Lógica Computacional y Demostración Automática

Carlos Areces
areces@loria.fr
http://www.loria.fr/~areces

INRIA Nancy Grand Est, France

Diciembre 2008

El Curso

- En el curso vamos a estudiar varios lenguajes lógicos (lógica proposicional, lógicas híbridas, lógicas para la descripción, lógica de predicados) pero desde un punto de vista computacional.
- Nos interesa en particular estudiar distintos métodos para determinar cuando una fórmula es satisfacible (i.e., cuando existe un modelo para una fórmula dada). Aunque también discutiremos otras tareas de inferencia (e.g., model checking).
- También nos interesa saber cuan complejos son estos problemas (NP, PSPACE, indecidible).
- Requisitos: Aunque la mayor parte del curso es autocontenida (i.e., voy a dar todas las definiciones necesarias para entender que estamos haciendo), asumo conocimientos básicos de lógica, algoritmos y complejidad.

El Curso

- Miércoles: Lógica Proposicional
  - + Método de David-Putnam
  - + Métodos Incompletos
  - + zchaff y walksat
- Jueves: Lógicas Híbridas
  - + Model Checking
  - + mcheck
- Viernes: Lógicas para la Descripción
  - + Método de Tableaux
  - + racer
- Sábado: Lógica de Predicados
  - + Método de Resolución
  - + spass

Lo que hacemos hoy

- Consulta Popular: Que saben de lógica?
- Lógica Proposicional
  - Una aplicacion simple
  - Aplicaciones más interesantes
  - Métodos completos
  - El método David-Putnam (DP)
  - Métodos incompletos
  - El Algoritmo Greedy
  - El Algoritmo GSAT

Que sabe usted de lógica?

- $\vee, \wedge, \neg, \rightarrow$
- Fórmula Satisfacible, Fórmula Válida (Tautología)
- Tablas de verdad, Método de Tableaux, Método de Resolución, Método de David-Putnam
- $\forall x, \exists x$
- Unificación
- $\Box, \Diamond$
- $@_i\varphi$, $\downarrow x.\varphi$
- $\forall R.\varphi$, $\sqcap$, $\sqcup$, $\sqsubseteq$

Lógica Computacional = Logica + Computadoras

- La Lógica nació como parte de la filosofía:
  - en sus orígenes (allá por la Grecia clásica) la lógica era usada para modelar el proceso de razonamiento humano
  - y para ayudar a derivar inferencias correctas
- Las cosas cambiaron con la llegada de la computadora
  - En realidad, la lógica jugó un papel fundamental en el desarrollo de las computadoras tanto en lo teórico (e.g., nociones de computabilidad) como en lo práctico (e.g., circuitos lógicos)
  - En este curso, vamos a estudiar como la Ciencia de la Computación contribuye directamente al área de Lógica

Por que los lógicos necesitamos computadoras?

- Bueno, por empezar, somo humanos y por lo tanto vagos. Para que hacer el trabajo si alguien más puede hacerlo por nosotros?
- Pero aún aquellos raros ejemplares de lógicos energéticos necesitan ayuda: algunos de los problemas que queremos resolver son simplemente demasiado complejos para hacer sin una computadora
- A veces es necesario chequear millones de posibilidades para verificar que un sistema satisface una determinada propiedad.
- Vamos a ver que, aun usando computadoras, tenemos que utiliar buenos algoritmos o todo el tiempo del mundo no nos alcanzaría.

Lógica Proposicional

- Como todos sabemos la lógica proposicional es fácil:
  Algunos simbolos proposicionales: p_1, p_2, p_3, ...
  Dos símbolos lógicos: $\neg$, $\vee$
  Dos símbolos sintácticos: (,)
- También la semántica es simple:
  $\neg\varphi$ es verdadera sii φ es falsa
  $\varphi \vee \psi$ es verdadera sii φ o ψ son verdaderas
- Dada una asignación V de valores de verdad (verdadero o falso) para todos los símbolos propocisionales podemos determinar el valor de verdad de de cualquier formula respecto de V.

