3 Problema de la Satisfacibilidad en Lógica Proposicional

3.1 El problema de la satisfacibilidad

Desde el punto de vista de la lógica, tanto si nos preguntamos por la "coherencia" de las oraciones como por las "deducciones" que puede obtenerse a partir de un conjunto de ellas, el problema final se traduce en un problema de satisfacibilidad. En efecto:

- Si queremos estudiar la "coherencia" de las oraciones, no preguntamos por la *satisfacibilidad* de dicho conjunto.
- Si queremos determinar si una cierta oración es consecuencia de otra, o bien estudiamos los *modelos* de las premisas y la conclusión (un modelo no es más que una interpretación de satisfacibilidad) o bien estudiamos la *satisfacibilidad* del conjunto junto con el negado de la supuesta conscuencia.

Así pues, desde un punto de vista computacional, nuestro objetivo está claro ¿cuándo se puede saber que un conjunto es satisfacible? ¿cómo lo puedo determinar? ¿cuál es la forma más rápida? Las respuestas a estas preguntas son soluciones al problema conocido como **problema de satisfacibilidad** o problema SAT. Estrictamente hablando el problema SAT no se plantea sobre cualquier conjunto de oraciones, sino sobre aquellas que tienen un forma o expresión concreta: tienen que ser cláusulas.

- El **problema SAT** es el problema de decisión que responde a la cuestión de si un conjunto de cláusulas es satisfacible.
- Los **problemas de decisión** son problemas que cuestionan una afirmación y se resuelven con un sí o un no.

Si el conjunto de partida no está formado por cláusulas caben dos posibilidades: o bien se resuelve el problema de satisfacibilidad trabajando sobre cualquier conjunto de oraciones o se transforma el conjunto original de oraciones a un conjunto de cláusulas, surgiendo en este segundo caso una cuestión añadida ¿se puede obtener un conjunto de cláusulas que sean equivalentes al conjunto inicial de oraciones?

Además surgen otras cuestiones computacionales. ¿Realmente se puede construir un algoritmo que resuelva el problema de decisión? ¿cuánto tarda en resolver este problema? Encontrar la respuesta a esta pregunta no fue trivial y cuando se obtuvo fue sorprendente: El **problema SAT** define una categoría nueva de problemas, los problemas NP-completos. Son problemas realmente muy difíciles de resolver y son aquellos que "intuitivamente" conllevan realizar un búsqueda de todas las posibles combinaciones de sus elementos para encontrar una solución (lo que puede suponer más de una vida con los computadores actuales), pero sin embargo dada la solución, ésta se puede comprobar muy rápidamente (lo que puede suponer pocos segundos con los computadores actuales).

La NP-completitud del problema SAT fue demostrada por Stephen Cook en 1971 (el Teorema de Cook). Hasta entonces, el concepto de un problema NP-completo, no estaba definido. El problema SAT es el representante de todos los problemas NP-completos. Esto quiere decir que quien resuelva SAT en un tiempo polinomial, habrá resuelto todos los problemas que ya se han identificado dentro de esta categoría. Pero el ser un NP-completo no significa que no se sepa llegar a su solución, solo significa que

se puede tardar demasiado en llegar a ella. Es por ello que existen algoritmos completos y no-completos para resolver SAT (ver más adelante).

3.1.1 Tipos de Problemas

Para saber más ....

Los **problemas de decisión** son problemas que cuestionan una afirmación y se resuelven con un sí o un no. Por ejemplo, la pregunta ¿existe una interpretación que hagan cierta a la oración $p \rightarrow q$? define un problema de decisión, pues se responde con un sí o un no. Dependiendo de cuán difícil es dar respuesta a la pregunta, así se define la dificultad del problema.

Los problemas son:

P Un problema de decisión que pueden ser resuelto en tiempo polinómico . Es decir, dada una instancia del problema, la respuesta sí o no se puede decidir en tiempo polinomial .

Ejemplo: Dado un grafo conectado, ¿pueden sus vértices ser coloreados utilizando dos colores para que ningún arco sea monocromático? La solución se obtiene realizando el siguiente algoritmo: comenzar con un vértice arbitrario, colorearlo de rojo y a todos sus vecinos azul y continuar con el siguiente no coloreado, vecino de los azules. Parar cuando se coloreen todos los vértices o hasta que estés obligado a hacer que un arco tenga sus dos extremos del mismo color.

