To P or not to P
That's the Question

Carlos Areces

areces@loria.fr

Agosto 2005,
Buenos Aires, Argentina

Charla Auspiciada por Bodegas LORIA :)

Para Empezar, una Historia.

Para Empezar, una Historia.

- Habia una vez . . . .

Para Empezar, una Historia.

- Habia una vez . . . .
- Otros artículos de Edward Closed:
  - El Condicional Contrafálico.
  - La Implicación Teológica

Mañana Llueve o no Llueve

Mañana Llueve o no Llueve

- Frases como “Juan ama a María, o no” o “Mañana Llueve o no Llueve” suenan un poco ridículas. No parecen decir nada.

Mañana Llueve o no Llueve

- Frases como “Juan ama a María, o no” o “Mañana Llueve o no Llueve” suenan un poco ridículas. No parecen decir nada.
- Notemos que no son teoremas como entendemos la palabra en Matemática, porque los teoremas matemáticos sí dicen algo (por ejemplo, el teorema de Fermat dice algo acerca de los números naturales).

Mañana Llueve o no Llueve

- Frases como “Juan ama a María, o no” o “Mañana Llueve o no Llueve” suenan un poco ridículas. No parecen decir nada.
- Notemos que no son teoremas como entendemos la palabra en Matemática, porque los teoremas matemáticos sí dicen algo (por ejemplo, el teorema de Fermat dice algo acerca de los números naturales).
- Sin embargo, las dos nociones están muy relacionadas (en realidad, coinciden). Cuál es el truco?

Mañana Llueve o no Llueve

- Frases como “Juan ama a María, o no” o “Mañana Llueve o no Llueve” suenan un poco ridículas. No parecen decir nada.
- Notemos que no son teoremas como entendemos la palabra en Matemática, porque los teoremas matemáticos sí dicen algo (por ejemplo, el teorema de Fermat dice algo acerca de los números naturales).
- Sin embargo, las dos nociones están muy relacionadas (en realidad, coinciden). Cuál es el truco?
- En vez de considerar fórmulas por separado, consideremos conjuntos de fórmulas. Y pensemos que cuando tomamos un conjunto T de fórmulas, la idea es que queremos que todas las fórmulas de T sean ciertas.

Consecuencia Lógica

Consecuencia Lógica

- Podemos ahora definir la noción Se sigue de o Se infiere de entre T y otra fórmula arbitraria φ, diciendo que

  φ *se sigue de* T *sii siempre que* T *es cierta,* φ *también.*

- O, puesto de otra forma, no hay ninguna situación (ningún modelo) en donde T sea cierta y φ sea falsa.

Consecuencia Lógica

- Podemos ahora definir la noción Se sigue de o Se infiere de entre T y otra fórmula arbitraria φ, diciendo que

  φ *se sigue de* T *sii siempre que* T *es cierta,* φ *también.*

- O, puesto de otra forma, no hay ninguna situación (ningún modelo) en donde T sea cierta y φ sea falsa.
- Pero esto es lo mismo que decir que $(\bigwedge T) \rightarrow \varphi$ es siempre cierta!

T, . . . de Teoría

T, . . . de Teoría

- Volviendo a la Matemática. La idea intuitiva es que en T anotamos nuestros axiomas. Las propiedades que asumimos son ciertas e indiscutibles. T es nuestra teoría.

T, . . . de Teoría

- Volviendo a la Matemática. La idea intuitiva es que en T anotamos nuestros axiomas. Las propiedades que asumimos son ciertas e indiscutibles. T es nuestra teoría.
- Por ejemplo los axiomas de Peano son una teoría sobre los número naturales.

T, . . . de Teoría

- Volviendo a la Matemática. La idea intuitiva es que en T anotamos nuestros axiomas. Las propiedades que asumimos son ciertas e indiscutibles. T es nuestra teoría.
- Por ejemplo los axiomas de Peano son una teoría sobre los número naturales.
- Podriamos chequear si el teorema de Fermat es cierto en los números naturales, chequeando que

$$(\bigwedge \mathsf{PEANO}) \rightarrow \mathsf{FERMAT}$$

es una fórmula ‘trivial’ (cierta en todos los modelos, i.e., una tautología).

Cuanto Cuesta?

Cuanto Cuesta?

- Pero podemos? Cuan difícil es determinar si una fórmula es una tautología?

Cuanto Cuesta?

- Pero podemos? Cuan difícil es determinar si una fórmula es una tautología?
- Como todo en la vida, la respuesta es: Depende.