Un problema del corazón

Al lógico cordobés Ceferino le preguntaron: salís con Ana, con Beatriz o con Celeste? Él pensó:
Salgo al menos con alguna de las tres. Si salgo con Beatriz pero no con Ana, entonces salgo con Celeste. O salgo con Ana y con Celeste, o no salgo con Beatriz. Si salgo con Ana, entonces también salgo con Beatriz.
Con quién sale Ceferino?

- Podemos modelar el problema usando Logica Proposicional?
- Que ganamos si lo hacemos?
- Que tipo de preguntas podemos hacerle a nuestro modelo?

Formalizando el Problema

Tres simbolos proposicionales

$A \equiv$ salgo con Ana $\quad \neg A \equiv$ no salgo con Ana
$B \equiv$ salgo con Beatriz $\quad \neg B \equiv$ no salgo con Beatriz
$C \equiv$ salgo con Celeste $\quad \neg C \equiv$ no salgo con Celeste

- Salgo al menos con alguna de las tres.
  $(A \vee B \vee C)$
- Si salgo con Beatriz pero no con Ana, entonces salgo con Celeste.
  $(B \wedge \neg A) \rightarrow C = (\neg B \vee A \vee C)$
- O salgo con Ana y con Celeste, o no salgo con Beatriz.
  $(A \wedge C) \vee \neg B = (A \vee \neg B) \wedge (C \vee \neg B)$
- Si salgo con Ana, entonces también salgo con Beatriz.
  $A \rightarrow B = (\neg A \vee B)$

Resolviendo el Problema

- Que podemos deducir?

$$\frac{(A \vee B \vee C) \quad (\neg A \vee B)}{B \vee C}$$

- Una consequencia de lo que nos dijo Ceferino es que sale al menos con Beatriz o con Celeste.
- Pero sale Ceferino con alguien?!!!En realidad hay dos situaciones que son consistentes con lo que dijo Ceferino.

$A =$ verdadero $\quad B =$ verdadero $\quad C =$ verdadero
$A =$ falso $\quad B =$ falso $\quad C =$ verdadero

Resolviendo el Problema

- Como podemos computar esta solucion?
- Podemos usar tablas de verdad

A	B	C	$(A \vee B \vee C)$	$(\neg B \vee A \vee C)$	$(A \vee \neg B)$	$(C \vee \neg B)$	$(\neg A \vee B)$	
T	T	T	T	T	T	T	T	T
T	T	F	T	T	T	F	T	F
T	F	T	T	T	T	T	F	F
T	F	F	T	T	T	T	F	F
F	T	T	T	T	F	T	T	F
F	T	F	T	F	F	F	T	F
F	F	T	T	T	T	T	T	T
F	F	F	F	T	T	T	T	F

- Pero este método no es muy eficiente. (Cuál es la complejidad de SAT para LP?)

Algunas técnicas para resolver SAT

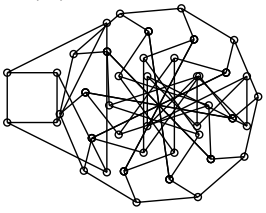
- Métodos Completos
  - Resolucion
  - Tableaux
  - Davis-Putman
  - Mapeo en ecuaciones lineares
- Métodos de Aproximación
  - Cambiar el valor de una variable en una fórmula insatisfecha
  - Algoritmos géneticos
  - Hill-climbing

Codificando Problemas

- Acabamos de ver como usar LP en un ejemplo muy simple.
- Pero el poder expresivo de PL es suficiente para hacer cosas mucho más interesantes:
  - coloreo de grafos
  - constraint satisfaction problems (CSP)
  - verificacion de hardware
  - planning
  - scheduling

Aplicación: Coloreo de Grafos

- El problema: Dados un grafo $G = \langle N, E \rangle$ donde N es un conjunto de nodos y E es un conjunto de ejes, y un número fijo de colores k. Decidir si podemos asignar colores a los nodos de N tal que:
  - Todos los nodos estan coloreados con uno de los k colores.
  - Para cada eje $(i, j) \in N$, el color de i es diferente del color de j.
- Ejemplo



Aplicación: Coloreo de Grafos

- Una codificación simple del problema de k-coloreo de un grafo con n nodos usa $n \cdot k$ símbolos proposicionales (una codificación más compacta usa sólo $n \log_2(k)$ símbolos proposicionales)
- Para $1 \leq i \leq n$, $1 \leq j \leq k$, escribimos p_{ij} para decir que 'el nodo i tiene color j'

