\equiv (\neg\alpha \vee \beta) && \text{implication elimination} \\
(\alpha \Leftrightarrow \beta) &\equiv ((\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)) && \text{biconditional elimination} \\
\neg(\alpha \wedge \beta) &\equiv (\neg\alpha \vee \neg\beta) && \text{De Morgan} \\
\neg(\alpha \vee \beta) &\equiv (\neg\alpha \wedge \neg\beta) && \text{De Morgan} \\
(\alpha \wedge (\beta \vee \gamma)) &\equiv ((\alpha \wedge \beta) \vee (\alpha \wedge \gamma)) && \text{distributivity of } \wedge \text{ over } \vee \\
(\alpha \vee (\beta \wedge \gamma)) &\equiv ((\alpha \vee \beta) \wedge (\alpha \vee \gamma)) && \text{distributivity of } \vee \text{ over } \wedge
\end{aligned}
$$

Validez

Una sentencia es válida si es cierta en todos los modelos. También se conoce como tautología. Por ejemplo:
$A \vee \neg A$ es cierto tanto para A=false como para A=true

El teorema de deducción dice que para dos sentencias a y b, $a \models b$ si $a => b$ es válido. Se comprueba mirando si $a => b$ es cierto en todos los modelos.

Satisfacibilidad

Una sentencia es satisfacible si es cierta en algún modelo. Por ejemplo, para $A \vee B$, existen 3 modelos que son ciertos.

La reducción al absurdo se basa en: $a \models b$ si $a \wedge \neg b$ es insatisfacible.

Métodos de prueba

Los métodos deprueba se dividen en:

- **Aplicación de reglas de inferencia**:
  - Generar nuevas sentencias en base a las antiguas.
  - La prueba es una secuencia de aplicación de reglas inferenciales para llegar a una conclusión. La idea es tomar las premisas o proposiciones previas y aplicar una serie de reglas de inferencia para llegar a una conclusión lógica. Por lo general, una prueba implica la aplicación de varias reglas de inferencia en secuencia para llegar a la conclusión deseada.
  - Se suelen traducir las sentencias a una forma normal.
- **Comprobación de modelos**:
  - Consiste en verificar si un conjunto de proposiciones es verdadero en algún modelo o interpretación. La idea es construir un modelo que satisfaga las proposiciones dadas y verificar si tal modelo existe.
  - Existen diferentes técnicas. Una de las técnicas es la enumeración en una tabla de verdad, que puede ser exponencial en el número de proposiciones. Otras técnicas incluyen el uso de algoritmos de búsqueda heurística en el espacio de modelos, como los algoritmos de mínimo conflicto o el de ascensión de colinas.

Monotonicidad

Se basa en que si $KB \models a$, entonces $KB \land b \models a$. Las sentencias que se infieren sólo se incrementan a medida que se añade información a la base del conocimiento. No se pueden invalidar conclusiones ya inferidas, o sea, que el cambio de opinión no sirve.

Factorización

La cláusula resultante de una regla de resolución debe contener sólo una cláusula de cada literal. La eliminación de las copias se llama factorización. Por ejemplo, $a \lor b$ junto a $a \lor \neg b$ se reduce en a.

Forma normal conjuntiva

Toda sentencia de lógica proposicional equivale a una conjunción de cláusulas. Una sentencia expresada como connción de cláusulas está en FNC.

El algoritmo para convertir una sentencia en una FNC es:

1. Substituír a<=>b por a=>b $\land$ b=>a
2. Substituír a=>b por $\neg$a $\lor$ b

3. Usar De Morgan para las negaciones
4. Usar la propiedad distributiva

Completitud

El cierre de resolución es un conjunto de cláusulas que se obtiene a partir de un conjunto inicial de cláusulas utilizando la regla de resolución de manera repetida. En otras palabras, el cierre de resolución es el conjunto de todas las cláusulas que se pueden deducir a partir del conjunto inicial de cláusulas utilizando únicamente la regla de resolución.