Cuanto Cuesta?

- Pero podemos? Cuan difícil es determinar si una fórmula es una tautología?
- Como todo en la vida, la respuesta es: Depende.
- Como vimos, podemos codificar en la pregunta ‘Es la fórmula φ una tautología’ problemas no triviales como el teorema de Fermat.

Cuanto Cuesta?

- Pero podemos? Cuan difícil es determinar si una fórmula es una tautología?
- Como todo en la vida, la respuesta es: Depende.
- Como vimos, podemos codificar en la pregunta 'Es la fórmula φ una tautología' problemas no triviales como el teorema de Fermat.
- Va a depender de cuan expresivo es nuestro lenguaje (qué problemas podamos codificar), cuán difícil va a ser saber si una fórmula es una tautología.

Curso Acelerado de Decidibilidad y Complejidad

Curso Acelerado de Decidibilidad y Complejidad

- Hay cosas que no pueden comprarse ni con todo el dinero del mundo.

Curso Acelerado de Decidibilidad y Complejidad

- Hay cosas que no pueden comprarse ni con todo el dinero del mundo.
- Hay problemas que no pueden resolverse ni con todo el tiempo y toda la memoria del mundo.

Curso Acelerado de Decidibilidad y Complejidad

- Hay cosas que no pueden comprarse ni con todo el dinero del mundo.
- Hay problemas que no pueden resolverse ni con todo el tiempo y toda la memoria del mundo.
- El Problema de la Parada: Dado un programa P, decidir si P termina para todo input.
- La lógica de primer orden puede codificar maquinas de Turing: En particular, dado un programa P existe una fórmula φ_P tal que φ_P es una tautología sii P termina.

Curso Acelerado de Decidibilidad y Complejidad

Curso Acelerado de Decidibilidad y Complejidad

- En aquellos casos en que el programa sí termina, podemos medir su complejidad en términos del tamaño del input:
  - Clase P: tiempo polinomial en MT determinística
  - Clase NP: tiempo polinomial en MT determinística
  - Clase PSPACE = NPSPACE: espacio polinomial en MT determinística o no determinística
  - Clase EXPTIME: tiempo exponencial en MT determinística
  - Clase NEXPTIME: tiempo exponencial en MT no determinística
  - Clase EXPSPACE = NEXPSPACE: espacio exponencial en MT determinística o no determinística

Lógica Proposicional (LP)

Lógica Proposicional (LP)

- Uno de los lenguajes más simples: Sólo puedo decir $P \wedge Q$, $\neg P$.

Lógica Proposicional (LP)

- Uno de los lenguajes más simples: Sólo puedo decir $P \wedge Q$, $\neg P$.
- Decidir si una fórmula de LP es tautología es un problema en NP (NP-complete).
- Que algoritmos existen para resolver este problema?
- Pero primero, para qué sirve?

Aplicaciones: Coloreo de Grafos

Aplicaciones: Coloreo de Grafos

- Con una codificación muy compacta, podemos expresar k-coloreo de un grafo de n nodos usando $n \log_2(k)$ símbolos proposicionales

Aplicaciones: Coloreo de Grafos

- Con una codificación muy compacta, podemos expresar k-coloreo de un grafo de n nodos usando $n \log_2(k)$ símbolos proposicionales
- Existe una codificación simple usando $n \cdot k$ símbolos proposicionales

Aplicaciones: Coloreo de Grafos

- Con una codificación muy compacta, podemos expresar k-coloreo de un grafo de n nodos usando $n \log_2(k)$ símbolos proposicionales
- Existe una codificación simple usando $n \cdot k$ símbolos proposicionales
- Para $1 \leq i \leq n$, $1 \leq j \leq k$, sea p_{ij} ‘el nodo i tiene color j’

Aplicaciones: Coloreo de Grafos

- Con una codificación muy compacta, podemos expresar k-coloreo de un grafo de n nodos usando $n \log_2(k)$ símbolos proposicionales
- Existe una codificación simple usando $n \cdot k$ símbolos proposicionales
- Para $1 \leq i \leq n$, $1 \leq j \leq k$, sea p_{ij} 'el nodo i tiene color j'
- Nodos vecinos tienen distinto color.

$$\neg p_{il} \vee \neg p_{jl},$$

para i y j nodos vecinos, y $1 \leq l \leq k$ (linear)

Aplicaciones: Coloreo de Grafos

Aplicaciones: Coloreo de Grafos

- Cada nodo tiene al menos un color.

