SAT

Satisfacibilidad de una fórmula proposicional

Instituto de Computación - Facultad de Ingeniería

Agenda

- Breve definición del problema del SAT
- Importancia del problema del SAT
- Problemas NP-Completos
- Una solución “simple”
- Métodos
- Codificaciones en lógica proposicional

Lógica proposicional

Sintaxis

Qué es una fórmula:

Vocabulario consiste de un conjunto P de variables proposicionales

Denotadas por $p, q, r, \ldots$ (ev. con subíndice)

Definimos el conjunto de fórmulas proposicionales sobre P

I. Toda variable proposicional es una fórmula.

II. Si F y G son fórmulas:

i. $\neg F$ es una fórmula

ii. $(F \wedge G)$ es una fórmula

iii. $(F \vee G)$ es una fórmula

Lógica proposicional

Semántica

Damos valor de verdad a una fórmula interpretando las variables proposicionales

I: Vars --> {0,1}

Definimos una valuación val(F):

i. $val(p) = I(p)$

ii. $val(\neg F) = 1 - val(F)$

iii. $val(F \wedge G) = \min\{val(F), val(G)\}$

iv. $val(F \vee G) = \max\{val(F), val(G)\}$

Lógica proposicional

Satisfacción, modelo

I satisface F , $I \models F$ si $val_I(F) = 1$

Decimos que I es un **modelo** de F

Ejemplo:

$F = (p \wedge (\neg p \vee q))$

Sea I1

I1(p) = 1, I1(q) = 0

val(F) = min(1,max((1-1),0) = min(1,0) = 0

I1 no satisface F

Lógica proposicional

Satisfacción, modelo

I satisface F , $I \models F$ si $val_I(F) = 1$

Decimos que I es un **modelo** de F

Ejemplo:

$F=(p \wedge (\neg p \vee q))$

Sea I2

I2(p) = 1, I2(q) = 1

val(F) = min(1,max((1-1),1) = min(1,1) = 1

I2 satisface F

Lógica proposicional

Representación en SAT

Principio del palomar: Tenemos m casillas y n palomas, Deseamos asignar palomas a casillas de modo que toda paloma esté asignada y ninguna casilla tenga más de una paloma.

Consideremos el problema para n=3, m=2

Nuestras variables son $p_{i,j}$, i є {1,2,3}, j є {1,2}

Lógica proposicional

Representación en SAT

Nuestras variables son $p_{i,j}$, i є {1,2,3}, j є {1,2}

Representamos las restricciones

1.- Cada paloma está en alguna casilla:

$(p_{1,1} \lor p_{1,2}) \land (p_{2,1} \lor p_{2,2}) \land (p_{3,1} \lor p_{3,2})$

Lógica proposicional

Representación en SAT

Nuestras variables son $p_{i,j}$, $i \in \{1,2,3\}$, $j \in \{1,2\}$

Representamos las restricciones:

1.- Cada paloma está en alguna casilla:

$(p1,1 \lor p1,2) \land (p2,1 \lor p2,2) \land (p3,1 \lor p3,2)$

2.- Ninguna paloma está en más de una casilla:

$(\neg p1,1 \lor \neg p1,2) \land (\neg p2,1 \lor \neg p2,2) \land (\neg p3,1 \lor \neg p3,2)$

3.- Cada casilla tiene a lo sumo una paloma:

$\neg(p1,1 \land p2,1) \land \neg(p1,1 \land p3,1) \land \neg(p2,1 \land p3,1) \land$

$\neg(p1,2 \land p2,2) \land \neg(p1,2 \land p3,2) \land \neg(p2,2 \land p3,2)$

Lógica proposicional

Representación en SAT

Nuestras variables son $p_{i,j}$, i є {1,2,3}, j є {1,2}

Representamos las restricciones

Nuestra fórmula es el and de todas las restricciones

$(p1,1 \lor p1,2) \land (p2,1 \lor p2,2) \land (p3,1 \lor p3,2) \land$

$(\neg p1,1 \lor \neg p1,2) \land (\neg p2,1 \lor \neg p2,2) \land (\neg p3,1 \lor \neg p3,2) \land$