La regla de resolución es una técnica de inferencia que se utiliza en lógica proposicional para derivar nuevas cláusulas a partir de cláusulas existentes. Dado un conjunto de cláusulas, se seleccionan dos cláusulas que contienen una variable proposicional complementaria (es decir, una cláusula contiene una variable y su negación aparece en la otra cláusula) y se realiza una operación de resolución para obtener una nueva cláusula que contiene todas las variables que no se cancelan en las dos cláusulas originales. Este proceso se repite de manera recursiva hasta que no se pueden obtener nuevas cláusulas.

El teorema fundamental de resolución dice que si se puede derivar la cláusula vacía a partir del conjunto inicial de cláusulas, entonces se concluye que el conjunto de cláusulas es insatisfacible. Por otro lado, si no se puede obtener la cláusula vacía a partir del conjunto inicial de cláusulas, entonces el conjunto de cláusulas es satisfacible.

Cláusula de Horn

Se consideran dos formas restringidas de cláusulas:

- **Cláusula definida**: es una disyunción de literales en la que exactamente un literal es positivo.
- **Cláusula de Horn**: es una disyunción de literales en la que a lo sumo un literal es positivo. Todas las cláusulas definidas son cláusulas de Horn, ya que son cláusulas con literales no positivos. Además, las cláusulas de Horn tienen la propiedad de que son cerradas bajo resolución, lo que significa que si se resuelven dos cláusulas de Horn, se obtiene una cláusula de Horn.

Cláusulas definidas

Toda cláusula definida se puede escribir como una implicación. Esto significa que cualquier expresión lógica en forma de cláusula (es decir, una disyunción de literales) puede ser reformulada como una implicación, donde la premisa o cuerpo de la implicación es una conjunción de literales positivos y la conclusión o cabeza de la implicación es un único literal positivo.

Las cláusulas de Horn permiten realizar inferencias eficientes utilizando algoritmos de encadenamiento progresivo o regresivo. Además, la decisión de implicación en las cláusulas de Horn puede realizarse en un tiempo lineal en el tamaño de la base de conocimientos (KB).

Encadenamiento progresivo

Este algoritmo sirve para determinar si un símbolo proposicional deriva de una base de conocimiento de cláusulas definidas. Se hace en tiempo lineal. Los pasos en los que se hace son:

1. Seleccionar una meta: Seleccionar la meta (o conclusión) que se desea alcanzar.
2. Buscar una regla que contenga la meta en su conclusión: Buscar una regla en la base de conocimientos que contenga la meta en su conclusión.
3. Verificar si las condiciones (antecedentes) de la regla se cumplen en la base de hechos: Verificar si todas las condiciones (antecedentes) de la regla están presentes en la base de hechos. Si no se cumplen todas las condiciones, la regla no es aplicable y se busca otra regla.
4. Si todas las condiciones se cumplen, agregar la conclusión de la regla a la base de hechos: Si todas las condiciones de la regla se cumplen, se agrega la conclusión de la regla a la base de hechos.
5. Verificar si se ha alcanzado la meta: Comprobar si se ha alcanzado la meta original. Si no se ha alcanzado, volver al paso 2 y buscar otra regla que pueda ser aplicada.
6. Repetir los pasos 2-5 hasta que se alcance la meta o se agoten las reglas: Repetir los pasos anteriores para todas las reglas aplicables en la base de conocimientos, hasta que se alcance la meta o se agoten todas las reglas.

Por ejemplo:

H1: A

H2: B

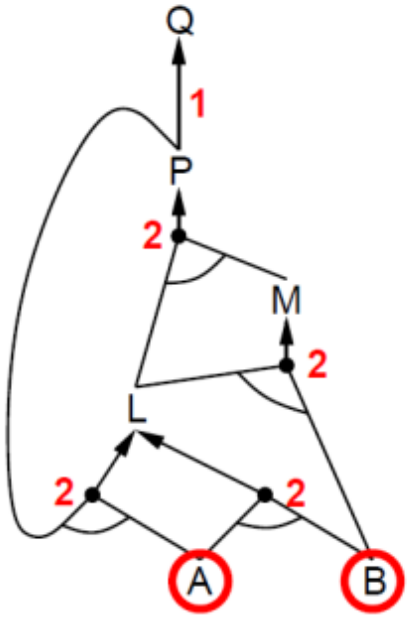
R1: $P \Rightarrow Q$

R2: $L \wedge M \Rightarrow P$

R3: $B \wedge L \Rightarrow M$

R4: $A \wedge P \Rightarrow L$

R5: $A \wedge B \Rightarrow L$



Marcamos las cláusulas A y B e indicamos el número de aristas que llegan a cada nodo.