Cada nodo tiene un color: $p_{i1} \vee \ldots \vee p_{ik}$,
para $1 \leq i \leq n$
Nodos vecinos tienen colores diferentes: $\neg p_{il} \vee \neg p_{jl}$,
para i y j nodos vecinos, y $1 \leq l \leq k$
Cada nodo no tiene mas de un color: $\neg p_{il} \vee \neg p_{im}$,
para $1 \leq i \leq n$, y $1 \leq l < m \leq k$

Aplicaciones: Coloreo de Grafos 2

- Resultados:
  - Los algoritmos de GSAT y WalkSAT son competitivos en comparación con algoritmos especifícos de coloreo de grafos
- Una aplicación en álgebra:
  - problemas relacionados con quasi-grupos pueden verse como casos particulares de coloreo de grafos.
  - algunos problemas abiertos en la teoría de quasi-grupos fueron codificados de esta forma y resueltos en forma automática mediante demostradores de teoremas para LP-SAT.
    E.g., existe un quasi-grupo que satisfaga las siguientes equaciones?

$$\forall a.(a \cdot a) = a$$
$$\forall a, b.((b \cdot a) \cdot b) \cdot b = a$$

Y problemas **realmente** importantes?

- Siga este link `http://www.sudokusolver.co.uk/`.

Métodos de Desición

- Los método de desición para resolver SAT debe:
  - Siempren responden SAT o UNSAT
  - En un tiempo finito
  - correctamente
- Los métodos completos más conocidos son
  - tablas de verdad
  - axiomatizaciones, calculo de Gentzen, deduccion natural
  - resolucion, tableaux
  - Davis-Putnam

Transformando una formula en forma clausal

- Forma clausal. Escribimos φ en forma normal conjuntiva (conjunctive normal form, CNF)

$$\varphi = \bigwedge_{l \in L} \bigvee_{m \in M} \psi_{(l,m)}, \text{donde } \psi_{(l,m)} \text{ es un literal (i.e., } p \text{ o } \neg p).$$

- Usando las siguientes equivalencias:

$$\begin{array}{rcl} (p \rightarrow q) & \leadsto & (\neg p \vee q) \\ (p \leftrightarrow q) & \leadsto & (p \rightarrow q) \wedge (q \rightarrow p) \\ (\neg(p \vee q)) & \leadsto & (\neg p \wedge \neg q) \\ (\neg(p \wedge q)) & \leadsto & (\neg p \vee \neg q) \\ (\neg\neg p) & \leadsto & p \\ (p \vee (q \wedge r)) & \leadsto & ((p \vee q) \wedge (p \vee r)) \end{array}$$

El conjunto de cláusulas asociado a

$$(l_{11} \vee \ldots \vee l_{1n_1}) \wedge (l_{21} \vee \ldots \vee l_{2n_2}) \wedge \ldots \wedge (l_{k1} \vee \ldots \vee l_{kn_k}) \quad \text{es}$$
$$\{\{l_{11}, \ldots, l_{1n_1}\}, \{l_{21}, \ldots, l_{2n_2}\}, \ldots, \{l_{k1}, \ldots, l_{kn_k}\}\}$$

Ejemplo 1

1. $\neg((p \vee q) \rightarrow (\neg q \rightarrow (p \vee q)))$
2. $\neg(\neg(p \vee q) \vee (\neg\neg q \vee (p \vee q)))$
3. $\neg(\neg(p \vee q) \vee (q \vee (p \vee q)))$
4. $(\neg\neg(p \vee q) \wedge \neg(q \vee (p \vee q)))$
5. $((p \vee q) \wedge \neg(q \vee (p \vee q)))$
6. $((p \vee q) \wedge (\neg q \wedge \neg(p \vee q)))$
7. $((p \vee q) \wedge (\neg q \wedge (\neg p \wedge \neg q)))$
8. $\{\{p, q\}, \{\neg q\}, \{\neg p\}\}$

Ejemplo 2

1. $(p \leftrightarrow q) \vee r$
2. $((p \rightarrow q) \wedge (q \rightarrow p)) \vee r$
3. $((\neg p \vee q) \wedge (\neg q \vee p)) \vee r$
4. $(((\neg p \vee q) \vee r) \wedge ((\neg q \vee p) \vee r))$
5. $\{\{\neg p, q, r\}, \{\neg q, p, r\}\}$

Métodos Completos: Davis-Putnam

- El método de Davis-Putnam es quizas el mas usado para demostracion automatica de LP-SAT
- A pesar de que ya tiene muchos años, es aun uno de los mas populares y exitosos entre los Métodos completos.

