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 filosoía:
  - 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: $p_1, p_2, p_3, \ldots$
  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	$\neg A$	$\equiv$	no salgo con Ana
B	$\equiv$	salgo con Beatriz	$\neg B$	$\equiv$	no salgo con Beatriz
C	$\equiv$	salgo con Celeste	$\neg C$	$\equiv$	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?

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

- 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 B = verdadero C = verdadero
A = falso B = falso 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

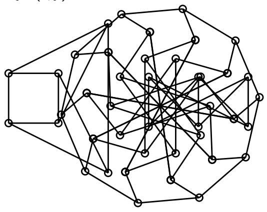
- Métodos Completos
  - Resolucion
  - Tableaux
  - Davis-Putman
  - Mapeo en ecuaciones lineares
- Métodos de Aproximación
  - Cambiar el valor de una variable en una fórmula insatisfecha
  - Algoritmos géneticos
  - Hill-climbing

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 específicos 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:

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

El conjunto de cláusulas asociado a

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

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

- 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 ≡ 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	15m	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.

Lógica Computacional y Demostración Automática

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

INRIA Nancy Grand Est, France

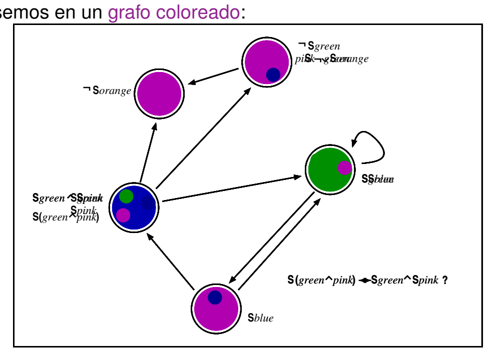
Diciembre 2008

Lo que hacemos hoy

- Describiendo estructuras relacionales
- Lógicas modales
- Sintaxis y semántica
- Lógicas híbridas
- Model Checking
- Aplicaciones
- El model checker `mcheck`

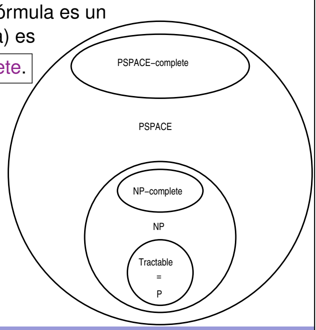
Estructuras Simples / Lenguajes Simples

Pensemos en un grafo coloreado:



Estructuras Simples / Lenguajes Simples

- Aun para este lenguaje (aparentemente) muy simple, decidir si una fórmula es un teorema (i.e., siempre cierta) es PSPACE-complete.



Una Familia de Lenguajes

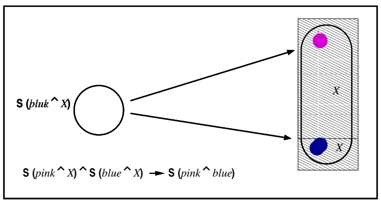
- El lenguaje que describimos puede ser inadecuado:
- En área de lógica modal, investiga el rango de posibles lenguajes que pueden usarse para describir estructuras relacionales.
- La preguntas más importantes son:
  - Podemos definir algoritmos de inferencia para estos lenguajes?
  - Cuan eficientes son?
  - Cuáles son los límites de expresividad de estos lenguajes?
- La tarea no es fácil, porque los distintos operadores del lenguaje pueden interactuar de formas inesperadas. E.g., agregar el operador S* transforma el problema de satisfiabilidad del lenguaje en EXPTIME-complete!

Posibles Aplicaciones

- Los lenguajes de la familia de las lógicas modales pueden usarse en áreas muy diversas:
  - Verificación de Software y Hardware.
  - Representación de Conocimientos.
  - Criptografía.
  - Inteligencia Artificial.
  - Filosofía.
  - Linguística Computacional.
  - Epistemología.
  - ...
- **Porque?** Muchas cosas pueden ser representadas como grafos (i.e., estructuras relacionales).
  Y como vimos, los lenguajes modales fueron desarrollados especialmente para razonar sobre grafos y describir sus propiedades.

Lógicas Híbridas

- Vamos a presentar un tipo particular de lenguajes modales llamados lenguajes híbridos.
- Intuitivamente, los lenguajes híbridos son lenguajes modales que incluyen alguna noción de igualdad.



El Sabor de un Clásico...

- Todo muy moderno, muy superado, pero yo prefiero la lógica clásica.
- Aunque no parezca, estamos haciendo Lógica Clásica!!!
- Los lenguajes que estuvimos discutiendo son fragmentos del lenguaje de primer (o segundo) orden. Lo único que hicimos fue elegir 'el pedacito' que necesitábamos para una aplicación dada.
- Esta es exactamente la forma en que vemos hoy por hoy a los lenguajes modales, como una forma de investigar fragmentos particularmente interesantes de los lenguajes clásicos.
- Donde "interesantes" significa
  - Decidibles, expresivos, de "baja" complejidad, modulares, ...