NP Un problema de decisión donde las instancias del problema para el cual la respuesta es sí tienen demostraciones que se pueden **verificar** en tiempo polinómico. Esto significa que si alguien nos da un ejemplo del problema y una solución para la respuesta "sí", podemos comprobar que es correcta en tiempo polinómico .

Ejemplo: La factorización de enteros es NP. Este problema plantéa que si dados números enteros n y m, ¿hay un número entero f con $1 < f < m$ tal que f divide n (f es un pequeño factor de n)? Se trata de un problema de decisión porque las respuestas son o sí o no. Si alguien nos da un ejemplo del problema (es decir, nos dan enteros concretos n y m) y un entero f, con $1 < f < m$, y afirma que f es un factor de n (el certificado o solución) se puede comprobar la respuesta en tiempo polinomial mediante la realización de la división n/f.

NP-completo Un problema NP, X, para el que es posible reducir cualquier otro problema NP, llamado Y, a X en tiempo polinómico. Intuitivamente esto significa que podemos resolver rápidamente Y si sabemos cómo resolver X rápidamente. Más precisamente, Y es reducible a X si existe un algoritmo de tiempo polinomial f para transformar las instancias y de Y para casos $x = f(y)$ de X en el tiempo polinomio con la propiedad de que la respuesta es sí a y si y sólo si la respuesta a $f(y)$ es sí .

Ejemplo: 3 - SAT. Este es el problema que se nos da una conjunción de cláusulas con 3-literales (es decir, las sentencias son de la forma

$$(x_{v11} \vee x_{v21} \vee x_{v31}) \wedge (x_{v12} \vee x_{v22} \vee x_{v32}) \wedge ...(x_{v1n} \vee x_{v2n} \vee x_{v3n})$$

donde cada x_{vij} es una variable booleana o la negación de una variable de una lista predefinida y finita $(x_1, x_2, ...x_n)$. Se puede demostrar que cada problema NP puede reducirse a 3-SAT. La demostración de esto es muy técnica y requiere del uso de la definición técnica de NP (basado en las máquinas de Turing no deterministas y similares). Esto se conoce como el teorema de Cook.

¿Qué hace que los problemas NP-completos sean tan importantes? Está en que si un algoritmo de tiempo polinomial determinista se puede encontrar para resolver uno de ellos, todos los problemas NP es resoluble en tiempo polinomial.

NP-duro Intuitivamente estos son los problemas que son incluso más duro que los problemas NP-completos . Tenga en cuenta que los problemas NP-duros no tienen que estar en NP (no tienen que ser los problemas de decisión). La definición precisa es que un problema X es NP-duro si hay un problema NP-completo Y tal que Y es reducible a X en tiempo polinómico. Pero ya que cualquier problema NP-completo se puede reducir a cualquier otro problema NP-completo en tiempo polinomial, todos los problemas NP-completos se pueden reducir a cualquier problema NP-duro en tiempo polinomial. Entonces, si hay una solución para un problema NP -duro en tiempo polinómico, hay una solución a todos los problemas NP en tiempo polinómico .

El problema de la parada de máquinas de Turing es el clásico problema NP-duro. Este es el problema en el que dado un programa/algoritmo P y una entrada I, ¿podemos saber si en algún momento se detendrá? Este es un problema de decisión pero no está en NP.

La afirmación **P** = **NP** es el más famoso problema de las ciencias de la computación, y una de las cuestiones pendientes más importantes en las ciencias matemáticas. Está claro que P es un subconjunto de NP. La pregunta abierta es si los problemas NP tienen soluciones en tiempo polinomial determinista. Se cree ampliamente que no lo hacen, pero tampoco se puede afirmar. De hecho , el Instituto Clay ofrece un millón de dólares para una solución al problema.

