Lógica modal computacional

Algoritmos efectivos

Daniel Gorín y Sergio Mera

1er cuatrimestre de 2010

Repaso

Estuvimos viendo...

- Complejidad de distintas lógicas modales
- En particular, algoritmos óptimos, pero imprácticos!

Hoy vamos a ver

- Algoritmos con peor complejidad
- Pero buen comportamiento empírico en casos promedio

Satisfacibilidad proposicional

Cuestión de ~~adivinar~~ buscar modelos

Algoritmo NP para satisfacibilidad proposicional

- Dada una fórmula φ:
  1. Adivinar una valuación v para las variables de φ
  2. Devolver 1 sii $v \models \varphi$
- Obviamente, no sirve como algoritmo efectivo

Algoritmo DPLL (Davis-Putnam-Logemann-Loveland)

- Dada una fórmula φ *en clause normal form*
  1. Buscar ("backtracking") una valuación que satisfaga φ
  2. Devolver 1 sii se encuentra tal valuación
- Es la base de todos los SAT Solvers modernos
- Pero éstos agregan mucha "inteligencia"!

DPLL en más detalle

Clause Normal Form(CNF)

- Conjunción de disyunciones de literales (cláusulas)
- Ej. $(p_1 \vee \neg p_2 \vee p_3) \wedge (p_2 \vee p_3 \vee \neg p_4) \wedge (\neg p_1 \vee \neg p_3)$

Reglas de DPLL

$$\textit{Split} \quad \frac{v \| S}{v[p \mapsto \top] \| S \;\mid\; v[p \mapsto \bot] \| S} \qquad p \text{ indefinido en } v$$

$$\textit{Unit}_{\top} \quad \frac{v \| S \wedge (C \vee p)}{v[p \mapsto \top] \| S} \qquad p \text{ indefinido en } v \text{ y } v \not\models C$$

$$\textit{Unit}_{\bot} \quad \frac{v \| S \wedge (C \vee \neg p)}{v[p \mapsto \bot] \| S} \qquad p \text{ indefinido en } v \text{ y } v \not\models C$$

$$\textit{Fail} \quad \frac{v \| S \wedge C}{\times} \qquad v \not\models C$$

Ejemplo

Ejercicio

Decidir si $\varphi = C_1 \wedge C_2 \wedge C_3 \wedge C_4 \wedge C_5 \wedge C_6$ es sat., donde:

$$C_1 = \neg p_1 \vee p_2 \vee p_3$$
$$C_2 = p_1$$
$$C_3 = \neg p_2 \vee p_3$$
$$C_4 = p_2 \vee \neg p_3$$
$$C_5 = p_2 \vee p_3$$
$$C_6 = \neg p_2 \vee \neg p_3$$

Del dicho al hecho…

Detalles que hacen la diferencia al implementar DPLL

- *Split* corresponde al backtracking, aplicarla poco!
- Tener buenas *heurísticas* sobre qué p elegir en cada *Split*
- Implementar:
  - *Conflict analysis:* qué está forzando el backtracking?
  - *Learning:* agregar cláusulas con estas nuevas restricciones
  - *Backjumping:* basado en esto, poder volver más de un paso
- Usar estructuras de datos especiales
- Implementar otras heurísticas (restarting, etc.)

Transformaciones a CNF

Toda fórmula tiene una *equivalente* en CNF

Por ejemplo, usar la siguiente reescritura:

$$\begin{aligned} \neg\neg\varphi &\rightsquigarrow \varphi \\ \neg(\varphi \wedge \psi) &\rightsquigarrow (\neg\varphi \vee \neg\psi) \\ \neg(\varphi \vee \psi) &\rightsquigarrow (\neg\varphi \wedge \neg\psi) \\ \varphi \vee (\psi \wedge \chi) &\rightsquigarrow (\varphi \vee \psi) \wedge (\varphi \vee \chi) \end{aligned}$$

Pero tenemos este... Teorema

Ningún polinomio acota el tamaño de una equivalente en CNF.

Existen transformaciones a CNF que...