$$p_{i1} \vee \ldots \vee p_{ik}, \text{ para } 1 \leq i \leq n \text{ (linear)}$$

Aplicaciones: Coloreo de Grafos

- Cada nodo tiene al menos un color.

$$p_{i1} \vee \ldots \vee p_{ik}, \text{ para } 1 \leq i \leq n \text{ (linear)}$$

- Cada nodo tiene a lo sumo un color.

$$\neg p_{il} \vee \neg p_{im},$$

para $1 \leq i \leq n$, y $1 \leq l < m \leq k$ (cuadrático)

Aplicaciones: Coloreo de Grafos

Aplicaciones: Coloreo de Grafos

- Resultado:
  - Los algoritmos de GSAT y WalkSAT tienen una performance competitiva con algoritmos especializados de coloreo de grafos

Aplicaciones: Coloreo de Grafos

- Resultado:
  - Los algoritmos de GSAT y WalkSAT tienen una performance competitiva con algoritmos especializados de coloreo de grafos
- Aplicaciones en álgebra:
  - Ciertos problemas en quasigrupos pueden verso como problemas especializados de coloreo de grafos.

Aplicaciones: Coloreo de Grafos

- Resultado:
  - Los algoritmos de GSAT y WalkSAT tienen una performance competitiva con algoritmos especializados de coloreo de grafos
- Aplicaciones en álgebra:
  - Ciertos problemas en quasigrupos pueden verso como problemas especializados de coloreo de grafos.
  - Algunos problemas abiertos en quasigrupos fueron codificados de esta forma y solucionados usando demostradores automáticos para LP. Por ejemplo: existe un quasigrupo satisfaciendo las siguientes equaciones:?

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

Aplicaciones: Hardware

Aplicaciones: Hardware

- Codificar especificaciones de hardware en LP es muchas veces muy simple
  - La presencia de señal en un cable, se representa mediante una variable proposicional
  - cada gate **AND** se representa mediante un conectivo **AND**
  - etc.

Aplicaciones: Hardware

- Codificar especificaciones de hardware en LP es muchas veces muy simple
  - La presencia de señal en un cable, se representa mediante una variable proposicional
  - cada gate **AND** se representa mediante un conectivo **AND**
  - etc.
- Problemas de Larrabee's , problemas de "stuck-at" y "bridge-fault"
- Síntesis de circuitos (Kamath et al.)

Applications: Conclusiones

Applications: Conclusiones

- Existen tambíen aplicaciones más “exóticas”
  - Constraint Satisfaction
  - Planning
  - Scheduling
  - RNA folding
  - hand-writing recognition
  - graph isomorphism

Applications: Conclusiones

- Existen tambíen aplicaciones más “exóticas”
  - Constraint Satisfaction
  - Planning
  - Scheduling
  - RNA folding
  - hand-writing recognition
  - graph isomorphism
- En general, el problema es encontrar la codificación adecuada

Applications: Conclusiones

- Existen tambíen aplicaciones más “exóticas”
  - Constraint Satisfaction
  - Planning
  - Scheduling
  - RNA folding
  - hand-writing recognition
  - graph isomorphism
- En general, el problema es encontrar la codificación adecuada

Representation is a Key Question

Métodos Completos

Métodos Completos

- Los métodos más tradicionales para resolver SAT de LP son completos
  - una respuesta correcta de SI o NO siempre se obtiene

Métodos Completos

- Los métodos más tradicionales para resolver SAT de LP son completos
  - una respuesta correcta de SI o NO siempre se obtiene
- Los métodos mas conocidos son
  - axiomatizaciones,
  - tablas de verdad,
  - resolución, tableaux
  - Davis-Putnam

Métodos Completos: Davis-Putnam

Métodos Completos: Davis-Putnam

- El método de Davis-Putnam es uno de los algoritmos más usados actualmente en demostradores automáticos

Métodos Completos: Davis-Putnam

- El método de Davis-Putnam es uno de los algoritmos más usados actualmente en demostradores automáticos
- Representemos la fórmula input φ como un conjunto de cláusulas, donde una cláusula es un conjunto de literales p o $\neg p$.

```
procedure DP(Sigma)
  (Sat)       if Sigma = {} then retornar “sat”
  (Empty)     if {} ∈ Sigma then retornar “unsat”
  (Taut)      if {p v ¬ p} ∪ C ∈ Sigma then DP(Sigma - {p v ¬ p} ∪ C)
  (Unit Pr)   if {l} ∈ Sigma then DP(Sigma {l = true})
  (Pure)      if Sigma tiene un literal puro l then DP(Sigma {l = true})
  (Split)     if DP(Sigma {l = true}) retorna “sat” then retornar “sat”
              else DP(Sigma {l = false})
```