Sintaxis

- El lenguaje hibrido que vamos a usar se define sobre la signatura $S = \langle \text{PROP}, \text{REL}, \text{NOM}\rangle$ donde
  - $\text{PROP} = \{p, q, r, \ldots\}$ es el conjunto de simbolos de proposicion
  - $\text{REL} = \{r_1, r_2, r_3, \ldots\}$ es el conjunto de simbolos de relacion
  - $\text{NOM} = \{i, j, k, \ldots\}$ es el conjunto de simbolos de constantes (los llamos nominales)
  - $\text{VAR} = \{x, y, z, \ldots\}$ es el conjunto de variables
- Las formulas se definen entonces como

$$\varphi \ := \ a \mid \neg\varphi \mid \varphi \wedge \varphi' \mid \langle r\rangle\varphi \mid \langle r\rangle^{-}\varphi \mid \mathsf{E}\varphi \mid @_i\varphi \mid {\downarrow} x.\varphi$$

donde $a \in \text{NOM} \cup \text{PROP} \cup \text{VAR}$, $r \in \text{REL}$, $i \in \text{NOM}$ y $x \in \text{VAR}$.(La formula $[r]\varphi$ se define como $\neg\langle r\rangle\neg\varphi$ y $\mathsf{A}\varphi$ se define como $\neg\mathsf{E}\neg\varphi$)

Semantica

- Un modelo $\mathcal{M}$ es de la forma $\langle M, I\rangle$ donde M es no vacio, y
  - $I(r)$ es una relacion binaria sobre M para $r \in \text{REL}$,
  - $I(p)$ es un subconjunto de M para $p \in \text{PROP}$,
  - $I(i)$ es un elemento de M para $i \in \text{NOM}$

  Una asignación g para $\mathcal{M}$ es una funcion $g : \text{VAR} \to M$.
- Definimos

$$\begin{array}{rcl}
\mathcal{M}, g, m \models i & \text{sii} & m = I(i),\ i \in \text{NOM} \\
\mathcal{M}, g, m \models p & \text{sii} & m \in I(p),\ p \in \text{PROP} \\
\mathcal{M}, g, m \models \neg\varphi & \text{sii} & \mathcal{M}, g, m \not\models \varphi \\
\mathcal{M}, g, m \models \varphi \wedge \psi & \text{sii} & \mathcal{M}, g, m \models \varphi \text{ y } \mathcal{M}, g, m \models \psi \\
\mathcal{M}, g, m \models \langle r\rangle\varphi & \text{sii} & \exists m' \in M \text{ t.q. } (m, m') \in I(r) \ \& \ \mathcal{M}, g, m' \models \varphi \\
\mathcal{M}, g, m \models \langle r\rangle^{-}\varphi & \text{sii} & \exists m' \in M \text{ t.q. } (m', m) \in I(r) \ \& \ \mathcal{M}, g, m' \models \varphi \\
\mathcal{M}, g, m \models \mathsf{E}\varphi & \text{sii} & \exists m' \in M \text{ t.q. } \mathcal{M}, g, m' \models \varphi \\
\mathcal{M}, g, m \models @_i\varphi & \text{sii} & \mathcal{M}, g, I(i) \models \varphi \\
\mathcal{M}, g, m \models {\downarrow} x.\varphi & \text{sii} & \mathcal{M}, g[x \leftarrow m], m \models \varphi
\end{array}$$

Que podemos escribir?

- Cuán expresivo es el lenguaje $\mathcal{L}(\text{NOM}, @, \langle r\rangle, \langle r\rangle^{-}, E, \downarrow)$
- Empecemos con menos (y para los que saben...)
- Cual es el poder expresivo de $\mathcal{L}(\langle r\rangle, \langle r\rangle^{-})$
  - Es el lenguaje temporal básico (futuro y pasado)
  - SAT es decidible (PSPACE-complete).
  - El fragmento guarded de LPO^2.
- Cual es el poder expresivo de $\mathcal{L}(\text{NOM}, \langle r\rangle, \langle r\rangle^{-})$
  - Es el lenguaje temporal hibrido básico
  - SAT es decidible (EXPTIME-complete).
  - El fragmento guarded de LPO^2 con constantes.
- Cual es el poder expresivo de $\mathcal{L}(\langle r\rangle, \downarrow, E)$
  - Es tan expresivo como LPO!!!
  - SAT es indecidible.
- Cual es el poder expresivo de $\mathcal{L}(\langle r\rangle, \downarrow)$
  - Es menos expresivo que LPO.
  - SAT es todavia indecidible.

Model Checking

- El problema de model checking global es el siguiente:
  - Dado un modelog $\mathcal{M}$
  - y una formula α,
  - retornar todos los estados de $\mathcal{M}$ en los que α es verdadero.
- Definimos el truth set de una fórmula α respecto de un modelo $\mathcal{M} = \langle W, R, V\rangle$ como:

$$\mathit{Truth}(\mathcal{M}, \alpha) = \{w \in W \mid \mathcal{M}, w \models \alpha\}$$

- Un model checker es un programa que dado $\mathcal{M}$ y α retorna $\mathit{Truth}(\mathcal{M}, \alpha)$.

El Model Checker MCLite

- MCLite es un model checker para el lenguaje $\mathcal{L}(\mathit{NOM}, \langle r\rangle, \langle r\rangle^{-}, @, E)$.
- Por simplicidad, trabajaremos con lógicas monomodales (sólo $\langle r\rangle$) pero la extensión a lógicas multimodales es simple.
- El algoritmo usa una estrategia bottom-up: examina las subfórmulas de la fórmula input α en forma incremental, hasta que finalmente el obtenemos el truth set de α.
- Si una subfórmula β de α es verdadera en un estado w, el model checker marca w con β.