H1: A

H2: B

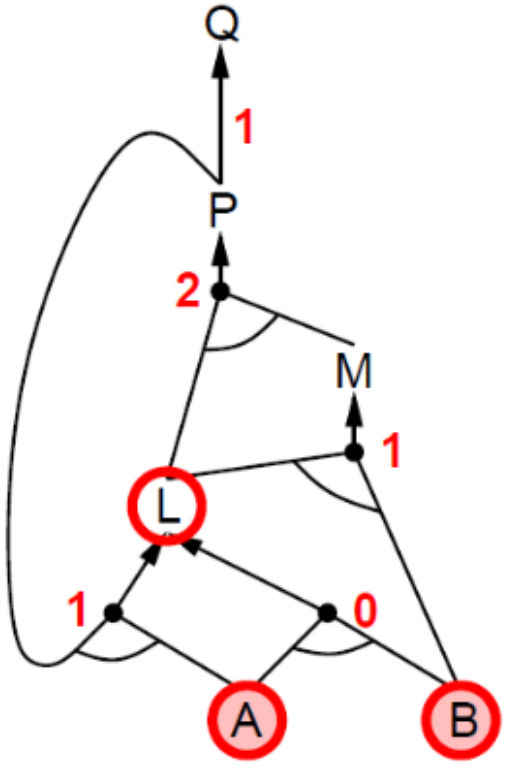
R1: $P \Rightarrow Q$

R2: $L \wedge M \Rightarrow P$

R3: $B \wedge L \Rightarrow M$

R4: $A \wedge P \Rightarrow L$

R5: $A \wedge B \Rightarrow L$



Con A y B obtenemos L y la arista pasa a tener peso 0.

H1: A

H2: B

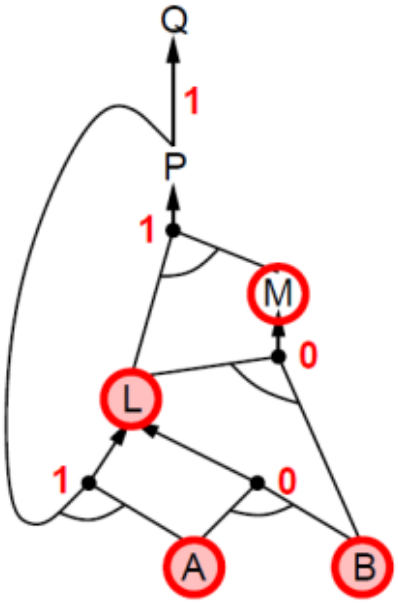
R1: $P \Rightarrow Q$

R2: $L \wedge M \Rightarrow P$

R3: $B \wedge L \Rightarrow M$

R4: $A \wedge P \Rightarrow L$

R5: $A \wedge B \Rightarrow L$



Con B y L obtenemos M y la arista pasa a tener peso 0.

