Satisfactibilidad y Validez

Semestre 2011-2

Dándole una razón práctica de ser a la lógica proposicional.

Tan siquiera, en mi opinión (Caleb).

- Cuando un enunciado es true con tan siquiera un modelo.
 - Se dice que el modelo m satisface al enunciado P.
- ¿Cómo encontramos a m?

- Algoritmo 1:
 - Probar todos los posibles modelos y ver si uno hace a P true.

Validez

- Cuando un enunciado P es true para todo modelo.
- ¿Cómo?

Validez

- Algoritmo 1 (de nuevo)
 - Probar todos los posibles modelos y ver si todos hacen a P true.

¿Mejores ideas?

Problemas de Satisfactibilidad

- Muchos problemas se pueden expresar como una lista de restricciones.
 - Su respuesta es el conjunto de valores de las variables del problema tal que dichas restricciones se satisfacen.

- Agendar el horario laboral a empleados de un hospital:
 - Algunos no trabajan de noche.
 - Ninguno puede trabajar más de X horas a la semana.
 - Algunas personas no pueden trabajar con otras

Otro Ejemplo

- Encontrar "bugs" a programas:
 - Se entrega una especificación lógica del programa
 - Controlador de tráfico aéreo
 - Se entrega una posible situación
 - "Dos aviones están en la pista al mismo tiempo"
 - ¿Se pueden satisfacer estas dos cosas?
 - Si sí, hay un problema con la especificación del programa.

- Usualmente (casi siempre), los problemas de satisfactibilidad se escriben de forma:
 - (CLÁUSULA) \((CLÁUSULA) \((CLÁUSULA) \)
- CLÁUSULA es un requisito, en la forma de:
 - (LITERAL) v (LITERAL)
- LITERAL es una opción que satisface el requisito, en la forma de una variable o su negación:
 - P, Q, ¬P, etc.

- $(PVQ)\Lambda(\neg PVR)$
- $(Rv \neg Qv \neg P) \wedge (\neg Sv P)$

• $(Q \wedge \neg P) \wedge (P \vee S \wedge R)$

- Todo enunciado en Lógica Proposicional se puede escribir en Forma Normal Conjuntiva.
 - CNF en corto (Conjunctive Normal Form)

- Para convertir:
 - Eliminar ⇒ y ⇔ utilizando ∧ y V.
 - Metemos ¬ dentro de paréntesis con las leyes De Morgan.
 - Distribuir ν sobre Λ.

- $(P \vee Q) \Rightarrow (R \Rightarrow S)$
- Eliminamos "flechas": ¬(PvQ)v(R⇒S)
- $\neg (PVQ)V(\neg RVS)$
- Metemos "negación": (¬P Λ ¬Q) ν (¬R ν S)
- Distribución de ν: (¬Pν(¬RνS)) Λ(¬Qν(¬RνS))
- Final: (¬Pv¬RvS) \(\(\tau\)(¬Qv¬RvS)

Otro Ejemplo

• $(P \wedge Q) \Leftrightarrow (\neg R \wedge Q)$

Aunque todo enunciado en Lógica
 Proposicional se puede convertir a CNF, puede crecer exponencialmente en tamaño.

Más Ejemplos

- (¬Q v P) v (¬R)
- $(\neg P \wedge Q) \wedge (R \wedge \neg S)$

Algunos Detalles Más...

- Cláusula vacía es False.
 - No hay opciones a satisfacer.
- Enunciado vacío es True.
 - No hay requisitos.
- Enunciado que contiene una cláusula vacía es falsa.
 - Requisito imposible.

Ahora sí...

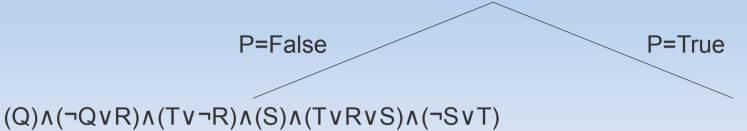
¿Dado un enunciado CNF, como podemos probar que es satisfactible?