- Devuelven una fórmula no equivalente, pero *equisatisfacible*
- Son polinomiales!
- Se basan en introducir nuevos símbolos; por ejemplo:

$$\varphi \vee (\psi \wedge \chi) \rightsquigarrow (\varphi \vee p) \wedge (\neg p \vee \psi) \wedge (\neg p \vee \chi)$$

Satisfacibilidad de K via DPLL

Algoritmo NPSPACE para K-satisfacibilidad

- Dada una fórmula φ
  1. Adivinar un Hintikka set H que contenga a φ
  2. Devolver 1 sólamente si:
     - para cada $\Diamond\psi \in H$, $\{\psi\} \cup \{\chi \mid \Box\chi \in H\}$ es satisfacible

DPLL modal

- Dada una fórmula φ en *modal clause normal form (mCNF)*
  1. Obtener (via DPLL) un Hintikka set H que contenga a φ
  2. Si para cada $\Diamond\varphi \in H$, $\{\varphi\} \cup \{\chi \mid \Box\chi \in H\}$ es satisfacible
     - Devolver 1
  3. Si no (i.e. si cierto $\{\varphi\} \cup \{\chi \mid \Box\chi \in H\}$ no es satisfacible)
     - Obtener (via DPLL) una extensión de H que contenga a
       $$\neg\Diamond\varphi \vee \bigvee\{\neg\Box\chi \mid \Box\chi \in H\}$$
     - Volver al punto 2.
  4. Devolver 0 siempre que una búsqueda por DPLL falle

DPLL modal

Detalles de implementación

Modal Clause Normal Form (mCNF)

- mCNF: Conjunción de *cláusulas modales*
- Cláusula modal: disyunción de *literales modales*
- Literales modales: p, $\neg p$, $\Box\chi$, $\neg\Box\chi$ (con χ cláusula modal)
- Valen las mismas observaciones hechas para CNF!

De modal a proposicional, ida y vuelta

- Usamos dos mappings $\rightsquigarrow_p$ y $\rightsquigarrow_m$

$$p_i \rightsquigarrow_p q_{p_i} \qquad q_{p_i} \rightsquigarrow_m p_i$$

$$\Box\chi \rightsquigarrow_p q_{\Box\chi} \qquad q_{\Box\chi} \rightsquigarrow_m \Box\chi$$

- Dada φ, usamos $\rightsquigarrow_p$ para obtener una fórmula prop. α
- Dada una $v \models \alpha$, v induce un único Hintikka set sobre φ
- Con $\rightsquigarrow_m$ recuperamos los literales modales *elegidos* por v.

Ejemplo

Ejercicio

Decidir si $\varphi = C_1 \wedge C_2 \wedge C_3 \wedge C_4 \wedge C_5$ es sat., donde:

$$
\begin{aligned}
C_1 &= p_1 \\
C_2 &= \neg p_1 \vee \neg \Box (p_1 \vee p_2) \\
C_3 &= \neg p_1 \vee p_2 \\
C_4 &= \neg p_2 \vee \Box p_1 \\
C_5 &= \neg p_1 \vee \Box p_2
\end{aligned}
$$

DPLL modal

Optimizaciones

No recomenzar de cero cada vez

- Hay SAT solvers que dejan agregar cláusulas y continuar
- Luego, si obtuvimos una valuación v tal que:
  - Exige que $\{\neg\Box\chi, \Box\psi_1, \ldots \Box\psi_n\}$ sea satisfacibles
  - Pero probamos que $\{\chi, \psi_1, \ldots \psi_n\}$ no lo es
- Debemos agregar la cláusula $\Box\chi \vee \neg\Box\chi_1 \vee \cdots \vee \neg\Box\chi_n$
- Y recontinuamos (salvando el "esfuerzo" en obtener v)

Caching

1. Si vimos que $\{\neg\Box\chi, \Box\psi_1, \ldots \Box\psi_n\}$ es satisfacible, recordarlo
2. Si luego queremos ver si $H = \{\neg\Box\varphi, \Box\varphi_1, \ldots \Box\varphi_k\}$ es sat,
   - y sabemos que un H' es sat., con $H' \supseteq H$, entonces H es sat.

Satisfacibilidad modal ~~adivinando~~ buscando modelos

Tableaux modales

Algoritmo NTIME(f) para lógicas con modelos f-acotados

- Dada una fórmula φ:
  1. Adivinar un modelo $\mathcal{M}$ de tamaño a lo sumo $f(|\varphi|)$
  2. Adivinar un w en el dominio de $\mathcal{M}$
  3. Devolver 1 sii $\mathcal{M}, w \models \varphi$
- Obviamente, no sirve como algoritmo efectivo

Algoritmo de Tableaux

- Dada una fórmula φ
  1. Buscar ("backtracking") sistemáticamente un modelo de φ
  2. Devolver 1 sii se encuentra tal modelo
- Es la base de muchos razonadores para lógicas modales
- Varios tipos de tableaux, vamos a ver sólo *tableaux etiquetados*

¿Qué es un algoritmo de tableau (etiquetado)?