H1: A

H2: B

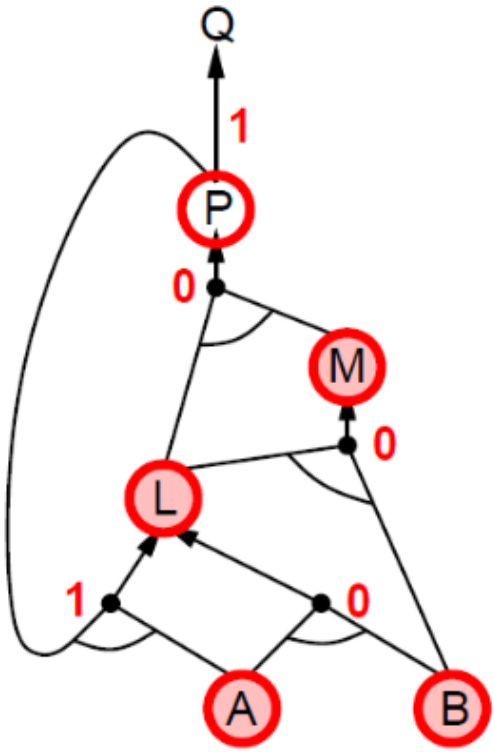
R1: $P \Rightarrow Q$

R2: $L \wedge M \Rightarrow P$

R3: $B \wedge L \Rightarrow M$

R4: $A \wedge P \Rightarrow L$

R5: $A \wedge B \Rightarrow L$



Con M y L obtenemos P y la arista pasa a tener peso 0.

H1: A

H2: B

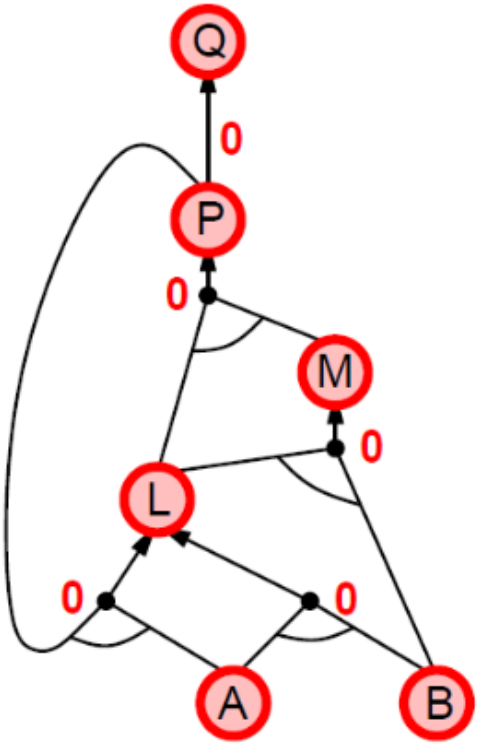
R1: $P \Rightarrow Q$

R2: $L \wedge M \Rightarrow P$

R3: $B \wedge L \Rightarrow M$

R4: $A \wedge P \Rightarrow L$

R5: $A \wedge B \Rightarrow L$



Con P y A obtenemos L. Con P obtenemos Q. Ahora todas las aristas tienen peso 0.

Encadenamiento regresivo

Funciona a la inversa. Se parte, en edste caso, de la consuta Q y se buscan las implicaciones de la base del conocimiento cuya conclusión es Q. Si todas las premisas de cada una de esas implicaciones son ciertas, Q es cierta. Es dirigido por las metas, no como el progresivo, dirigido por los datos.

Por ejemplo:

H1: A

H2: B

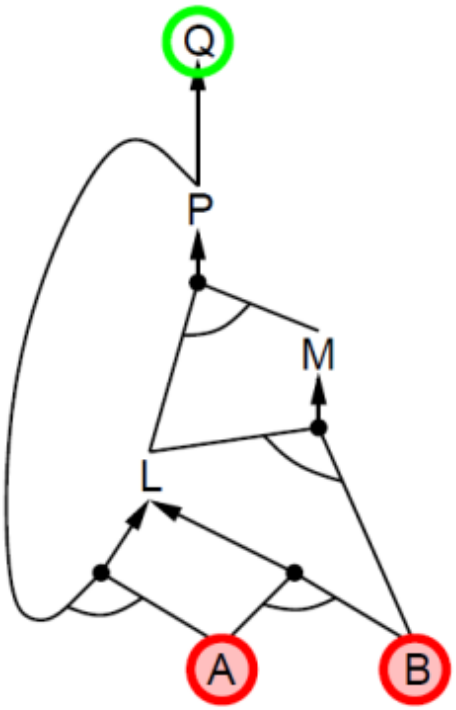
R1: $P \Rightarrow Q$

R2: $L \wedge M \Rightarrow P$

R3: $B \wedge L \Rightarrow M$

R4: $A \wedge P \Rightarrow L$

R5: $A \wedge B \Rightarrow L$



Marcamos Q y las permisas de las que partimos.