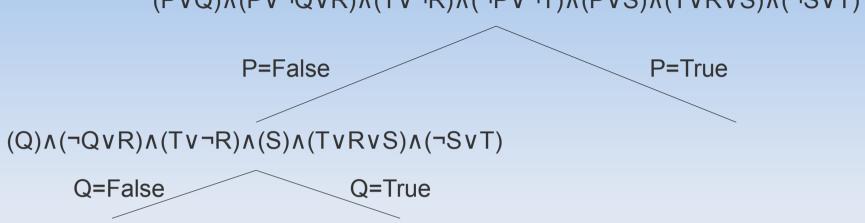
- Algoritmo 1:
 - Probar todos los posibles modelos y ver si uno hace a P true.
 - El número posible de modelos crece exponencialmente con el número de variables.

Algoritmo 2:

- Crear un árbol de búsqueda donde en cada nivel consideramos ambos valores (True, False) para cada variable (P).
- En una rama, asumir que P es True:
 - Borrar las cláusulas que contengan P (son satisfechas).
 - Remover ¬P del resto de las cláusulas (no es una opción).
- En la otra rama, que es False.
 - Lo inverso: borrar cláusulas con ¬P, y remover P.
- Proseguir con el resto de las variables.

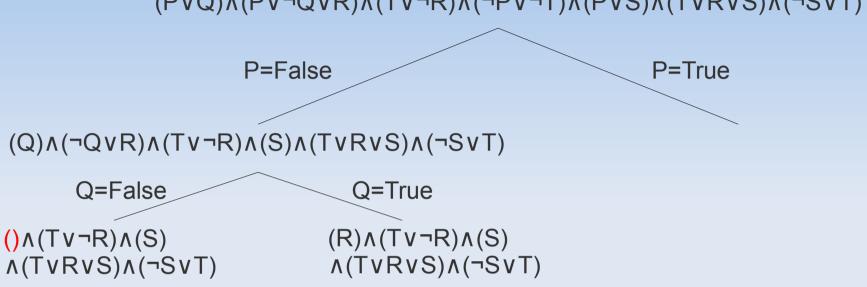




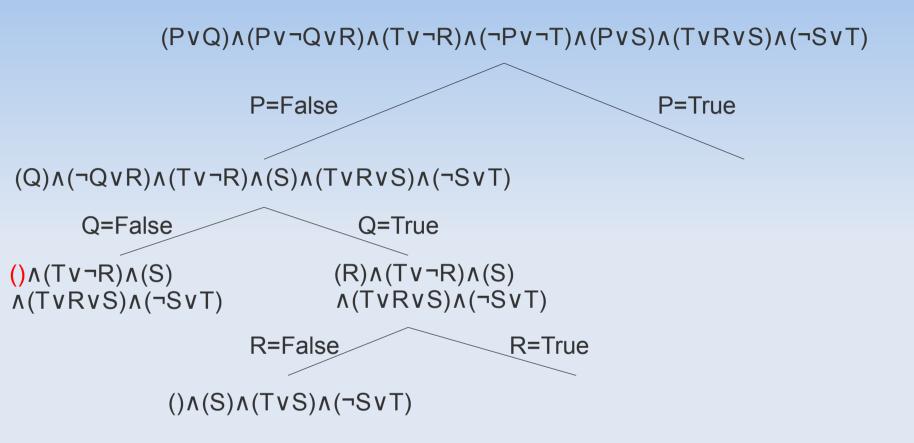


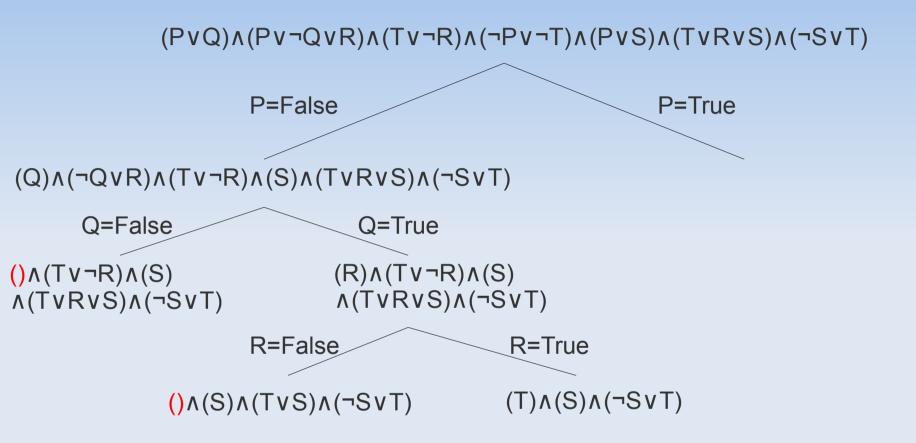
 $(PVQ)\Lambda(PV\neg QVR)\Lambda(TV\neg R)\Lambda(\neg PV\neg T)\Lambda(PVS)\Lambda(TVRVS)\Lambda(\neg SVT)$

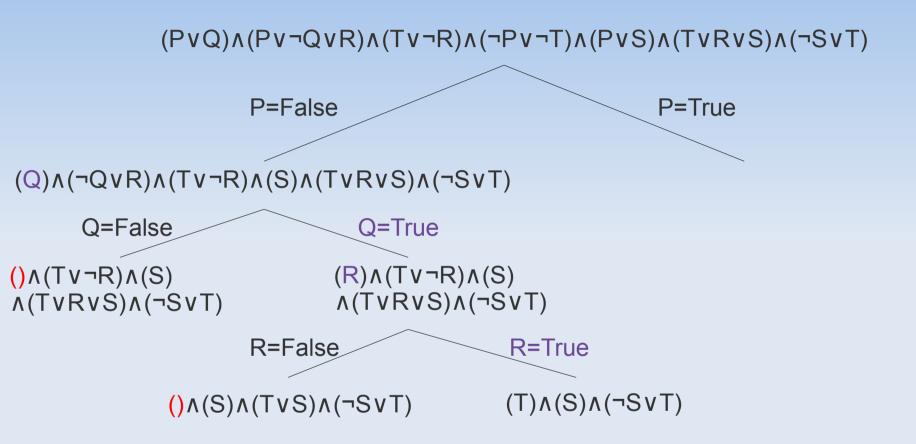
 $P=False \qquad \qquad P=True \\ (Q)\Lambda(\neg QVR)\Lambda(TV\neg R)\Lambda(S)\Lambda(TVRVS)\Lambda(\neg SVT) \\ Q=False \qquad \qquad Q=True \\ ()\Lambda(TV\neg R)\Lambda(S) \\ \Lambda(TVRVS)\Lambda(\neg SVT)$