$\neg(p1,1 \land p2,1) \land \neg(p1,1 \land p3,1) \land \neg(p2,1 \land p3,1) \land$

$\neg(p1,2 \land p2,2) \land \neg(p1,2 \land p3,2) \land \neg(p2,2 \land p3,2)$

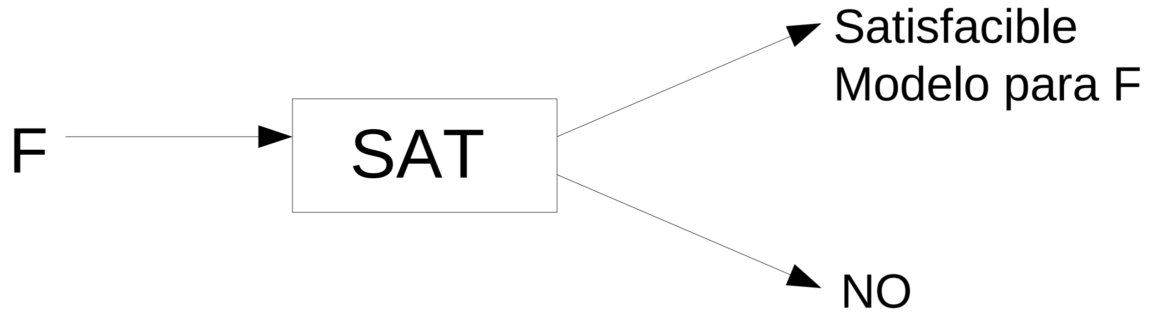
NO debe tener modelo

Forma Normal Conjuntiva

- Una fórmula está en Forma Normal Conjuntiva (FNC) si es una
  - conjunción
    - de disyuciones
      - de literales
- Ejemplo: $p \wedge (q \vee \neg r)$
- La FNC es una forma clausal
- Un programa Prolog es una fórmula (1er orden) en FNC.

El problema SAT

- SAT : Satisfacibilidad de una fórmula proposicional.
  - Dada una fórmula F de la Lógica Proposicional (habitualmente en FNC):



Expresividad de SAT

- Si deseamos saber si una fórmula F es siempre verdadera:

$SAT(\neg F) = NO$ F es una tautología

- Si deseamos saber si $G \models F$

$SAT(G \wedge \neg F) = NO$

Expresividad de SAT

- En muchos casos nos interesa que la fórmula sea satisfactible.

- El modelo que SAT nos devuelve es la solución a un problema que codificamos en una fórmula proposicional.

- Ejemplos:
  - Planificación
  - Optimización de código
  - Búsqueda en IA, juegos

Clase de complejidad NP

NP = Problemas de decisión resolubles por una máquina de Turing no determinista con una cota polinómica en la cantidad de pasos

NP = Problemas con orden polinómico de chequeo de una conjetura

SAT $\in$ NP porque:

- Una valuación V puede ser conjeturada en |F| pasos
- Testear si V $\models$ F es polinómico en el tamaño de F

Problemas NP-Completos

(Cook, The Complexity of Theorem Proving Procedures, 1971)

- SAT es el primer problema para el cual se demostró que es NP-Completo
- O sea, todos los problemas de la clase NP reducen polinomialmente a SAT
- La demostración se hace mediante la reducción de una máquina de Turing no determinista polinómica a SAT.
- A partir de ahí se demuestra la NP-Completitud de muchos porblemas reduciendo a SAT

2 puntos importantes:

- La noción de reducción, la "universalidad" de SAT
- La equivalencia en complejidad de distintos problemas

Algunos problemas NP-COMPLETOS

SAT (Problema de satisfacibilidad booleana, para fórmulas en forma normal conjuntiva)