H1: A

H2: B

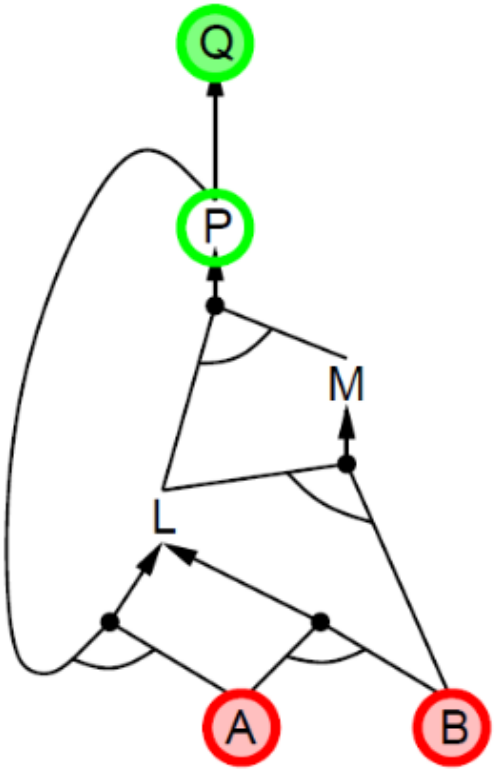
R1: $P \Rightarrow Q$

R2: $L \wedge M \Rightarrow P$

R3: $B \wedge L \Rightarrow M$

R4: $A \wedge P \Rightarrow L$

R5: $A \wedge B \Rightarrow L$



Para obtener Q necesitamos P.

H1: A

H2: B

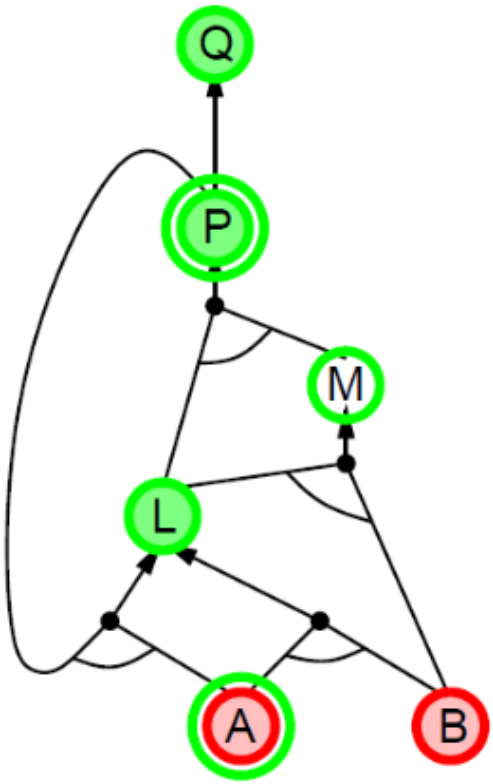
R1: $P \Rightarrow Q$

R2: $L \wedge M \Rightarrow P$

R3: $B \wedge L \Rightarrow M$

R4: $A \wedge P \Rightarrow L$

R5: $A \wedge B \Rightarrow L$



Para obtener P necesitamos L y M o L y A.