3.2 Algoritmos para el problema SAT en L0

En L0 el problema SAT es decidible; es decir, podemos encontrar algoritmos completos que nos indicarán si un conjunto de cláusulas es satisfacible o no. A su vez estos se pueden dividir en algoritmos que trabajan con formas clausales y los que no. Desde el punto de vista de completitud se pueden mencionar:

- Algoritmos completos[1].
  Los que proporcionando una asignación, ésta es un modelo.
  - Tablas de verdad.
  - Algoritmo DP (Davis, Martin; Putnam, Hillary. 1960)
  - Algoritmo DPLL (Davis, Putnam, Logemann y Loveland, 1962)
  - Árboles semánticos.
  - Tableros semánticos.
  - Resolución.
  - ...
- Algoritmos no Completos[2].
  - GSAT (Greedy SAT)

[1] Determinan tanto la satisfacibilidad como la insatisfacibilidad.
[2] Puede que no encuentre un modelo, aún existiendo.

- WalkSat (GSAT con probabilidades)
- SA-SAT (Simulating Annealing SAT)
- ...

... para saber más ...
Existen mucho problemas SAT pero el más importante es el problema 3-SAT pues todo problema SAT se puede reducir a un 3-SAT utilizando las siguientes reglas.

- Si una cláusula consta de un único literal ℓ, considerar literales nuevos x_1 y x_2 para sustituir ℓ por la expresión:

$$(\ell \vee x_1 \vee x_2) \wedge (\ell \vee \neg x_1 \vee x_2) \wedge (\ell \vee x_1 \vee \neg x_2) \wedge (\ell \vee \neg x_1 \vee \neg x_2)$$

- Si una cláusula consta de dos literales, $\ell_1 \vee \ell_2$, considerar un literal nuevo x para sustituir la cláusula por la expresión:

$$(\ell_1 \vee \ell_2 \vee x) \wedge (\ell_1 \vee \ell_2 \vee \neg x)$$

- Si una cláusula consta de tres literales dejarla como está.
- Si un cláusula consta de k-literales (cuatro o más literales), $\ell_1 \vee \ell_2 \ldots \ell_k$, considerar un conjunto de literales nuevos $\{x_1, x_2, \ldots, x_{k-3}$ para sustituir la cláusula por la expresión:

$$(\ell_1 \vee \ell_2 \vee x_1) \wedge (\neg x_1 \vee \ell_3 \vee x_2) \wedge (\neg x_2 \vee \ell_4 \vee x_3) \wedge \ldots \wedge (\neg x_{k-3} \vee \ell_{k-1} \vee \ell_k)$$

  Por ejemplo, para $\ell_1 \vee \ell_2 \vee \ell_3 \vee \ell_4 \vee \ell_5$ se considerarían átomos $\{x_1, x_2\}$ y se construiría la expresión:

$$(\ell_1 \vee \ell_2 \vee x_1) \wedge (\neg x_1 \vee \ell_3 \vee x_2) \wedge (\neg x_2 \vee \ell_4 \vee \ell_5)$$

3.2.1 Formas normales conjuntivas

En un párrafo anterior se planteaba la siguiente cuestión: dado un conjunto de oraciones en forma no clausal ¿se puede obtener un conjunto de cláusulas que sean equivalentes al conjunto inicial de oraciones? La respuesta es sí. Esto sí es posible en la lógica proposicional. No lo es en otras lógicas.

El procedimiento para pasar cualquier oración a un conjunto de cláusulas consiste en utilizar la reglas de equivalencia adecuadas para conseguir los siguientes objetivos y en este orden:

1. Eliminar las implicaciones y dobles implicaciones.
2. Reducir el ámbito de las negaciones: Si una negación afecta a una expresión compleja obtener expresiones equivalente hasta que aparezcan los negados en una expresión literal.
3. Aplicar la distributividad hasta conseguir una conjunción de disyunciones.

¿Podrías dar nombre y escribir cuáles son las reglas de equivalencia que permiten estas transfromaciones?

Observa que estudiar la satisfacibilidad de la oración obtenida al final del proceso anterior, que es una conjunción de cláusulas, es el mismo problema que el de estudiar la satisfacibilidad de un conjunto de cláusulas: el conjunto formado por las cláusulas de la forma normal conjuntiva.