- 0-1 INTEGER PROGRAMMING (Problema de la Programación lineal entera)
- CLIQUE (Problema del clique)
- DIRECTED HAMILTONIAN CIRCUIT (Problema del circuito hamiltoniano dirigido)
- UNDIRECTED HAMILTONIAN CIRCUIT (Problema del circuito hamiltoniano no dirigido)

Wikipedia

Significado de NP-Completitud

- No se conoce un algoritmo polinómico para un problema NP-Completo
- Los mejores algoritmos tienen tiempo exponencial (peor caso).
- Sin embargo, en la práctica, el peor caso no suele ocurrir.

- Los algoritmos actuales de SAT resuelven instancias de millones de cláusulas con cientos de miles de variable en segundos.
- Pero, pueden fracasar ante instancias "difíciles" de tamaño similar o menor.

SAT

- 1er método simple
- Pasaje a forma normal conjuntiva
  - Equivalencias lógicas
  - Tseitin
- DPPL

SAT: Método “simple”

Consideremos la fórmula

$$F := (p \wedge q) \vee \neg(\neg p \wedge (q \vee \neg r))$$

¿F es satisfacible?

p	q	r	F
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

SAT: Método “simple”

Consideremos la fórmula

$F := (p \wedge q) \vee \neg(\neg p \wedge (q \vee \neg r))$

¿F es satisfacible?

p	q	r	F
0	0	0	0
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

SAT: Método “simple”

Consideremos la fórmula

$F := (p \wedge q) \vee \neg(\neg p \wedge (q \vee \neg r))$

¿F es satisfacible?

p	q	r	F
0	0	0	0
0	0	1	1
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

SAT: Método "simple"

Consideremos la fórmula

$F := (p \wedge q) \vee \neg(\neg p \wedge (q \vee \neg r))$

¿F es satisfacible?

SAT !!
Modelo: 0,0,1

p	q	r	F
0	0	0	0
0	0	1	1
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

SAT: Método "simple"

Consideremos la fórmula

$F := (p \wedge q) \vee \neg(\neg p \wedge (q \vee \neg r))$

¿F es satisfacible?

INSAT: Construir toda la tabla
N variables : 2^N filas

p	q	r	F
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

SAT: Método “simple”

Consideremos la fórmula

$F := (p \wedge q) \vee \neg(\neg p \wedge (q \vee \neg r))$

¿F es satisfacible?

INSAT: Construir toda la tabla
N variables : 2^N filas

N puede ser muy grande !!

p	q	r	F
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

Complejidad de SAT “simple”

Si M es el largo de la fórmula y N la cantidad de variables

$M\ 2^N$ es una cota de la complejidad del algoritmo.

La expresión en lógica proposicional de un problema real requiere de una gran cantidad de variables y cláusulas (la lógica proposicional es un lenguaje de *bajo nivel*).

Complejidad de SAT “simple”

Si M es el largo de la fórmula y N la cantidad de variables

$M\ 2^N$ es una cota de la complejidad del algoritmo.

Otros métodos y heurísticas permiten bajar en la práctica sensiblemente los tiempos, pero la cota teórica sigue siendo exponencial.

Instancias fáciles y difíciles

- Aunque todos los algoritmos tienen tiempo exponencial en el peor caso, se logran mejores comportamientos en
  - Problemas sub-restringidos (SAT, en general)
  - Problemas sobre-restringidos (NO-SAT, en general)

SAT y CNF

La mayor parte de los algoritmos suponen que la fórmula está en CNF

ya que:

- Los métodos tienden a ser más simples

- Algún método requiere CNF (resolución)

El resultado de expresar un problema en lógica proposicional no siempre queda en CNF.