H1: A

H2: B

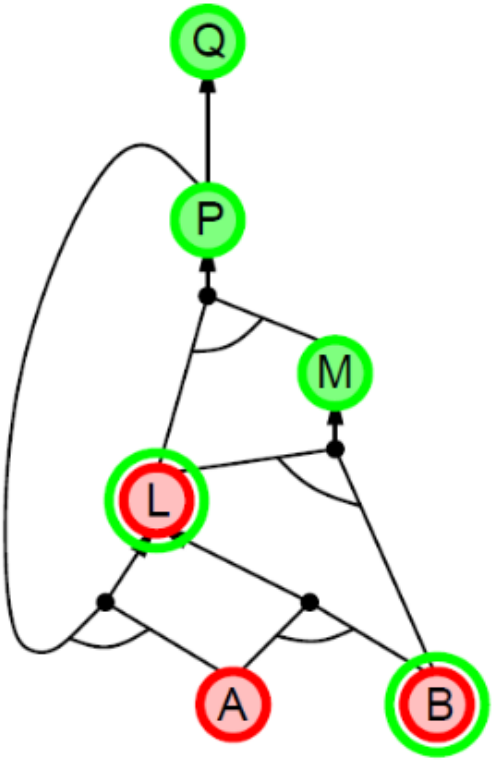
R1: $P \Rightarrow Q$

R2: $L \wedge M \Rightarrow P$

R3: $B \wedge L \Rightarrow M$

R4: $A \wedge P \Rightarrow L$

R5: $A \wedge B \Rightarrow L$



Para obtener M necesitamos L y B.

El encadenamiento progresivo se da por ejemplo en humanos, al tomar una decisión rutinaria. Tiene complejidad lineal. El regresivo se da en resolución de problemas. Por ejemplo, ¿cómo puedo construír una caseta para el perro? Su complejidad suele ser menor que lineal.

Comprobación de modelos proposicional efectiva

El problema SAT (Satisfacibilidad Booleana) se refiere a la pregunta de si existe al menos una asignación de valores de verdad a un conjunto de variables booleanas que satisfaga una expresión booleana dada.

Muchos problemas de computación se pueden reducir a comprobar la satisfacibilidad de una sentencia proposicional.

La escalabilidad SAT se puede ccorregir con:

- **Análisis de componentes**: Este es un enfoque para resolver problemas SAT que implica dividir el problema en subproblemas más pequeños y manejables. En particular, se busca identificar conjuntos de variables y cláusulas que no interactúan entre sí, lo que permite resolverlos de manera independiente.
- **Ordenación de variables y valores**: Esta técnica implica elegir un orden en el que se asignan valores de verdad a las variables en un problema SAT. Se pueden utilizar diferentes heurísticas para determinar el orden, como la heurística de grado, que selecciona primero las variables que aparecen en el mayor número de cláusulas.
- **Backtracking inteligente**: El backtracking es una técnica utilizada en la mayoría de los algoritmos SAT para explorar diferentes asignaciones de valores de verdad. El backtracking inteligente utiliza información obtenida durante la ejecución del algoritmo para tomar decisiones más informadas sobre qué variables asignar y en qué orden. Por ejemplo, el aprendizaje de cláusulas conflicto se utiliza para identificar cláusulas que causan conflictos y agregarlas a la fórmula para evitar problemas similares en el futuro.
- **Recomienzo aleatorio**: Esta técnica implica detener la ejecución del algoritmo SAT después de un cierto número de intentos y comenzar de nuevo desde cero con una asignación aleatoria de valores de verdad. Esto puede evitar que el algoritmo quede atrapado en un subconjunto particular del espacio de soluciones que no tiene solución.
- **Indexación adecuada**: Esta técnica implica organizar los datos del problema SAT de tal manera que la búsqueda y recuperación de información sea lo más eficiente posible. En particular, se pueden utilizar técnicas de indexación para buscar rápidamente cláusulas que contengan ciertas variables o valores de verdad.