Tipos de Datos y Funcionaes Auxiliares

- Sea $\mathcal{M} = \langle W, R, V\rangle$ un modelo y α la fórmula que queremos chequear.
- Sea $\mathit{sub}(\alpha)$ el conjunto de subfórmulas de α.
- El model checker mantendrá una bit table L de tamaño $|\mathit{sub}(\alpha)| \times |W|$ tal que, para cada subfórmula β de α y cada $w \in W$,
  $L(\beta, w) = 1$ if $M, w \models \beta$ y $L(\beta, w) = 0$ if $M, w \not\models \beta$
- Ademas, sea $L(\beta) = \{w \in W \mid L(\beta, w) = 1\}$
- Al terminar la corrida tendremos que $\mathit{Truth}(M, \alpha) = L(\alpha)$
- Finalente dado $w \in W$, sea $R(w)$ el conjunto de R-sucesores de w y $R^{-}(w)$ el conjunto de R-predecesores de w.

MCLite(M, α)

for $\beta \in \mathit{sub}(\alpha)$, $w \in W$ **do:** $L(\beta, w) \leftarrow 0$
for $i \leftarrow 1$ **to** $|\alpha|$ **do:**
  for $\beta \in \mathit{sub}(\alpha)$ **such that** $|\beta| = i$ **do:**
    case of β
      • $\beta \in \mathit{ATOM}$ **:** $\forall w \in W$, **if** $w \in V(\beta)$ **then** $L(\beta, w) \leftarrow 1$
      • $\beta = \beta_1 \wedge \beta_2$ **:** $\forall w \in W$, **if** $L(\beta_1, w) = L(\beta_2, w) = 1$ **then** $L(\beta, w) \leftarrow 1$
      • $\beta = \neg\beta_1$ **:** $\forall w \in W$, **if** $L(\beta_1, w) = 0$ **then** $L(\beta, w) \leftarrow 1$
      • $\beta = \langle r\rangle\beta_1$ **:** $\mathbf{MC}_{\langle r\rangle}(M, \beta_1)$
      • $\beta = \langle r\rangle^{-}\beta_1$ **:** $\mathbf{MC}_{\langle r\rangle^{-}}(M, \beta_1)$
      • $\beta = E\beta_1$ **:** $\mathbf{MC}_{E}(M, \beta_1)$
      • $\beta = @_i\beta_1$ **:** $\mathbf{MC}_{@}(M, i, \beta_1)$
return $L(\alpha)$

$\mathbf{MC}_{\langle r\rangle}(M, \alpha)$

for $w \in L(\alpha)$ **do**
  for $v \in R^{-1}(w)$ **do**
    $L(\langle r\rangle\alpha, v) \leftarrow 1$

$\mathbf{MC}_{\langle r\rangle^{-}}(M, \alpha)$

for $w \in L(\alpha)$ **do**
  for $v \in R(w)$ **do**
    $L(\langle r\rangle^{-}\alpha, v) \leftarrow 1$

$\mathbf{MC}_{E}(M, \alpha)$

if $L(\alpha) \neq \emptyset$ **then**
  for $w \in W$ **do**
    $L(E\alpha, w) \leftarrow 1$

$\mathbf{MC}_{@}(M, i, \alpha)$

let $\{v\} = V(i)$
if $L(\alpha, v) = 1$ **then**
  for $w \in M$ **do**
    $L(@_i\alpha, w) \leftarrow 1$

Worst-case Complexity de MCLite

- Dado $f, g : N \to N$, recordemos que

$$f(n) = O(g(n))$$

significa que hay una constante $c > 0$ y un número natural $n_0 \geq 1$ tal que

$$f(n) \leq c \times g(n)$$

para todo $n \geq n_0$.
- Por ejemplo, $2n + 1 = O(n)$.

Contamos...

- Sea $n = |W|$, $m = |R|$, y k el tamaño de α.
- Notar que $|sub(\alpha)| = O(k)$. Ademas, chequear y cambiar $L(\beta, w)$ es $O(1)$. Igual que $w \in V(p)$ si V es un bit table.
- Entonces, inicializar L toma $O(k \times n)$. El loop principal itera $O(k)$ veces. La complejidad de cada iteracion depende de β.
- In particular,
  - if $\beta \in ATOM$, then the check of β takes $O(n)$;
  - if $\beta = \beta_1 \wedge \beta_2$, then the check of β takes $O(n)$;
  - if $\beta = \neg\beta_1$, then the check of β takes $O(n)$;
  - if $\beta = E\beta_1$, then the check of β takes $O(2n) = O(n)$;
  - if $\beta = @_i\beta_1$, then the check of β takes $O(2n) = O(n)$;
  - if $\beta = \langle r \rangle\beta_1$, then the check of β takes $O(n + m)$;
  - if $\beta = \langle r \rangle^-\beta_1$, then the check of β takes $O(n + m)$
- Es decir, la complejidad de MCLite es: $O(k \times (n + m))$.

El problema con $\downarrow$

- Cuando agregamos $\downarrow$ no podemos usar una estrategia bottom-up.
- Por que? Consideremos las formulas $\langle r \rangle\alpha$ y $\downarrow x.\langle r \rangle\beta(x)$, donde $\beta(x)$ es una formula donde aparece la variable x.
  - En el primer caso, podemos chequear α, etiquetar el modelo, y luego chequear $\langle r \rangle\alpha$ como hicimos en MCLite.
  - En el segundo caso, no podemos chequear primero $\beta(x)$, porque no sabemos a qué elemento esta linqueado x.