Así pues, si en vez de tener una oración, se considera un conjunto de oraciones de partida:

- Podemos repetir el proceso anterior para cada una de las oraciones que no están en forma normal conjuntiva
- Cada oración dará como resultado un conjunto de cláusulas.
- Estudiar la satisfacibilidad del conjunto de todas las oraciones de partida es equivalente a estudiar la satisfacibilidad del conjunto de todas las clásulas obtenidas por el punto anterior.

En consecuencia, dado cualquier conjunto de oraciones, mediante la transformación a forma normal conjuntiva, podemos construir un conjunto de cláusulas (equivalente al primero) sobre el que podemos aplicar algoritmos que resuelvan el problema SAT.

3.3 Algoritmo DPLL

El algoritmo DPLL fue presentado en 1962 por Martin Davis, Hilary Putnam, George Logemann y Donald W. Loveland y es una refinación del previo algoritmo de Davis-Putnam (1960). Es un algoritmo que trabaja sobre un conjunto de cláusulas.

Mostramos con un ejemplo cómo funciona. Supón que tienes estas dos oraciones y preguntamos por la satisfacibilidad de cada una por separado: $\varphi = (p \vee q \vee \neg t) \wedge (\neg q \vee \neg r \vee s) \wedge (q \vee t)$ y $\xi = \varphi \wedge r$ ¿qué debería ocurrir para que sean satisfacibles? Bueno, por definición de conjunción todas las cláusulas deben ser ciertas y para que cada una de ellas sea cierta al menos uno de sus literales debe ser cierto. Por tanto,

- Si al exigir que una cláusula sea cierta el valor de sus literales nos viene "forzado" (no queda más remedio que adopte dicho valor) entonces suponer que se está dando dicho valor.

  Por ejemplo, en la expresión ξ, para que la oración sea cierta, el literal r deberá ser cierto (necesariamente), por lo que se tendrá que **exigir** que $v_I(r) = V$.

- Si en la exigencia no se puede determinar el valor de sus literales (hay varias combinaciones de verdad de sus literales para hacer la cláusula cierta), entonces considerara uno de sus literales y suponer que es cierto.

  Por ejemplo, no está claro para ninguna de las cláusulas de φ el valor de verdad de sus literales, parece necesario estudiar las posibles interpretaciones; pero podemos **suponer** que cierto literal sea cierto y seguir estudiando la situación bajo ese contexto, digamos que se puede seleccionar el r y suponer que la oración es satisfacible bajo el supuesto $v_I(r) = V$.

Sea como fuere, con dicho valor dado o dicho valor supuesto estamos fijando la asignación de un literal de la oración lo que determina de forma única una oración que es equivalente a la primera que además es una expresión más pequeña.

Por ejemplo, si en φ se exige que $v_I(q) = V$ dicha oración es equivalente a $\varphi(q)$:

$$\varphi_{|v_I(q)=V} \stackrel{def}{=} (p \vee V \vee \neg t) \wedge (\neg V \vee \neg r \vee s) \wedge (V \vee t) \equiv \varphi(q) \stackrel{def}{=} \neg r \vee s$$

Así pues, dada una oración en forma normal conjuntiva, al exigir que sea satisfacible podemos considerar el valor de verdad de uno de sus literal (ya se porque nos viene dado o porque lo suponemos) lo que nos conduce a un oración mucho más simple y equivalente a la primera (en condiciones de interpretación que hace a la oración de partida).

En esto consiste DPLL, en obtener expresiones expresiones cada vez más pequeñas obtenidas por la "propagación (del valor de verdad) de un literal". Primero se propaga un literal, se obtiene una expresión más simple que la primera, después se propaga otro literal, para obtener una expresión más sencilla que la segunda, se propaga un tercer literal para tener una expresión más reducida aún, y así sucesivamente. Observa que todas son equivalente bajo la exigencia de satisfacibilidad y que además en cada propagación de literal se está realizando una asignación con lo que en cada paso se construye parcialmente una interpretación. El resultado que se espera al final del proceso es una interpretación (conjunto de asignaciones todos los átomos) en la que la evaluación de partida quede totalmente evaluada con el valor V - pues buscamos satisfacibilidad.

Sin embargo, durante el proceso puede que nos encontremos con situaciones "extrañas". Imagina que después de sucesivas propagaciones se obtiene la oración $\xi = p \wedge \neg p$. Si exigimos satisfacibilidad, cada una de las cláusulas debe ser cierta.