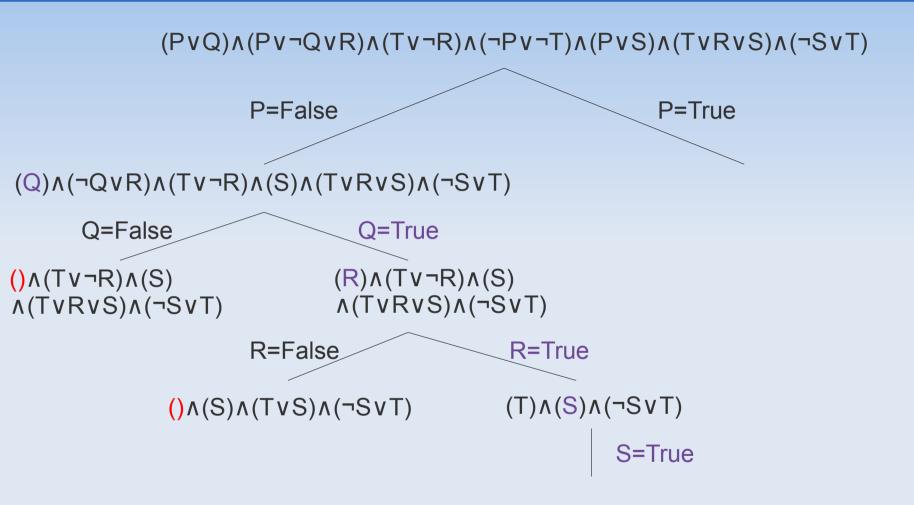


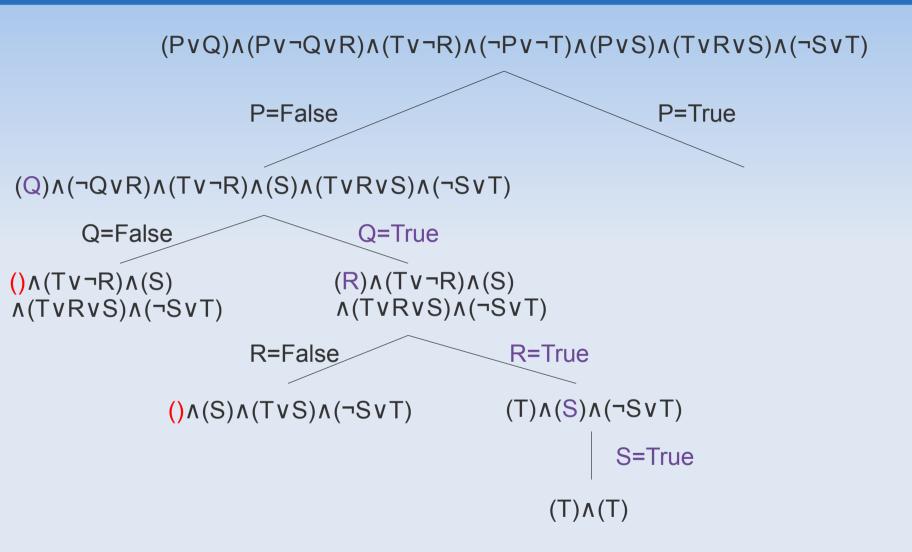
 $(PVQ)\Lambda(PV\neg QVR)\Lambda(TV\neg R)\Lambda(\neg PV\neg T)\Lambda(PVS)\Lambda(TVRVS)\Lambda(\neg SVT)$ P=False $(Q)\Lambda(\neg QVR)\Lambda(TV\neg R)\Lambda(S)\Lambda(TVRVS)\Lambda(\neg SVT)$ Q=False Q=True $()\Lambda(TV\neg R)\Lambda(S)$ $\Lambda(TVRVS)\Lambda(\neg SVT)$ R=False R=True