Parte de la resolución es entonces pasar a CNF

Pasaje a CNF

- Por equivalencias lógicas
- Tseitin
- A partir de la tabla de verdad (DNF y CNF)

Pasaje a CNF

- Por equivalencias lógicas

1. Aplicar las 3 siguientes reglas exhaustivamente

$\neg\neg F \Rightarrow F$

$\neg(F \wedge G) \Rightarrow \neg F \vee \neg G$

$\neg(F \vee G) \Rightarrow \neg F \wedge \neg G$

2. Aplicar distributiva exhaustivamente

$F \vee (G \wedge H) \Rightarrow (F \vee G) \wedge (F \vee H)$

Pasaje a CNF

- Por equivalencias lógicas

Ejemplo:

$F = (p \wedge q) \vee \neg(\neg p \wedge (q \vee \neg r))$

Se pueden aplicar simplificaciones

$p \wedge p \Rightarrow p$

$p \vee p \Rightarrow p$

Eliminar clásulas que contengan $p \vee \neg p$

El proceso es muy costoso, la fórmula CNF resultante puede tener un tamaño exponencial en función de la fórmula original.

Pasaje a CNF: Tseitin

$F = (p \wedge q) \vee \neg(\neg p \wedge (q \vee \neg r))$

e_1

$e_1 \leftrightarrow e_2 \vee e_3$

$e_2 \leftrightarrow p \wedge q$

$e_3 \leftrightarrow \neg e_4$

$e_4 \leftrightarrow e_5 \wedge e_6$

$e_5 \leftrightarrow \neg p$

$e_6 \leftrightarrow q \vee \neg e_7$

$e_7 \leftrightarrow \neg r$

Cada una de las subfórmulas se traduce a CNF sin explosión combinatoria.

Pasaje a CNF: Tseitin

En la práctica se usan variaciones de Tseitin.

La fórmula CNF resultante tiene las siguientes propiedades:

- Es satisfacible sii la original lo es.
- Todo modelo proyectado a las variables originales es un modelo de la fórmula original
- Todo modelo de la fórmula original puede ser completado a un modelo de la fórmula en CNF

Por lo tanto, no se pierden ni se agregan modelos en el pasaje.

DPLL

Inspirado en

Robert Nieuwenhuis, Albert Oliveras, Cesare Tinelli. *Solving SAT and SAT Modulo Theories: From an abstract Davis–Putnam–Logemann–Loveland procedure to DPLL(T)*. J. ACM 53(6): 937-977 (2006)

DPLL : Davis–Putnam–Logemann–Loveland

- Primera versión (Davis and Putnam) en 1960: problemas de memoria (10)
- Segunda versión (Davis, Logemann and Loveland) en 1962: (10)
  Depth-first-search con backtracking
- Mejoras al 2000, DPLL más eficiente: (100K)
  GRASP, SATO, Chaff, MiniSAT

DPLL

Dada F en CNF, DPLL trata de construir un modelo M para F, de forma incremental.

M se representa como una secuencia de variables.

p – la variable p es verdadera en el modelo

p' – la variable p es falsa en el modelo

Ejemplo**:** $M(p) = 1$, $M(q) = 0$, $M(r) = 1$ es $pq'r$

Las secuencias pueden tener literales de decisión, representados l^d

Un literal de decisión es un posible punto de backtracking, se elige uno de los 2 valores posibles y luego se puede analizar según el otro.

DPLL

Dada F en CNF, DPLL trata de construir un modelo M para F, de forma incremental.

Se introduce un sistema de transiciones para modelar DPLL.

Los estados del sistema son de la forma M || F

Ejemplos:

$pq'r^{d}t \mid\mid (p \lor \neg q) \land (r \lor \neg t) \land (s \lor \neg p)$

$\Phi \mid\mid (p \lor \neg q) \land (r \lor \neg t)$

El sistema se especifica por reglas que indican que transiciones pueden ocurrir.

DPLL

Dada F en CNF, DPLL trata de construir un modelo M para F, de forma incremental.

Se introduce un sistema de transiciones para modelar DPLL.