Davis-Putnam: Las Reglas

Davis-Putnam: Las Reglas

- (Pure) usualmente no se incluye, porque es muy costosa de evaluar
- Lo mismo pasa con (Taut): tautologías obvias aparecen solamente al principio de la búsqueda

Davis-Putnam: Las Reglas

- (Pure) usualmente no se incluye, porque es muy costosa de evaluar
- Lo mismo pasa con (Taut): tautologías obvias aparecen solamente al principio de la búsqueda
- (Unit) no es esencial para completitud (se puede obtener a partir de (Split) y (Empty))
- Pero es crucial en términos de performance. Por ejemplo, ya por sí sola, (Unit) es completa para cláusulas Horn

Davis-Putnam: Organizando la Búsqueda

Davis-Putnam: Organizando la Búsqueda

- La regla (Split) es no-determinística: Qué literal Elegimos?

Davis-Putnam: Organizando la Búsqueda

- La regla (Split) es no-determinística: Qué literal Elegimos?
  - MOM's heuristics: **m**ost **o**ften in the **m**inimal size clauses.

Davis-Putnam: Organizando la Búsqueda

- La regla (Split) es no-determinística: Qué literal Elegimos?
  - MOM's heuristics: **m**ost **o**ften in the **m**inimal size clauses.
  - Jeroslow-Wang's heuristics: estima la contribución que cada literal podria hacer a la satisfiabilidad del conjunto de cláusulas.

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

DP: Performance

DP: Performance

- La complegidad de peor caso del algoritmo que presentamos es $O(1,696^n)$, y con algunas modificaciones menores se puede lograr $O(1,618^n)$.
- Notar que esto es una mejora importante!...

DP: Performance

- La complegidad de peor caso del algoritmo que presentamos es $O(1,696^n)$, y con algunas modificaciones menores se puede lograr $O(1,618^n)$.
- Notar que esto es una mejora importante!...

$$
\begin{aligned}
2^{100} &= 1{,}267{,}650{,}000{,}000{,}000{,}000{,}000{,}000{,}000 \\
1{,}696^{100} &= 87{,}616{,}270{,}000{,}000{,}000{,}000{,}000 \\
1{,}618^{100} &= 790{,}408{,}700{,}000{,}000{,}000{,}000
\end{aligned}
$$

DP: Performance

- La complegidad de peor caso del algoritmo que presentamos es $O(1,696^n)$, y con algunas modificaciones menores se puede lograr $O(1,618^n)$.
- Notar que esto es una mejora importante!...

$$
\begin{aligned}
2^{100} &= 1{,}267{,}650{,}000{,}000{,}000{,}000{,}000{,}000{,}000 \\
1{,}696^{100} &= 87{,}616{,}270{,}000{,}000{,}000{,}000{,}000 \\
1{,}618^{100} &= 790{,}408{,}700{,}000{,}000{,}000{,}000
\end{aligned}
$$

- DP puede resolver problemas de hasta 500 variables en forma robusta

DP: Performance

- La complegidad de peor caso del algoritmo que presentamos es $O(1,696^n)$, y con algunas modificaciones menores se puede lograr $O(1,618^n)$.
- Notar que esto es una mejora importante!...

$$
\begin{aligned}
2^{100} &= 1{,}267{,}650{,}000{,}000{,}000{,}000{,}000{,}000{,}000 \\
1{,}696^{100} &= 87{,}616{,}270{,}000{,}000{,}000{,}000{,}000 \\
1{,}618^{100} &= 790{,}408{,}700{,}000{,}000{,}000{,}000
\end{aligned}
$$

- DP puede resolver problemas de hasta 500 variables en forma robusta
- (Para fórmulas CNF donde la probabilidad de una variable apareciendo positiva, negativa o no apareciendo es 1/3, DP necesita ne promedio solo tiempo cuadrático.)

Métodos Incompletos: Motivación

Métodos Incompletos: Motivación

- DP puede resolver problemas de hasta 500 variables en forma robusta . . .