Backtracking DPLL

```
function DPLL_SATISFIABLE? (s) returns true or false
        inputs: s,una sentencia en lógica proposicional.
        clauses← el conjunto de cláusulas en la representación CNF de s
        Symbols←una lista de los símbolos proposición en s
        return DPLL (clauses,symbols, {})
```

```
function DPLL (clauses,symbols, model) returns true or false
        if toda cláusula en clauses es cierta en model then return true
        if alguna cláusula en clauses es falsa en model then return false
        P, value ← FIND-PURE-SYMBOL(symbols, clauses,model)
        if P es no-null then return DPLL (clauses,symbols-P, model ∪ {P=value})
         P, value ← FIND-UNIT-CLAUSE(clauses,model)
        if P es no-null then return DPLL (clauses,symbols-P, model ∪ {P=value})
        P ← FIRST(symbols); rest ←REST (symbols)
        return DPLL (clauses,rest,model ∪ {P=true}) or
                DPLL (clauses,rest,model ∪ {P=false})
```

Funciona de la siguiente manera:

1. Simplificación de la fórmula: se aplican las reglas de simplificación para eliminar cláusulas y literales redundantes. Por ejemplo, si una cláusula contiene una literal que siempre es verdadera, entonces esa cláusula puede ser eliminada.
2. Asignación de valores de verdad: se elige una variable que aún no ha sido asignada un valor de verdad y se le asigna uno (verdadero o falso). Se actualiza la fórmula para reflejar esta asignación y se simplifica nuevamente. Si se encuentra una contradicción, se retrocede en el árbol de decisión y se intenta otra asignación.
3. Ramificación: si no se encuentra una contradicción, se elige otra variable no asignada y se repite el proceso de asignación de valores de verdad. Esto se conoce como ramificación.
4. Poda: si se encuentra una contradicción en algún punto del proceso, se retrocede en el árbol de decisión y se realiza una poda. Esto implica deshacer una o más asignaciones anteriores y elegir otro valor para la variable correspondiente.
5. Terminación: el algoritmo continúa ramificando y podando hasta que se encuentra una asignación de valores de verdad que satisface la fórmula o se determina que no existe tal asignación.

Algoritmos de búsqueda local

La ascensión de colinas (HILL-CLIMBING) y el enfriamiento simulado (SIMULATED-ANNEALING) son algoritmos de búsqueda heurística que se pueden aplicar a problemas SAT

para encontrar una solución satisfactoria. Ambos algoritmos parten de una solución inicial y realizan movimientos para tratar de mejorarla.

En la ascensión de colinas, se parte de una solución aleatoria y se generan soluciones vecinas realizando pequeños cambios en la asignación de valores de verdad. Luego, se elige la solución vecina que tiene el valor de evaluación más alto y se repite el proceso hasta que se encuentra una solución satisfactoria o se llega a un máximo local. En este caso, la solución actual se considera la mejor encontrada hasta el momento.

En el enfriamiento simulado, se realiza un proceso similar pero se permite la aceptación de soluciones peores con cierta probabilidad, lo que permite escapar de máximos locales. A medida que se realizan más iteraciones, la probabilidad de aceptar soluciones peores disminuye gradualmente, lo que permite que el algoritmo converja a una solución satisfactoria.

El algoritmo WALKSAT es un ejemplo específico de ascensión de colinas que se puede utilizar para resolver problemas SAT. En este algoritmo, se elige una cláusula falsa al azar y se cambia el valor de verdad de un símbolo en la cláusula elegida. Luego, se evalúa si la nueva solución es mejor que la anterior y se repite el proceso hasta que se encuentra una solución satisfactoria o se alcanza el número máximo de iteraciones permitidas. Si se alcanza el número máximo de iteraciones y no se ha encontrado una solución satisfactoria, se devuelve un fallo.

```
function WALKSAT (clauses,p, max_flips) returns modelo satisafactorio or fallo.
    inputs: clauses,un conjunto de cláusulas en lógica proposicional.
            p, la probabilidad de elegir un "movimiento aleatorio", típicamente sobre 0.5
            max_flips, nº de saltos permitidos antes de rendirse

    model ←una asignación aleatoria de true/false a los símbolos en clauses
    for i=1 to max_flips do
                if  model satisfies clauses then return model
                clause ← una cláusula elegida aleatoriamente de clauses que es falsa en model
                with probability p cambiar el valor en model de un símbolo seleccionado  aleatoriamente de clause.
                else cambiar cualquier símbolo en clause que maximize el nº de cláusulas satisfechas
    return failure
```