Estado inicial:

Φ || F , El modelo es vacío, F es la fórmula para SAT

Estados finales:

fail --> NO-SAT

M contiene todas las variables de F --> Se encontró un modelo

DPLL : Reglas extensión

Extensión del modelo:

UnitProp (propagación unitaria)

$M \parallel F, C \vee l \Rightarrow M l \parallel F, C \vee l$ *(C eventualmente vacío) si*

- $M \models \neg C$ *y*
- l no está definido en M

Decisión

$M \parallel F \Rightarrow M\, l^{d} \parallel F$ *si*

- l o $\neg l$ occurre en F *y*
- l no está definido en M

DPLL : Reglas reparación

Reparación del modelo :

Fail

$M \parallel F, C \Rightarrow fail$ si

- $M \models \neg C$ y
- M *no contiene literales de* decisión

Backtrack

$M\ l^d\ N \parallel F, C \Rightarrow M\ \neg l \parallel F, C$ *si*

- $M\ l^d\ N \models \neg C$ *y*
- N *no contiene literales de decisión*

DPLL : Ejemplos

$\Phi \,||\, \neg 1 \vee 2,\ \neg 3 \vee 4,\ \neg 5 \vee \neg 6,\ 6 \vee \neg 5 \vee 2 \Rightarrow$

$\Phi \,||\, \neg 1 \vee 2 \vee 3,\ 1,\ \neg 2 \vee \neg 3,\ \neg 2 \vee 3,\ 2 \vee 3,\ 2 \vee \neg 3 \Rightarrow$

SAT, otros puntos para mejorar

- **Mejorar propagación unitaria / backtracking (80% de los ciclos!)**
- **Heurísticas para ordenar las variables (aparecer en más cláusulas,p.ej.)**
- **Análisis de conflictos: ante una contradicción, ver cuales son las variables responsables. En vez de backtracking, backjumping (**
- **Aprendizaje de cláusulas: Agregar nuevas clásulas para bloquear sub-asignaciones mala.**
- **Restart: desde 0, eventualmente manteniendo las clásulas aprendidas.**

Codificación

Problemas expresados por restricciones (CSP)

Ejemplos:

Palomar

Cuadrado latino

N-Reinas

Codificación

Problemas expresados por restricciones (CSP)

Variables $v_1, v_2, \ldots, v_n$

Dominio finito $dom(v_i)$ para cada variable

Restricciones:

- Combinaciones de valores prohibidas

- Combinaciones de valores permitidas

Codificación: variables

- Problemas expresados por restricciones (CSP)
- Variables $v_1, v_2, ..., v_n$
- Dominio finito $dom(v_i)$ para cada variable
- Codificación directa: **variables**
- 1 var.proposicional por cada var. en cada valor
- $x_{v,i}$ -- verdadera si la var. v tiene el valor i

Codificación: cláusulas

3 clases de cláusulas

1- Cada variable toma al menos un valor

2- Ninguna variable toma más de un valor

3- Cláusulas de conflicto, dependen del problema

Codificación: ejemplo

Representación en SAT

Nuestras variables son $p_{i,j}$, i є {1,2,3}, j є {1,2}

Representamos las restricciones:

1.- Cada paloma está en alguna casilla:

$(p1,1 \lor p1,2) \land (p2,1 \lor p2,2) \land (p3,1 \lor p3,2)$

2.- Ninguna paloma está en más de una casilla:

$(\neg p1,1 \lor \neg p1,2) \land (\neg p2,1 \lor \neg p2,2) \land (\neg p3,1 \lor \neg p3,2)$

3.- Cada casilla tiene a lo sumo una paloma:

$\neg(p1,1 \land p2,1) \land \neg(p1,1 \land p3,1) \land \neg(p2,1 \land p3,1) \land$

$\neg(p1,2 \land p2,2) \land \neg(p1,2 \land p3,2) \land \neg(p2,2 \land p3,2)$

Referencias

Handbook of satisfiablity, Ed by Biere et al.

IOS press , 2009

Chap. 2 y 3

http://www.satcompetition.org/

http://www.satisfiability.org/