Métodos Incompletos: Motivación

- DP puede resolver problemas de hasta 500 variables en forma robusta . . .
- . . . pero los problemas que comunmente aparecen en la práctica tiene 1000s de variables

Métodos Incompletos: Motivación

- DP puede resolver problemas de hasta 500 variables en forma robusta . . .
- . . . pero los problemas que comunmente aparecen en la práctica tiene 1000s de variables
- Dependiendo de la aplicación, precidimientos de semi-decisión pueden ser útiles: encontrar una solución si una solución existe

Métodos Incompletos: Motivación

- DP puede resolver problemas de hasta 500 variables en forma robusta . . .
- . . . pero los problemas que comunmente aparecen en la práctica tiene 1000s de variables
- Dependiendo de la aplicación, precidimientos de semi-decisión pueden ser útiles: encontrar una solución si una solución existe
- E.g., existencia de un plan $\equiv$ encontrar un modelo, y no nos interesan los casos en los que un plan no existe

Métodos Incompletos: Motivación

- DP puede resolver problemas de hasta 500 variables en forma robusta . . .
- . . . pero los problemas que comunmente aparecen en la práctica tiene 1000s de variables
- Dependiendo de la aplicación, precidimientos de semi-decisión pueden ser útiles: encontrar una solución si una solución existe
- E.g., existencia de un plan $\equiv$ encontrar un modelo, y no nos interesan los casos en los que un plan no existe
- Por otro lado, ciertas aplicaciones requieren anytime answers, i.e., “best guess” en cualquier punto del algoritmo.

Revoleando Monedas: El Algoritmo “Greedy”

Revoleando Monedas: El Algoritmo “Greedy”

- Algoritmo original de Koutsopias y Papadimitriou
  - Idea Principal: flip variables hasta que no pude incrementarse el número de clausulas satisfechas.

procedure greedy(Sigma)

Revoleando Monedas: El Algoritmo “Greedy”

- Algoritmo original de Koutsopias y Papadimitriou
  - Idea Principal: flip variables hasta que no pude incrementarse el número de clausulas satisfechas.

```
procedure greedy(Sigma)
  T := random(Sigma) // random assignment
```

Revoleando Monedas: El Algoritmo “Greedy”

- Algoritmo original de Koutsopias y Papadimitriou
  - Idea Principal: flip variables hasta que no pude incrementarse el número de clausulas satisfechas.

```
procedure greedy(Sigma)
  T := random(Sigma) // random assignment
  repeat until no improvement possible
```

Revoleando Monedas: El Algoritmo “Greedy”

- Algoritmo original de Koutsopias y Papadimitriou
  - Idea Principal: flip variables hasta que no pude incrementarse el número 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
```

Revoleando Monedas: El Algoritmo “Greedy”

- Algoritmo original de Koutsopias y Papadimitriou
  - Idea Principal: flip variables hasta que no pude incrementarse el número 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
```

Revoleando Monedas: El Algoritmo “Greedy”

- Algoritmo original de Koutsopias y Papadimitriou
  - Idea Principal: flip variables hasta que no pude incrementarse el número 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 todo problema satisfacible con $O(n^2)$ clauses (lamentablemente, no son muchos.)

El Algoritmo GSAT

El Algoritmo GSAT

- El algoritmo original es de Selman, Levesque y Mitchell
  - Agrega restart al algoritmo Greedy, y permite tambien flips ‘hacia el costado’ (que no necesariamente incremental la función de costo)

El Algoritmo GSAT

- El algoritmo original es de Selman, Levesque y Mitchell
  - Agrega restart al algoritmo Greedy, y permite tambien flips 'hacia el costado' (que no necesariamente incremental la función de costo)

```
procedure GSAT(Sigma)
  for i := 1 to MAX-TRIES // Los restarts restarts
    T := random(Sigma) // random assignment
    for j := 1 to MAX-FLIPS // para asegurar terminación
      if T satisfies Sigma then return T
      else T := T with variable flipped to maximize
                  number of satisfied clauses
      // No importa si el número de clausulas satisfechas no se incremente
      // estos son los saltos de costado
    end
  end
```

GSAT: Evaluación

GSAT: Evaluación

formulas	
var	clauses
50	215
100	430
140	602
150	645
300	1275
500	2150

GSAT: Evaluación

formulas		GSAT		
var	clauses	M-FLIPS	restarts	time
50	215	250	6.4	0.4s
100	430	500	42.5	6s
140	602	700	52.6	14s
150	645	1500	100.5	45s
300	1275	6000	231.8	12m
500	2150	10000	995.8	1.6h