- *Reglas de tableaux:* definen un árbol de posibilidades.
- Los nodos de un árbol son, en general:
  - Fórmulas "etiquetadas" $w{:}\psi$, donde w es una etiqueta.
  - "Relaciones" Rwv donde w y v son etiquetas.
- Cada rama del árbol codifica de alguna manera un modelo.
- Las reglas nos dicen:
  1. Cómo detectar que una rama no nos sirve (reglas de clash).
  2. Cómo *expandir* una rama sin clash.
- *Algoritmo:* Dada φ, explorar (backtracking) el árbol de $w{:}\varphi$.

Ejemplo de tableaux etiquetado

Lógica modal con pasado

Reglas de expansión

$$\wedge \quad \frac{w{:}\varphi \wedge \psi}{w{:}\varphi, w{:}\psi} \qquad \vee \quad \frac{w{:}\varphi \vee \psi}{w{:}\varphi \mid w{:}\psi}$$

$$\Diamond \quad \frac{w{:}\Diamond\varphi}{Rwv, v{:}\varphi} \text{ (nuevo } v\text{)} \qquad \Diamond^{-1} \quad \frac{w{:}\Diamond^{-1}\varphi}{Rvw, v{:}\varphi} \text{ (nuevo } v\text{)}$$

$$\Box \quad \frac{w{:}\Box\varphi, Rwv}{v{:}\varphi} \qquad \Box^{-1} \quad \frac{w{:}\Box^{-1}\varphi, Rvw}{v{:}\varphi}$$

Reglas de clash

$$\textit{clash} \quad \frac{w{:}p, w{:}\neg p}{\bot}$$

- Asumimos fórmulas en *negation normal form*.
- Las reglas $\Diamond$ y $\Diamond^{-1}$ se usan sólo si no existe tal v en la rama.

Ejemplo

Ejercicio

Decidir si $\varphi = p_1 \wedge \Diamond p_2 \wedge \Diamond \Box^{-1} \Box (\neg p_2 \vee \Box^{-1} \neg p_1)$ es satisfacible.

Tableau etiquetado para la lógica modal con pasado

Completitud

Teorema (Completitud)

Si Γ es una rama abierta y saturada para φ, φ es satisfacible.

Demostración

- Extraemos un modelo de Γ. Sea $\mathcal{M}_\Gamma = \langle W_\Gamma, R_\Gamma, V_\Gamma \rangle$ donde

$$W_\Gamma = \{w \mid w{:}\varphi \in \Gamma\}$$
$$R_\Gamma = \{(w, v) \mid Rwv \in \Gamma\}$$
$$V_\Gamma(p) = \{w \mid w{:}p \in \Gamma\}$$

- Sea ψ la *mínima* fórmula tal que $w{:}\psi \in \Gamma$ y $\mathcal{M}_\Gamma, w \not\models \psi$.
  - $\psi \neq p$ (porque en ese caso $w \in V(p)$) y $\psi \neq \neg p$ (habría clash)
  - $\psi \neq \psi_1 \vee \psi_2$ porque tendríamos $w{:}\psi_i \in \Gamma$ y no sería mínima
  - $\psi \neq \psi_1 \wedge \psi_2$ por razones análogas
  - $\psi \neq \Diamond\chi$ porque tendríamos $Rwv, w{:}\chi \in \Gamma$ y no sería mínima
  - $\psi \neq \Box\chi$, $\psi \neq \Box^{-1}\chi$ y $\psi \neq \Diamond^{-1}\chi$ por razones análogas.
- Luego, no existe tal fórmula; $w{:}\psi \in \Gamma$ implica $\mathcal{M}, w \models \psi$

Tableau etiquetado para la lógica modal con pasado

Terminación

Teorema

Toda rama saturada de un tableaux para φ es finita.

Demostración

Sea Γ una rama saturada y $\mathsf{LABEL}(w) = \{\psi \mid w{:}\psi \in \Gamma\}$.