MCFull(M, g, α)

for $\beta \in sub(\alpha)$, $w \in W$ **do:** $L(\beta, w) \leftarrow 0$
case of α**:**
$\bullet\ \alpha \in ATOM$: $\forall w \in W$, **if** $w \in V(\alpha)$ **then** $L(\alpha, w) \leftarrow 1$
$\bullet\ \alpha \in VAR$: $L(\alpha, g(\alpha)) \leftarrow 1$
$\bullet\ \alpha = \alpha_1 \wedge \alpha_2$: **MCFull**(M, g, α_1); **MCFull**(M, g, α_2)
  for $w \in W$ **do:**
    if $L(\alpha_1, w) = 1 = L(\alpha_2, w) = 1$ **then** $L(\alpha, w) \leftarrow 1$
$\bullet\ \alpha = \neg\alpha_1$: **MCFull**(M, g, α_1)
  for $w \in W$ **do:**
    if $L(\alpha_1, w) = 0$ **then** $L(\alpha, w) \leftarrow 1$
$\bullet\ \alpha = \langle r \rangle\alpha_1$: **MCFull**(M, g, α_1); **MC**$_{\langle r \rangle}(M, \alpha_1)$
$\bullet\ \alpha = \langle r \rangle^-\alpha_1$: **MCFull**(M, g, α_1); **MC**$_{\langle r \rangle^-}(M, \alpha_1)$
$\bullet\ \alpha = @_t\alpha_1$: **MCFull**(M, g, α_1); **MC**$_@(M, t, g, \alpha_1)$ //**almost**
$\bullet\ \alpha = E\alpha_1$: **MCFull**(M, g, α_1); **MC**$_E(M, \alpha_1)$
$\bullet\ \alpha = \downarrow x.\alpha_1$: **Check**$_\downarrow(M, g, x, \alpha_1)$
return $L(\alpha)$

Check$_\downarrow(M, g, x, \alpha)$

for $w \in W$ **do:**
  $g(x) \leftarrow w$
  MCFull(M, g, α)
  if $L(\alpha, w) = 1$ **then**
    Clear(L, x)
    $L(\downarrow x.\alpha, w) \leftarrow 1$
  else
    Clear(L, x)

La funcion Clear(L, x) pone a 0 el valor de L(α, w) para todo $w \in W$ y toda formula α con x libre.

Complejidad de MCFull

- Supongamos que α no contiene el operador $\downarrow$. Sea $\alpha = \tau\alpha_1$, donde τ es el operador principal de α.
- Sea $C(\alpha)$ el costo de MCFull en α y sea $C_\tau(\alpha)$ el costo de MC_τ en α. Entonces,

$$C(\tau\alpha_1) = C(\alpha_1) + C_\tau(\alpha)$$

- Por lo que, en el peor caso, el costo de MCFull en α es $O(k \times (n + m))$ como para MCLite.
- Por ejemplo, si $\alpha = \langle r \rangle\langle r \rangle\langle r \rangle p$, entonces

$$\begin{aligned} C(\alpha) = C(\langle r \rangle\langle r \rangle p) + (n + m) = \\ C(\langle r \rangle p) + 2(n + m) = \\ C(p) + 3(n + m) = n + 3(n + m) \end{aligned}$$

Complejidad de MCFull

- Supongamos ahora que α contiene $\downarrow$.
- Sea d el grado de anidamiento de $\downarrow$ en α. Por ejemplo $\downarrow x.\langle r \rangle x$ tiene grado 1, y $\downarrow x.\langle r \rangle\downarrow y.@_x y$ tiene grado 2.
- La función Check$_\downarrow(M, g, x, \beta)$ corre en $n \times C(\beta)$.
- Por lo que, en el peor caso, el costo de MCFull en α es $O(k \times (n + m) \times n^d)$.
- Por ejemplo, si $\alpha = \downarrow x.\langle r \rangle\downarrow y.@_x y$, entonces

$$\begin{aligned} C(\alpha) = n \times C(\langle r \rangle\downarrow y.@_x y) = \\ n \times (n + m + C(\downarrow y.@_x y)) = \\ n \times (n + m + n \times C(@_x y)) = \\ n \times (n + m + n \times (n + C(y))) = \\ n \times (n + m + n \times (n + 1)) = O(n^3) \end{aligned}$$

Complejidad de MCFull

- Es decir, la complejidad temporal de MCFull es exponential en el grado de anidammiento de $\downarrow$ que, en general, es proporcional al tamaño de α.
- Como el tamaño del stack de recursion de MCFull está limitado por el tamaño de α, la complejidad espacial de MCFull es polynomial.
- Es decir, model checking para $\mathcal{L}(\text{NOM}, \langle r \rangle, \langle r \rangle^-, @, E, \downarrow)$ esté en PSPACE.
- Nos podemos preguntar: es este el mejor algoritmo?