Sea Σ el conjunto de cláusulas asociado a la fórmula φ

```
procedure DP(Σ)
if Σ={} then return SAT              // (SAT)
if {} ∈ Σ then return UNSAT          // (UNSAT)
if Σ has unit clause {l}
   then DP(Σ[{l=true}])              // (Unit Pr.)
Choose literal l and
   if DP(Σ[{l=true}]) return SAT
      then return SAT
      else return DP(Σ[{l=false}])   // (Split)
```


Métodos Completos: Davis-Putnam 2

- DP no es el desarrollado por Davis & Putnam, sino el método perfeccionado de Davis, Logemann & Loveland
  - el algoritmo original propuesto por Davis & Putnam usaba una regla de resolucion en vez de la regla de splitting, lo que podria llevar a un uso exponencial de espacio
- Reglas Adicionales:
  - un literal puro (pure literal) es un literal que aparece siempre en forma positiva o siempre en forma negativa en el conjunto de clausulas; podemos asignar a ese literal el valor verdadero (si aparece positivo) o falso (si aparece negativo) y eliminarlo.

`(Pure) if Sigma has pure literal l then DP(Sigma {l=true})`

  - Tautology Deletion

`(Taut) if Sigma contains C ∪ {p,¬p} then DP(Sigma\ C ∪ {p,¬p})`

Davis-Putnam: Las Reglas

- La regla (Pure) usualmente no es implementada, ya que el costo de su evaluacion puede ser mas alto que los beneficios que produce
- Lo mismo vale para la regla (Taut): las tautologies solo aparecen al comienzo de la busqueda
- La regla (Unit) no es esencial y su efecto puede obtenerse mediante una combinacion de las reglas (Split) y (Empty)
- Pero (Unit) es crucial para el buen comportamiento computacional del método. Por ejemplo, la regla (Unit) por si misma es completa sobre clausulas Horn.

Ejemplos de DP

- $\neg((p \vee q) \rightarrow (\neg q \rightarrow (p \vee q)))$

  $\{\{p, q\}, \{\neg q\}, \{\neg p\}\}$
- $(p \leftrightarrow q) \vee r$

  $\{\{\neg p, q, r\}, \{\neg q, p, r\}\}$

Davis-Putnam: La regla (Split)

La regla (Split) es no-deterministica: Que literal elegimos?

- Heuristica MOM: elegir el literal que ocurre 'most often in the minimal size clauses' (con empates resueltos en forma aleatoria o siguiendo un orden predeterminado). Este método es uno de los mejores en terminos de resultados, velocidad y simplicidad.
- Heuristica de Jeroslow-Wang: estimamos la contribucion que cada literal podria hacer a la satisfiabilidad del conjunto de clausulas y elegimos el coeficiente mas alto

$$\text{score}(l) = \sum_{c \in \Sigma \ \& \ l \in c} 2^{-|c|}$$

- SATZ, uno de los mejores implementaciones actuales de DP, usa una heuristica que intenta maximizar el uso de unit propagation: genera todos los posibles branchings con distintos literales y aplica inmediatamente unit propagation sobre el resultado, para continuar la ejecucion con el conjunto de clausulas mas pequeño.