1. $\mathsf{LABEL}(w)$ es finito porque son todas subfórmulas de φ.
2. Luego, $\{v \mid Rwv \in \Gamma\}$ es finito (ver nota sobre $\Diamond$ y $\Diamond^{-1}$).
3. Entonces, Γ es infinito sii existe una cadena $w_1, w_2, \ldots$ tal que w_i *generó* a w_{i+1} usando la regla $\Diamond$ ó la regla $\Diamond^{-1}$.
4. Pero si w genera a v, $d(\mathsf{LABEL}(w)) > d(\mathsf{LABEL}(v))$ (sale por inducción en la derivación de Γ, d es profundidad modal).
5. Por lo tanto, para algún j, $d(\mathsf{LABEL}(w_j)) = 0$, absurdo.

Detalles de implementación

¿Importa el orden en que aplicamos las reglas?

- No afecta la terminación del algoritmo.
- Sí afecta el tamaño del árbol generado!
  - Considerar: $(p_1 \vee p_2) \wedge ((p_3 \vee p_4) \wedge ((p_5 \vee p_6) \wedge (p \wedge \neg p)))$
  - ¿Qué sucede si siempre preferimos aplicar $\wedge$ antes que $\vee$?
  - ¿Qué sucede si siempre preferimos aplicar $\vee$ antes que $\wedge$?

Heurísticas básicas

- Usar reglas sin branching (e.g., $\wedge$) antes que aquellas con branching (como $\vee$)
- Usar reglas proposicionales (e.g., $\wedge$ y $\vee$) antes que reglas modales (como $\Diamond$ y $\Box$)

Optimizaciones

- Las reglas $\wedge$, $\vee$ y *clash* son un tableaux proposicional
- Pero es preferible DPLL para razonamiento proposicional
- Los demostradores basados en tableaux incorporan elementos de DPLL:
  - Branching semántico (una forma de splitting)
  - Backjumping
  - Caching

Terminación en casos más complejos

Un tableaux para K sobre la clase de modelos transitivos (K4)

$\wedge$ $\dfrac{w{:}\varphi\wedge\psi}{w{:}\varphi, w{:}\psi}$ $\qquad$ $\vee$ $\dfrac{w{:}\varphi\vee\psi}{w{:}\varphi \mid w{:}\psi}$ $\qquad$ *clash* $\dfrac{w{:}p, w{:}\neg p}{\bot}$

$\Diamond$ $\dfrac{w{:}\Diamond\varphi}{Rwv, v{:}\varphi}$ (nuevo v) $\qquad$ $\Box_4$ $\dfrac{w{:}\Box\varphi, Rwv}{v{:}\varphi, v{:}\Box\varphi}$

Teorema

Este tableaux es completo para K4.

Demostración

- Dada Γ (abierta y saturada) armamos $\mathcal{M}_\Gamma$ con
$$R_\Gamma = \{(w, v) \mid Rwv \in \Gamma\}^+$$
- Sup. $\Box\varphi$ es la mínima tal que $w{:}\Box\psi \in \Gamma$ y $\mathcal{M}_\Gamma, w \not\models \Box\psi$.
- Luego, existe $\{Rwv_1, Rv_1v_2, \ldots Rv_2v_k\} \subset \Gamma$ y $\mathcal{M}, w \not\models \varphi$.
- Γ está saturada por $\Box_4$, entonces $\{v_i{:}\varphi, v_i{:}\Box\varphi\} \subset \Gamma$ $(\forall i \leq k)$
- Luego $v_k{:}\varphi \in \Gamma$ y $\Box\varphi$ no sería mínima. El resto queda igual.

Terminación en casos más complejos

Blocking

- ¿Podemos repetir el argumento de terminación?
- ¿Qué sucede al ejecutar este tableaux sobre $w{:}(\Diamond p \wedge \Box\Diamond p)$?
- *Conclusión:* una rama abierta saturada puede no ser finita!

Técnicas de blocking

- Se usan para garantizar terminación en implementaciones
- *Idea:*
  - Algunas $w{:}\varphi$ pueden "bloquearse" o "desbloquearse"
  - Algunas reglas no se aplican sobre fórmulas bloqueadas
- Muchos tipos de blocking
  - subset blocking
  - dynamic blocking
  - ...
- En cada caso se debe probar terminación... y completitud!