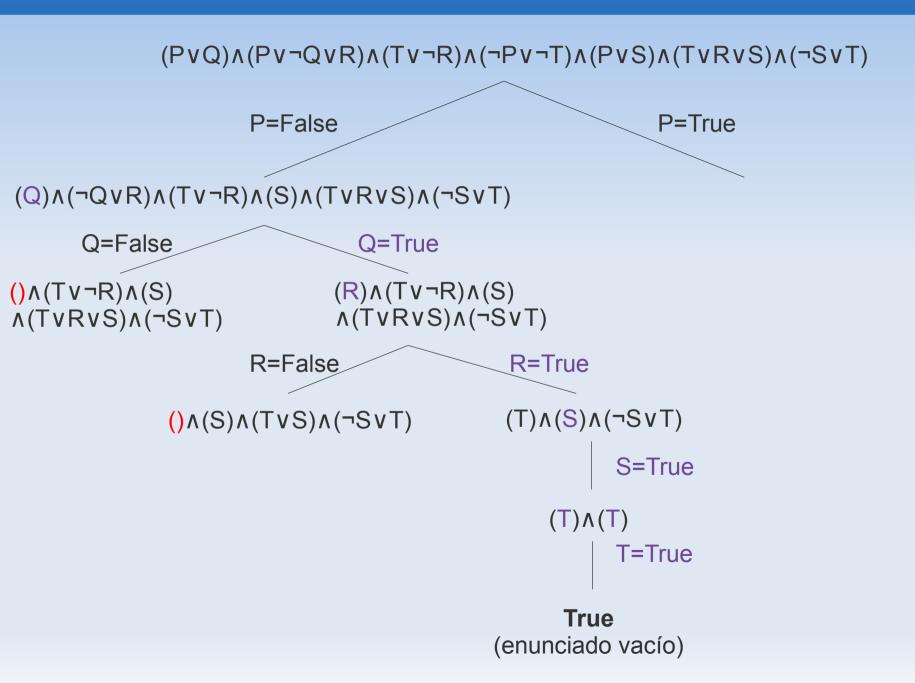




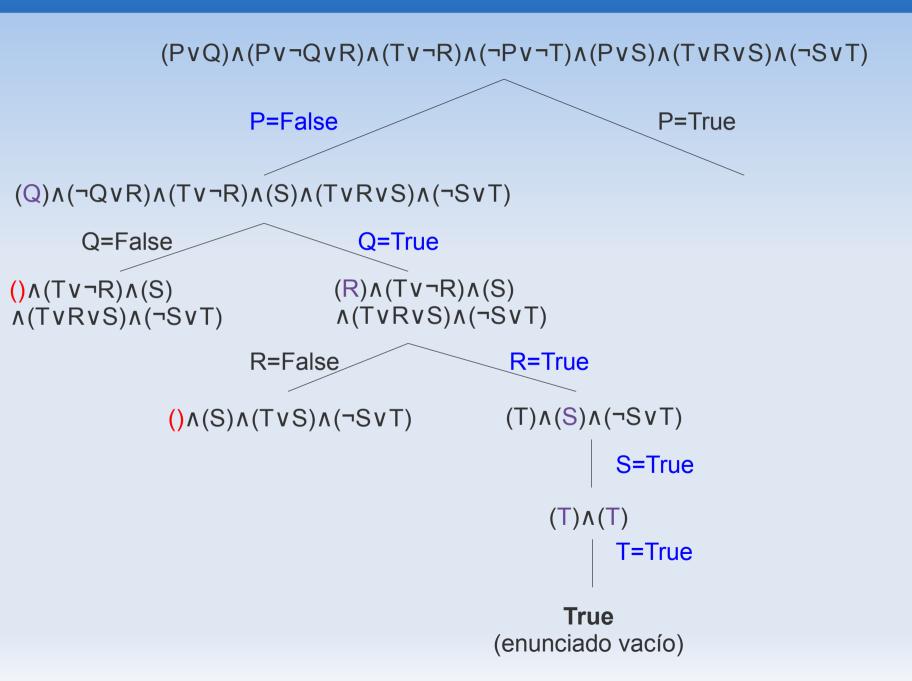








Resultado



Otro Ejemplo (Literal Puro)

- $(TV\neg X)\Lambda(\neg SVT)\Lambda(SV\neg X)$
 - T aparece sólo sin negar.
 - X aparece sólo negado.
 - Ambos son "puros".
- Si T=True: (Sv¬X)
- Si X=False: (¬SvT)
- Ambos simplifican significativamente al enunciado.

DPLL

- Algoritmo creado por:
 - Davis, Putnam, Logeman y Loveland
- Trabaja recursivamente, con un enunciado CNF (E) como entrada.

DPLL a Detalle

- Si E está vacío, regresa True.
- Si hay una cláusula vacía en E, regresa False.
- Si hay una cláusula con un sólo literal U en E, regresa DPLL(E(U=True)). Cláusula Unitaria
- Si hay un literal puro (ocurre sólo sin negar o negado)
 Q en E, regresa DPLL(E(Q=True))
- De otra manera: escoge un literal V al azar y:
 - Si DPLL(E(V=True)) es True, regresa True.
 - Si no, regresa DPLL(E(V=False)).