Métodos Incompletos (o de Aproximacion): Motivacion

- DP puede resolver en tiempo razonable problemas con 500 variables proposicionales...
- ... pero los problemas que surgen habitualmente en la practica tienen 1000s de variables!
- Dependiendo de la aplicacion, Métodos de semi-decision pueden ser utiles: encontrar una solucion en algunos casos
- E.g., encontrar un plan $\equiv$ encontrar un modelo, y podemos no estar interesados en los casos en los que no existe un plan
- Ademas, podemos estar interesados en "anytime answers" que dan "best guess" en cualquier momento que querramos detener el algoritmo

Revoleando Monedas: El Algoritmo "Greedy"

- El algoritmo fue propuesto por Koutsopias y Papadimitriou
  - Idea principal: cambiamos el valor de una variable hasta que no podemos incrementar el numero de clausulas satisfechas.

```
procedure greedy(Sigma)
  T := random(Sigma) ; random assignment
  repeat until no improvement possible
    T := T with variable flipped that increases
         the number of satisfied clauses
  end
```

- El algoritmo encuentra un modelo para casi todos los problemas satisfacibles con n variables proposicionales y $O(n^2)$ clausulas (lamentablemente, muy pocos problemas son de este tipo)

El procedimiento GSAT

- El algoritmo fue propuesto por Selman, Levesque y Mitchell
  - Agrega restarts al algoritmo greedy, y permite "pasos al costado" (i.e., que no incrementan la funcion de costo)

```
procedure GSAT(Sigma)
  for i := 1 to MAX-TRIES ; estos son los restarts
    T := random(Sigma) ; asignacion al azar
    for j := 1 to MAX-FLIPS ; asegura terminacion
      if T satisfies Sigma then return T
      else T := T with variable flipped to maximize
                number of satisfied clauses
      ; No importa si el # de clausulas satisfechas
      ; no se incrementan. Estos son los "side steps"
    end
  end
```


GSAT: Evaluacion

- El procedimiento GSAT ha sido muy influencial
- GSAT es excelente en algunos tipos de problemas (random 3-SAT, n-queens, etc.)

formulas		GSAT			DP		
var	clauses	M-FLIPS	restarts	time	choices	depth	time
50	215	250	6.4	0.4s	77	11	1.4s
100	430	500	42.5	6s	84 x 10^3	19	2.8m
140	602	700	52.6	14s	2.2 x 10^6	27	4.7h
150	645	1500	100.5	45s	–	–	–
300	1275	6000	231.8	12m	–	–	–
500	2150	10000	995.8	1.6h	–	–	–

GSAT: Pasos al costado

- Recordemos: la diferencia mas importante entre el algoritmo greedy y GSAT es la posibilidad de pasos al costado
- Hay alguna diferencia?

type	formulas		M-FLIPS	no sideways moves			all moves		
	vars	clauses		%-solved	restarts	time	%-solved	tries	time
random	50	215	1000	69 %	537	10s	100 %	6	1.4s
random	100	430	100000	39 %	63382	15 m	100 %	81	2.8m
30-queens	900	43240	100000	100 %	50000	30h	100 %	1	2.5s

Logica Proposicional: Conclusiones

- Los Métodos completos garantizan solucionar el problema LP-SAT (y en muchos casos, e.g. DP, permiten encontrar todas las soluciones posibles)
- Los Métodos de aproximacion garantizan correctitud pero no completitud (i.e., si encuentran una solucion, es correcta, pero pueden terminar diciendo 'No se').
- El método de DP es muy usado, pero notemos que DP es en realidad un esquema general para una familia de algoritmos. Como vimos, se pueden tomar decisiones diferentes hacerca de como implementarlo (como elegimos literales, como hacemos backtracking, etc.)
- Aun por ejemplos "simples" en en logica proposicional las cosas puden ponerse dificiles si no usamos optimizaciones inteligentes.

Zchaff

- Un demostrador muy optimizaco implementando una version de DP (conocida como el algoritmo 'chaff').
- Site: `http://www.princeton.edu/~chaff/zchaff.html`
- Tambien conocido como el 'Princeton Prover'.
- zChaff se hizo famoso al resolver problemas con mas de un millon de variables y mas de 10 millones de clausulas.
- Es usado en otros systems como el planner BlackBox, el Model Checker NuSMV, el demostrador GrAnDe, etc.

WalkSat

- Walksat esla implementacion de un algoritmo de busqueda local para resolver SAT para PL (es una mejora de GSAT).
- Site: `http://www.cs.rochester.edu/u/kautz/walksat`
- Ha resultado particularmente exitoso en resolver problemas resultantes de la conversion a SAT de problemas de planning.