Lower bounds

- Sabemos que $\mathcal{L}(\mathsf{NOM}, \langle r \rangle, \langle r \rangle^{-}, @, E, \downarrow)$ tiene la misma expresividad que LPO.
- Es conocido que el problema de model checking para LPO es PSPACE-complete, por lo tanto también para $\mathcal{L}(\mathsf{NOM}, \langle r \rangle, \langle r \rangle^{-}, @, E, \downarrow)$.
  Pero podemos demostrar un resultado mas fuerte.
- Llamemos, fragmento no decorado de L al fragmento de L que no usa ni símbolos de proposición ni nominales
  Theorem: El problema de model checking para el fragmento no decorado de $\mathcal{L}(\downarrow)$ es PSPACE-complete.

El model checker `mcheck`

- Es un model checker de juguete.
- Warning: no intentar usarlo para nada serio.
- Si estan interesados en model checkers para lógicas hibridas hay otras alternativas pero el input format de `mcheck` es mas simple.

Lógica Computacional y Demostración Automática

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

INRIA Nancy Grand Est, France

Diciembre 2008

Lógicas para la Descripción

- Las lógicas para la descripción (description logics, DL) son lenguajes formales especialmente diseñados para la representación del conocimiento.
- Descienden originalmente del trabajo en redes semánticas (semantic networks) de Quillian y el paradigma de Frames de Minsky.
- Las principales características de estos lenguajes son:
  - Un lenguaje simple de usar (una extensión del lenguaje proposicional, sin variables de estado);
  - Que incluye una noción restringida de cuantificación;
  - Con operadores especialmente elegidos para facilitar la creación de definiciones;
  - Con un buen balance entre expresividad y tratabilidad;
  - Que cuenta con sistemas de inferencia altamente optimizados.

Lógicas para la Descripción

En una DL tenemos operadores para construir definiciones usando conceptos y roles:

- Los conceptos corresponden a “Clases de Elementos” y seran interpretados como subconjuntos del universo.
- Los roles corresponden a “Vinculos entre Elementos” y seran interpretados como relaciones binarias en el universo.

Ejemplo: El “Padre Contento”

$\text{Conceptos} = \{$ Hombre, Mujer, Contento, Rico $\}$
$\text{Roles} = \{$ tiene-hijo $\}$

$$\begin{aligned}\text{PadreContento} \equiv\ & \text{Hombre} \sqcap \\ & \exists\,\text{tiene-hijo.Hombre} \sqcap \\ & \exists\,\text{tiene-hijo.Mujer} \sqcap \\ & \forall\,\text{tiene-hijo.(Contento} \sqcup \text{Rico)}\end{aligned}$$

carlos:$\neg$PadreContento

Areas de Aplicación

- Bases de conocimiento terminológicas y ontologías
  - DLs fueron desarrolladas para esta tarea
  - Especialmente útiles como lenguaje de definición y mantenimiento de ontologías
- Applicaciones en Bases de Datos
  - DLs pueden capturar la semantica de varias metodologías de modelado en BD e.g., diagramas de ER, UML, etc
  - los motores de inferencia DL pueden usarse para proveer soporte durante el diseño de diagramas, mantenimiento y consulta

Areas de Aplicación

- Web Semántica
  - Agregar ‘markup semántico’ a la información en la web
  - Este markup usaría repositorios de ontologías como repositorío común de definiciones con una semántica clara
  - los motores de inferencia DL se usarían para el desarrollo, mantenimiento y fusión de estas ontologías y para la evalucion dynámica de recursos (e.g., búsqueda).
- Linguística Computacional
  - Muchas tareas en linguística computacional requieren inferencia y 'background knowledge': desde tareas puntuales como resolución de referencias a problemas generales como question-answering.
  - En ciertos casos, el poder expresivo ofrecido por DLs es suficiene y no necesitamos recurrir a LPO.

Lógicas para la Descripción

- El lenguaje se define en tres niveles.
  - Conceptos: Por un lado construimos conceptos complejos usando conceptos (atómicos o creados via definición) y roles: E.g., $\exists$tiene-hijo.Hombre
  - Definiciones: Usamos conceptos para armar definiciones (o relaciones entre definiciones): E.g., PadreContento $\equiv \ldots$
  - Aserciones: Podemos asignar conceptos o roles a elementos particulares de nuestro modelo: E.g., carlos:$\neg$PadreContento
- Una base de conocimiento es un par $\langle D, A\rangle$ con D un conjunto de definiciones (la “T-Box”) y A un conjunto de aserciones (la “A-Box”).

$\mathcal{ALC}$: Construcción de Conceptos

- Un concepto puede ser
  - $\top$, el concepto trivial, del que todo elemento es miembro.
  - Un concepto atómico: Hombre, Mujer
  - Operadores booleanos: Si C y D son conceptos entonces los siguientes son conceptos

$C \sqcap D$	la conjunción de C y D	Rico $\sqcap$ Apuesto
$C \sqcup D$	la disjunción de C y D	Rico $\sqcup$ Apuesto
$\neg C$	la negación de C	$\neg$Politico

  - Operadores relacionales: Si C es un concepto y R es un role, los siguientes son conceptos

$\forall R.C$	todo elem. acc. via R está en C	$\forall$hijo-de.Mujer
$\exists R.C$	un elem. acc. via R está en C	$\exists$hijo-de.Mujer

Construcción de Definiciones

La T-Box es una lista de definiciones.
Dados dos conceptos C y D, hay dos tipos de definiciones:

- Definiciones Parciales: $C \sqsubseteq D$. Las condiciones indicadas en C son suficientes para calificar a los elementos de C como miembros de D, pero no son condiciones necesarias; o vice-versa.