- Si nos fijamos en la primera cláusula p, tendremos que exigir que $v_I(p) = V$, lo que nos da la oración: $\xi_{|v_I(p)=V} \stackrel{def}{=} V \wedge \neg V \equiv F \stackrel{def}{=} \xi(p)$
- Y si nos fijamos en la segunda cláusula $\neg p$, deberá cumplirse $v_I(p) = F$, lo que nos da la oración: $\xi_{|v_I(p)=F} \stackrel{def}{=} F \wedge \neg F \equiv F \stackrel{def}{=} \xi(\neg p)$

En cualquier caso, bajo la exigencia de satisfacibilidad de ξ se concluiría que $\xi \equiv F$.

Es decir, puede darse que partiendo de una oración φ bajo el supuesto de que es satisfacible $v(\varphi) = V$ se concluya que la oración insatisfacible F es equivalente a φ **considerando la interpretación obtenida durante el proceso**.

Como no es posible que una oración satisfacible sea equivalente a una expresión insatisfacible, algo está fallando. ¿En qué nos estamos equivocando? La respuesta en simple, en suponer que la interpretación obtenida durante el proceso hace cierta a la oración.

Para evitar el problema lo que hemos de hacer es construir otra interpretación diferente y ver si la oración es satisfacible (justificado por esa otra interpretación). Es decir, hemos de "deshacer" las propagaciones realizadas.

- Empezaríamos "despropagando" el último literal ℓ para propagar el literal $\neg\ell$. Es decir, si con $v(\ell) = V$ se llegó a que la oración de partida bajo este supuesto se comporta igual que una contradicción, probemos con $v(\ell) = F$.

- Si seguimos obteniendo la oración F, es porque el "fallo" puede encontrarse en la penúltima propagación: descartamos cualquier propagación del último literal para "despropagar" el penúltimo literal ℓ' para a continuación propagar el literal $\neg\ell'$.

  En este punto completaríamos la interpretación propagando el último literal, ya sea bajo el supuesto de $v(\ell) = V$ o de $v(\ell) = F$.

- Si seguimos obteniendo la oración F, es porque entonces hubo una mala elección del antepenúltimo literal: descartamos cualquier propagación del penúltimo y del último literal para "despropagar" el antepenúltimo literal ℓ'' y volver a propagar el literal $\neg\ell''$.

  Para completar la interpretación propagaríamos el penúltimo y el último literal con asignaciones que garantices que la oración es satisfacible.

- El proceso se repite una y otra vez si la oración final sigue siendo F. A las malas, puede que tengamos que "despropagar" el primer literal y probar con su otro valor de verdad para repetir todo el proceso.

Observa que a las muy muy malas, puede que en todas las posibles propagaciones la oración final sea siempre F. Es decir, en todas las posibles interpretaciones la oración final siempre es F. En esto caso la oración de partida no le queda otra posiblidad que ser insatisfacible.

Esta es la ideal del algoritmo: por "fuerza bruta" (búsqueda exhustiva) encontrar una posible interpretación donde la oración sea cierta. Para formalizar sus pasos, conocer su nombres y orden de ejecución consulta las transparencias.

3.4 Algoritmo de Resolución

Este algoritmo que también trabaja con formas clausales se basa en el silogismo disyuntivo.

$$\{\gamma \vee \ell, \Gamma \vee \neg\ell\} \models \gamma \vee \Gamma$$

que particularizado a dos clausales se tiene:

Dadas dos cláusulas $\alpha = \ell_1 \vee \ell_2 \vee \ldots \vee \ell_n$ y $\beta = \ell'_1 \vee \ell'_2 \vee \ldots \vee \ell'_m$ donde se cumple que un literal de α es el negado de un literal de β, sin pérdida de generalidad podemos suponer que $\ell_1 = \neg\ell'_1$, entonces

$$\{\ell_1 \vee \ell_2 \vee \ldots \vee \ell_n,\ \ell'_1 \vee \ell'_2 \vee \ldots \vee \ell'_m\} \models \ell_2 \vee \ldots \vee \ell_n \vee \ell'_2 \vee \ldots \vee \ell'_m$$

La expresión de la derecha recibe el nombre de resolvente de α y de β respecto del literal ℓ y se denota por $R_\ell(\alpha, \beta)$.