GSAT: Evaluación

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	84×10^3	19	2.8m
140	602	700	52.6	14s	2.2×10^6	27	4.7h
150	645	1500	100.5	45s	–	–	–
300	1275	6000	231.8	12m	–	–	–
500	2150	10000	995.8	1.6h	–	–	–

GSAT: Evaluación

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	84×10^3	19	2.8m
140	602	700	52.6	14s	2.2×10^6	27	4.7h
150	645	1500	100.5	45s	–	–	–
300	1275	6000	231.8	12m	–	–	–
500	2150	10000	995.8	1.6h	–	–	–

- GSAT es especialmente bueno para ciertos problemas (random 3-SAT, *n*-queens, etc.)

Y Todo Esto en NP!

Y Todo Esto en NP!

- El conjunto de tautologías de la lógica de primer orden es indecidible pero recursivamente enumerable: existe un algoritmo que produce la lista completa de todos los teoremas (el problema es que no hay garantía de cuándo aparecera el teorema que buscamos).

Y Todo Esto en NP!

- El conjunto de tautologías de la lógica de primer orden es indecidible pero recursivamente enumerable: existe un algoritmo que produce la lista completa de todos los teoremas (el problema es que no hay garantía de cuándo aparecera el teorema que buscamos).
- La lógica de segundo orden es altamente indecidible y no recursivamente enumerable: no existe un programa que pueda listar sus tautologías.

The Land in Between: PSPACE, ExpTime, NExpTime

The Land in Between: PSPACE, ExpTime, NExpTime

- El problema SAT de ciertos fragmentos del la lógica de primer orden es decidible, con complejidad en PSPACE, ExpTime, NExpTime.

The Land in Between: PSPACE, ExpTime, NExpTime

- El problema SAT de ciertos fragmentos del la lógica de primer orden es decidible, con complejidad en PSPACE, ExpTime, NExpTime.
- En particular, yo trabajo en ciertos lenguajes llamados Lógicas Modales.

The Land in Between: PSPACE, ExpTime, NExpTime

- El problema SAT de ciertos fragmentos del la lógica de primer orden es decidible, con complejidad en PSPACE, ExpTime, NExpTime.
- En particular, yo trabajo en ciertos lenguajes llamados Lógicas Modales.
- Pueden pensarse como lenguajes diseñados a medida para hablar sobre estructuras relacionales (grafos).

The Land in Between: PSPACE, ExpTime, NExpTime

- El problema SAT de ciertos fragmentos del la lógica de primer orden es decidible, con complejidad en PSPACE, ExpTime, NExpTime.
- En particular, yo trabajo en ciertos lenguajes llamados Lógicas Modales.
- Pueden pensarse como lenguajes diseñados a medida para hablar sobre estructuras relacionales (grafos).
- Si bien la complejidad de su problema SAT es más elevado que el de LP, en muchos casos son una alternativa atractiva
  - La expresividad de LP puede ser simplemente insuficiente para una cierta tarea
  - O la representación en LP puede ser mucho mas complicada/compleja que la que se obtiene usando un lenguaje modal.

Aplicaciones de Lenguajes Modales

Aplicaciones de Lenguajes Modales

Para nombrar sólo algunas aplicaciones:

- Verificación de Software (lógicas temporales, lógicas de tiempo real, lógicas dinámicas).
- Teoría de Agentes (lenguajes epistémicos)
- Especificación de Protocolos
- Problemas de Criptologia (BDI, Belief, Desires, Intentions)
- Descripción de Ontologías (Lógicas para la Descripción)

Aplicaciones de Lenguajes Modales

Para nombrar sólo algunas aplicaciones:

- Verificación de Software (lógicas temporales, lógicas de tiempo real, lógicas dinámicas).
- Teoría de Agentes (lenguajes epistémicos)
- Especificación de Protocolos
- Problemas de Criptologia (BDI, Belief, Desires, Intentions)
- Descripción de Ontologías (Lógicas para la Descripción)

Mas información?

Aplicaciones de Lenguajes Modales

Para nombrar sólo algunas aplicaciones:

- Verificación de Software (lógicas temporales, lógicas de tiempo real, lógicas dinámicas).
- Teoría de Agentes (lenguajes epistémicos)
- Especificación de Protocolos
- Problemas de Criptologia (BDI, Belief, Desires, Intentions)
- Descripción de Ontologías (Lógicas para la Descripción)

Mas información?

- Curso optativo en Lógicas Modales Computacionales el año que viene!!!