$\exists$hijo-de.Hombre $\sqcap$ $\exists$hijo-de.Mujer $\sqsubseteq$ PadreOcupado (condición suficiente)
PadreOcupado $\sqsubseteq$ $\exists$hijo-de.$\top$ (condición necesaria)

- Definiciones Totales: $C \equiv D$. Las condiciones indicadas en D son necesarias y suficientes para calificar a los elementos de D como miembros de C. Los conceptos C y D son equivalentes.

Abuela $\equiv$ Mujer $\sqcap$ $\exists$hijo-de.$\exists$hijo-de.$\top$

Construcción de Aserciones

Podemos "asignar propiedades" a elementos particulares de la situación que estamos describiendo en la A-Box.
Dados elementos a y b, un concepto C y una relacion R

- Asignación de Elementos a Conceptos: $a{:}C$. Indica que el concepto C es applicable al elemento a. Es decir, todas las condiciones indicadas por C se aplican a a.

$$\text{carlos}{:}\text{Argentino}$$
$$\text{carlos}{:}(\text{Argentino} \sqcap \exists\text{vive-en.Europa})$$

- Asignación de Relaciones entre Elementos: $(a, b){:}R$. Indica que los elementos a y b estan relacionados via el rol R.

$$(\text{carlos},\text{nancy}){:}\text{vive-en}$$

Semántica: Modelos

Sea CON un conjunto de conceptos atómicos, ROL un conjunto de roles y IND un conjunto de individuos.
Una interpretación o un modelo para $\mathcal{ALC}$ es un par $\mathcal{I} = \langle \Delta^{\mathcal{I}}, \cdot^{\mathcal{I}} \rangle$ tal que

- $\Delta^{\mathcal{I}}$ es un conjunto no vacío arbitrario
- $\cdot^{\mathcal{I}}$ es una función de interpretación de conceptos atómicos, roles e individuos tal que

$$C^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \text{ para } C \in \text{CON}$$
$$R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}} \text{ para } R \in \text{ROL}$$
$$a^{\mathcal{I}} \in \Delta^{\mathcal{I}} \text{ para } a \in \text{IND}$$

(i.e., un modelo de DL no es otra cosa que un modelo de LPO para la signatura $\langle \text{CON} \cup \text{ROL}, \{\}, \text{IND} \rangle$)

Semántica: Conceptos

Dado una interpretación $\mathcal{I}$ podemos definir la interpretación de un concepto arbitrario de $\mathcal{ALC}$ recursivamente como

$$\begin{array}{rcl} (\neg C)^{\mathcal{I}} & := & \Delta^{\mathcal{I}} \backslash C^{\mathcal{I}} \\ (C \sqcap D)^{\mathcal{I}} & := & C^{\mathcal{I}} \cap D^{\mathcal{I}} \\ (\exists R.C)^{\mathcal{I}} & := & \{i \mid \exists j.(i,j) \in R^{\mathcal{I}} \text{ y } j \in C^{\mathcal{I}}\} \end{array}$$

$C \sqcup D$ es equivalente a $\neg(\neg C \sqcap \neg D)$ y
$(\forall R.C)$ es equivalente a $\neg(\exists R.\neg C)$.

Semántica: Definiciones y Aserciones

Dada una interpretación $\mathcal{I}$ decimos que

- $\mathcal{I}$ satisface una definición parcial $C \sqsubseteq D$ (definición total $C \equiv D$) sii
$$C^{\mathcal{I}} \subseteq D^{\mathcal{I}} \quad (C^{\mathcal{I}} = D^{\mathcal{I}})$$
- $\mathcal{I}$ satisface una T-Box T sii satisface todas las definiciones (parciales o totales) en T
- $\mathcal{I}$ satisface una asercion $a{:}C$ ($(a,b){:}R$) sii
$$a^{\mathcal{I}} \in C^{\mathcal{I}} \quad ((a^{\mathcal{I}}, b^{\mathcal{I}}) \in R^{\mathcal{I}})$$
- $\mathcal{I}$ satisface una A-Box A sii satisface todas las aserciones en A
- $\mathcal{I}$ satisface una KB $K = \langle T, A \rangle$ sii $\mathcal{I}$ satisface T y A.

Una KB K es consistente (o satisfacible) sii existe una interpretación $\mathcal{I}$ que la satisface.

Un Ejemplo Completo

La T-Box:
$$\begin{array}{rcl} \text{Mujer} & \sqsubseteq & \text{Persona} \sqcap \exists\text{sexo.Femenino} \\ \text{Hombre} & \sqsubseteq & \text{Persona} \sqcap \exists\text{sexo.Masculino} \\ \text{PadreOMadre} & \equiv & \text{Persona} \sqcap \exists\text{hijo-de.Persona} \\ \text{Madre} & \equiv & \text{Mujer} \sqcap \text{PadreOMadre} \\ \text{Padre} & \equiv & \text{Hombre} \sqcap \text{PadreOMadre} \end{array}$$

La A-Box:
$$\begin{array}{c} \text{alicia}{:}\text{Madre} \\ (\text{alicia},\text{betty}){:}\text{hijo-de} \\ (\text{alicia},\text{carlos}){:}\text{hijo-de} \end{array}$$