El algoritmo de resolución en L0 establece que si se "enfrentan" todas las cláusulas entre sí incluidas todas las resolventes obtenidas hasta que no se obtengan nuevas resolventes, entonces el conjunto de partida es satisfacible. Por contra si en ese "enfrentamiento" se obtiene la cláusula vacía el conjunto es insatisfacible.

La cláusula vacía surge al enfrente un literal y su opuesto. Es decir cuando se tiene

$$\{\ell, \neg\ell\} \models \square$$

Observa que al calcular la resolvente se obtiene una nueva cláusula quitando los literales contrarios. Como en este caso cada cláusula consta de un solo literal, la cláusula resolvente (la resultante) no tiene letras: es la cláusula vacía y se denota por $\square$. La cláusula vacía siempre es insatisfacible.

3.5 Árboles semánticos

La técnica de los árboles semánticas puede verse como el algoritmo DPLL cuando la oración de partida no está en Forma Normal Conjuntiva. Al igual que DPLL podemos determinar todas las interpretaciones que hacen a una oración cierta. Un árbol semántico asociado a una expresión ϕ es una árbol donde:

- a cada nodo no hoja se le asocia una letra u de la fórmula ϕ y dos nodos hijos con ramas etiquetas.

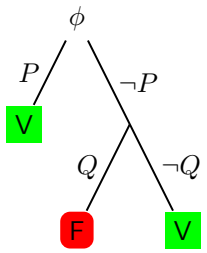
  Una rama representa la asignación del valor V a la letra u asociada y la otra rama representa la asignación del valor F a la letra u. Para indicar estas asignaciones se etiquetan las ramas con u y $\neg u$.

- Cada nodo hoja (las que se encuentran al final) representa el valor de verdad de la expresión ϕ para la interpretación realizada en el camino desde la raíz hasta este nodo. Los nodos hojas se etiqueta con la evaluación.

Imagina que la tabla de verdad de una expresión ϕ es:

P	Q	ϕ
V	V	V
V	F	V
F	V	F
F	F	V

Entonces su árbol semántico es:

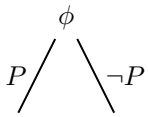


El árbol se ha construido como te indico:

1. Inicialmente se considera el nodo cuya raíz es ϕ y de las dos letras atómicas que tiene ϕ se ha considerado P.

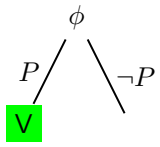
   Así pues, deberán generarse dos ramas desde el nodo raíz, una etiquetada con P para indicar que se hace la asignación $v(P) = V$ y otra etiquetada con $\neg P$ para la asignación $v(P) = F$.

   Lo que genera el grafo:

2. Si nos vamos por la rama de la izquierda, para la asignación $v(P) = V$, observamos en la tabla de verdad que ϕ siempre es cierta.

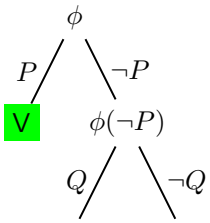
Con lo que generamos un grafo con un nodo hoja (el verde):



3. Ahora vamos a la rama de la derecha, para la asignación $v(\neg P) = V$, y observamos en la tabla de verdad que ϕ puede ser cierta o puede ser falsa. Todo depende de la asignación de Q.

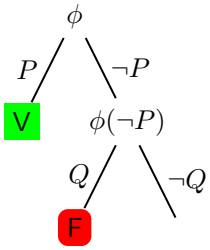
   Bueno, pues analizamos cada caso. Para la rama de la derecha generamos dos nuevas ramas, una etiquetada con Q para indicar que se hace la asignación $v(Q) = V$ y otra etiquetada con $\neg Q$ para la asignación $v(Q) = F$.