Discusión

¿Es DPLL más eficiente que listar todos los posibles modelos de E y revisar si satisfacen a E?

Discusión

 ¿Cómo se escogería más "inteligentemente" a V, en vez de ser al azar?

Heurística MOMS

- Maximum number of Occurrences, Minimum
 Sized clauses
- Escoge la variable más instanciada:
 - En (TvX)Λ(¬SvT)Λ(SvXvT), sería T
- ¿Por qué?

Heurística MOMS

- Escoge variables altamente restringida.
- Así, si se va a fallar, que se falle pronto.

¿Es Lo Mejor que Podemos Hacer?

¿Qué tan correcto es un resolvedor SAT?

- Sensatez: si es que regresa una respuesta, ésta siempre es correcta.
- Completo: siempre regresa una respuesta.

DPLL

- ¿Es sensato?
- ¿Es completo?

• ¿Es rápido?

Rapidez

- En práctica, se ha observado que si el algoritmo completo, usualmente también es lento.
- Relativamente hablando...

GSAT

- Algoritmo de "mejora iterativa".
- Basado en el algoritmo de búsqueda: Hill-Climbing.
- Iterar n veces:
 - Escoger un modelo al azar: M.
 - Iterar m veces:
 - Cambiar de valor a la variable que cuesta menos cambiarle su valor.
 - Si el costo resultante es cero, regresar M.
- Costo: número de cláusulas insatisfechas en el enunciado.

GSAT

- ¿Es sensato?
- ¿Es completo?

• ¿Es lento?

GSAT vs DPLL

- DPLL es sensato y completo.
- GSAT es sensato, pero no completo.
 - No puedes utilizarlo efectivamente para generar todas los posibles modelos satisfactadores.
- En competencias de SAT, GSAT le estaba ganando a DPLL, hasta que los desarrolladores encontraron mejores heurísticas.

GSAT vs DPLL

- Cook y Mitchell discuten que:
 - Problemas con pocas cláusulas (restricciones), conocidos como "Débilmente Restringidos" (Weakly Constrained) son fáciles para ambos.
 - Muchas soluciones
 - "Altamente Restringidos" (Highly Constrained) son fáciles para DPLL, pero no para GSAT.
 - Pocas soluciones, el proceso de simplificación en DPLL omite conjuntos grandes de modelos no satisfactadores.
 - Para GSAT, es como una aguja en un pajar.
 - Cualquier cosa en medio, es difícil para ambos.

¿Qué pensamos de GSAT ahora?

WalkSAT

- Como GSAT, pero con "ruido" adicional.
- Iterar n veces:
 - Escoger un modelo al azar: M.
 - Iterar m veces:
 - Escoge al azar una cláusula no satisfecha U.
 - Con probabilidad p=0.5:
 - O cambiar la variable con menos costo en C.
 - O cambiar una variable al azar en C.
 - Regresar M si el costo es cero.
- Costo: el mismo que en GSAT.

WalkSAT

- La razón tras cambiar el valor a variables al azar es:
 - De vez en cuando, es necesario "empeorar" las cosas para encontrar situaciones que, potencialmente, nos lleven a "mejorar".
- Empíricamente, se ha encontrado que hace mejor trabajo que GSAT, pero se le ha atribuido "poca ciencia" a esta técnica.

¿Cuál les gusta más?

Siguiente Clase

- La van a dar ustedes.
- Se formarán dos equipos:
 - Uno estudiará el resolvedor GRASP.
 - El otro estudiará el resolvedor CHAFF.
- Darán una presentación de 20 a 30 minutos:
 - Explicar el resolvedor que les tocó.
 - Explicar otro resolvedor encontrado en la web (de artículo: PDF al maestro y copias a la clase).
 - Compararlos (puntos extras si muestran resultados empíricos) y argumentar cuál es el que se prefiere.