Sintaxis y Semántica de $\mathcal{ALC}$

Constructor	Sintaxis	Semántica
concepto atómico	C	$C^{\mathcal{I}}$
top	$\top$	$\Delta^{\mathcal{I}}$
negación ($\mathcal{C}$)	$\neg C$	$\Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$
conjunción	$C_1 \sqcap C_2$	$C_1^{\mathcal{I}} \cap C_2^{\mathcal{I}}$
disyunción	$C_1 \sqcup C_2$	$C_1^{\mathcal{I}} \cup C_2^{\mathcal{I}}$
quant. universal ($\mathcal{U}$)	$\forall R.C$	$\{d_1 \mid \forall d_2 \in \Delta^{\mathcal{I}}.(R^{\mathcal{I}}(d_1,d_2) \to d_2 \in C^{\mathcal{I}})\}$
quant. existencial ($\mathcal{E}$)	$\exists R.C$	$\{d_1 \mid \exists d_2 \in \Delta^{\mathcal{I}}.(R^{\mathcal{I}}(d_1,d_2) \wedge d_2 \in C^{\mathcal{I}})\}$

Bases de Conocimiento

Una base de conocimientos K es un par $K = \langle T, A \rangle$ tal que

- T es la T(erminological)-Box, un conjunto finito de expresiones de la forma $C \sqsubseteq D$. Las fórmulas en T se llaman terminological axioms.
- A es la A(ssertional)-Box, un conjunto finito de expresiones de la forma $a{:}C$ o $(a,b){:}R$. Las fórmulas en A se llaman assertions.

Sea $\mathcal{I}$ una interpretación y φ un axioma terminológico o una asercion. Decimos que $\mathcal{I} \models \varphi$ si

- $\varphi = C \sqsubseteq D$ y $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$, o
- $\varphi = a{:}C$ y $a^{\mathcal{I}} \in C^{\mathcal{I}}$, o
- $\varphi = (a,b){:}R$ y $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in R^{\mathcal{I}}$.

Decimos que $\langle T, A \rangle$ es satisfacible si existe una interpretación $\mathcal{I}$ que satisface T y A.

Tareas de Inferencia

La tarea básica de inferencia es satisfiabilidad de conceptos:

INPUT: Un concepto C
OUTPUT: Si, si existe una interpretación $\mathcal{I}$ tal que $C^{\mathcal{I}} \neq \{\}$
No, en otro caso.

Equivalentemente, subsumpción de conceptos:

INPUT: un par de conceptos (C, D)
OUTPUT: Si, si para toda interpretación $\mathcal{I}$, $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$
No, en otro caso.

T: Satisfiabilidad y subsumpción son interdefinibles (en un lenguaje con conjunción y negación).

D: [$\Rightarrow$] C no es satisfacible sii $C \sqsubseteq \neg\top$
[$\Leftarrow$] $C \sqsubseteq D$ sii $C \sqcap \neg D$ no es satisfacible

Tareas de Inferencia Respecto de una Base de Conocimiento

Sea T una base de conocimientos, $C_1, C_2 \in$ CON, $R \in$ ROL y $a, b \in$ IND, podemos definir las siguientes *tareas de inferencia*

- Subsumption, $T \models C_1 \sqsubseteq C_2$. Chequear si para toda interpretación $\mathcal{I}$ tal que $\mathcal{I} \models T$ tenemos que $C_1^{\mathcal{I}} \subseteq C_2^{\mathcal{I}}$.
- Instance Checking, $T \models a{:}C$. Chequear si para toda interpretación $\mathcal{I}$ tal que $\mathcal{I} \models T$ tenemos que $a^{\mathcal{I}} \in C^{\mathcal{I}}$.
- Concept Consistency ($T \not\models C \doteq \bot$). Chequear si para alguna interpretación $\mathcal{I}$ tal que $\mathcal{I} \models T$ tenemos qeu $C^{\mathcal{I}} \neq \{\}$.

Otras: Relation Checking ($T \models (a, b){:}R$), Knowledge Base Consistency ($T \not\models \bot$), etc.

Tareas de Inferencia Complejas

Otras tareas de inferencia mas complejas, pueden definirse a partir de las anteriores, por ejemplo:

- Retrieval: dado un concepto C, retornar todos los individuos mencionados en la base de datos que son instancia de C.
  $\{a \in \text{IND}(T) \mid T \models a{:}C\}$
- Conceptos mas específicos: dado un indivíduo a, retornar los conceptos más específicos en la ontología de los cuales a es un miembro.
- Conceptos parientes (descendientes) inmediatos: dado un concepto C, retornar los conceptos inmediatamente sobre (bajo) C en la jerarquía.