Lo que genera el grafo:



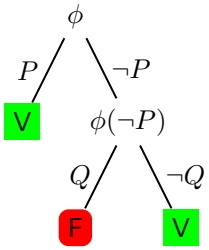
4. Ahora nos vamos a la rama de la izquierda, la que se corresponde con la interpretación $v(\neg P) = v(Q) = V$, y que según la tabla de verdad hace a ϕ falsa:

Con lo que generamos un grafo con un segundo noda hoja (el rojo):



5. Por último nos vamos a la rama de la derecha, la que se corresponde con la interpretación $v(\neg P) = v(\neg Q) = V$, y hace a ϕ cierta.

Con lo que finalizamos el grafo:



Observando el árbol podemos decir que ϕ es satisfacible para las siguientes interpretaciones:

- Cuando $v(P) = V$, sin importar el resto de las asignaciones.
- Cuando $v(\neg P) = v(\neg Q) = V$,

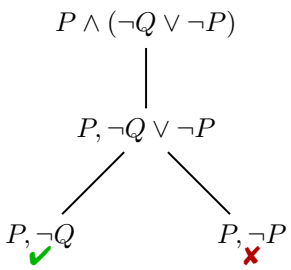
¿Serías capaz de escribir el algoritmo general de obtención de grafos semánticos?

3.6 Tableaux semánticos en L0

Un tableaux semántico es un árbol donde cada nodo representa a las subexpresiones de la expresión de partida a los que hay que asignar el valor V para que la expresión original también sea V.

Conforme se desarrolla el árbol, las expresiones son más pequeñas hasta que en los nodos hojas las expresiones son solo literales. Como cada nodo representa las expresiones que deben ser ciertas, los nodos hoja determinan las interpretaciones donde la oración de partida es cierta.

Por ejemplo, el tableaux semántico de la expresión $\alpha = P \wedge (\neg Q \vee \neg P)$ es



Lo que significa que para que $v(\alpha) = V$ (nodo raíz) se tienen que cumplir que $v(P) = V = v(\neg Q \vee \neg P)$ que es el nodo intermedio. A su vez se tiene que cumplir una de estas dos posibilidades (porque hay dos nodos hoja): o bien, $v(P) = v(\neg Q) = V$, o bien $v(P) = v(\neg P) = V$. Ahora bien, como las lógicas clásicas no permiten este segundo caso, la única posibilidad es considerar la interpretación $v(P) = V$ y $v(Q) = F$ para que la oración de partida sea cierta. De ahí que al nodo formado por $\{P, \neg Q\}$ se marque con el símbolo ✔ para indicar que es una interpretación correcta y posible, y al nodo formado por $\{P, \neg P\}$ se marque con el símbolo ✘ para indicar que es una interpretación contradictoria y no posible.

Los tableaux semántico en $L0$ construye el grafo catalogando una de las oraciones de un nodo a "expandir" como de tipo conjunción o de tipo disyunción. Si se cataloga de tipo conjunción del nodo se considerará un nodo hijo que contendrán nuevas expresiones más simples. Si se cataloga de tipo disyunción se considerarán dos nodos hijos que a su vez contendrán nueva expresiones más sencillas.

Una oración de tipo conjunción se denomina α-fórmula y una de tipo disyunción se denomina β-fórmula y las oraciones más sencillas que generan y la relación entre tableaux semánticos y satisfactibilidad de las expresiones se puede mirar en las transparencias de clase.

3.7 Razonamiento automático

Con las técnicas anteriores es fácil diseñar programas que "razonen" automáticamente. Solo hay que tener en cuenta que: $\{\alpha_1, \alpha_2, \ldots, \alpha_n\} \models \beta$ sii

- $\alpha_1 \wedge \alpha_2 \wedge \ldots \wedge \alpha_n \rightarrow \beta$ es tautología.
- $\alpha_1 \wedge \alpha_2 \wedge \ldots \wedge \alpha_n \wedge \neg\beta$ es contradicción.

hechos que ya sabemos comprobar usando tablas de verdad, árboles semánticos o tableux.

O si se prefiere, usando formas normales conjuntivas, comprobando que

- $(\alpha_1 \wedge \alpha_2 \wedge \ldots \wedge \alpha_n \rightarrow \beta)_{FNC}$ es tautología.
- $(\alpha_1 \wedge \alpha_2 \wedge \ldots \wedge \alpha_n \wedge \neg\beta)_{FNC}$ es contradicción.

Hecho que se pueden contrastar utilizando DPLL o resolución.