Relación con LPO

La mayoría de los DLs son fragmentos decidibles de LPO.
Tomar un lenguaje de PO que tenga
un predicado unario C por cada concepto atómico C
un predicado binario R por cada rol R
una constante a por cada indivíduo a.
Definimos las siguientes traducciones$t_x : \mathcal{ALC} \to LPO$ y $t_y : \mathcal{ALC} \to LPO$ por recursión mutua

$$\begin{array}{rcl} t_x(C) & = & C(x) \\ t_x(\neg C) & = & \neg t_x(C) \\ t_x(C \sqcap D) & = & t_x(C) \wedge t_x(D) \\ t_x(\exists R.C) & = & \exists y.(R(x,y) \wedge t_y(C)) \end{array} \qquad \begin{array}{rcl} t_y(C) & = & C(y) \\ t_y(\neg C) & = & \neg t_y(C) \\ t_y(C \sqcap D) & = & t_y(C) \wedge t_y(D) \\ t_y(\exists R.C) & = & \exists x.(R(y,x) \wedge t_x(C)) \end{array}$$

Traduciendo Bases de Conocimiento

Una T-Box $T = \{C_i \sqsubseteq D_i \mid i \leq n\}$ se traduce como

$$t(T) = \forall x.(\bigwedge_{1 \leq i \leq n} t_x(C_i) \to t_x(D_i))$$

Una A-Box $A = \{a{:}C_i \mid i \leq n\} \cup \{(a, b){:}R_i \mid i \leq m\}$ se traduce como

$$t(A) = \bigwedge_{1 \leq i \leq n} t_x(C_i)[x/a] \wedge \bigwedge_{1 \leq i \leq m} R_i(a, b)$$

T: C es satisfacible sii $t_x(C)$ es satisfacible.
T: C es satisfacible respecto de $\langle T, A\rangle$ sii $t_x(C) \wedge t(T) \wedge t(A)$ es satisfacible.
T: C es subsumido por D sii $\forall x.(t_x(C) \to t_x(D))$ es válido.
T: C es subsumido por D respecto de $\langle T, A\rangle$ sii ...
$(t(T) \wedge t(A)) \to \forall x.(t_x(C) \to t_x(D))$ es válido.

Otros Operadores / Constraints

	Number restriction ($\mathcal{N}$)
$(\leq n\ R)$	$\{d_1 \mid \|\{d_2 \mid R^{\mathcal{I}}(d_1, d_2)\}\| \leq n\}$
$(\geq n\ R)$	$\{d_1 \mid \|\{d_2 \mid R^{\mathcal{I}}(d_1, d_2)\}\| \geq n\}$
	Qualified number restrictions ($\mathcal{Q}$)
$(\leq n\ R\ C)$	$\{d_1 \mid \|\{d_2 \mid R^{\mathcal{I}}(d_1, d_2) \wedge d_2 \in C^{\mathcal{I}}\}\| \leq n\}$
$(\geq n\ R\ C)$	$\{d_1 \mid \|\{d_2 \mid R^{\mathcal{I}}(d_1, d_2) \wedge C^{\mathcal{I}}\}\| \geq n\}$
	One-Of ($\mathcal{O}$)
$\{a_1, \ldots a_n\}$	$\{a_1^{\mathcal{I}}, \ldots, a_n^{\mathcal{I}}\}$

Otros Operadores / Constraints

	Inverse roles ($\mathcal{I}$)
R^-	$\{(i, j) \mid (j, i) \in R^{\mathcal{I}}\}$
	Role Intersection ($\mathcal{R}$)
$R \sqcap S$	$\{(i, j) \mid (i, j) \in R^{\mathcal{I}} \wedge (i, j) \in S^{\mathcal{I}}\}$
	Roles transitivos ($\mathcal{R}^+$)
$R = R^+$	$R^{\mathcal{I}}$ es una relación transitiva
	Roles funcionales ($\mathcal{F}$)
R feature	$R^{\mathcal{I}}$ es una función (función parcial)
	Jerarquía de Roles ($\mathcal{H}$)
$R \sqsubseteq S$	$R^{\mathcal{I}} \subseteq S^{\mathcal{I}}$

DLs y Lógicas Modales

- DLs y las lógicas modales (ML) son formalismos muy próximos.
- Podríamos decir que DLs son 'el lado computacional de ML' y que ML son 'el lado teórico' de DLs.

$$\begin{array}{rcl} \mathcal{ALC} & \Longleftrightarrow & \mathbf{K}_m \text{ (}\mathbf{K}\text{ multimodal)} \\ \neg C & \Longleftrightarrow & \neg C \\ C \sqcap D & \Longleftrightarrow & C \wedge D \\ \exists R.C & \Longleftrightarrow & \langle R\rangle C \end{array}$$

roles transitivos	$\Longleftrightarrow$	frame transitivos (**K4**)
expr. regulares sobre roles	$\Longleftrightarrow$	propositional dynamic logic (PDL)
roles inversos	$\Longleftrightarrow$	lógicas temporales ($\mathbf{K}_t$)
number restrictions	$\Longleftrightarrow$	graded modalities ($\Diamond_n \varphi$)

Complejidad Vía Traducción

Sabiendo que muchas DLs son fragmentos de LPO (vía la traducción t), podemos transferir resultados conocidos sobre complejidad de estos fragmentos a ciertos lenguajes de descripción.

- $\mathcal{ALC}$ es un fragmento de LPO con sólo dos variables (LPO^2) que se sabe decidible.
- Más aun, $\mathcal{ALC}$ con roles inversos y operadores Booleanos sobre roles está todavía en LPO^2!
- Que pasa si agregamos $\mathcal{Q}$? Aunque nos salimos de LPO^2, caemos en C^2: LPO^2 extendida con 'counting quantifiers' ($\exists^n x.\varphi(x)$ es 'existen n indivíduos diferentes en el dominio que satisfacen φ'), también decidible

Lower Bounds vs. Upper Bounds

- Usamos la traducción t para obtener Upper Bounds (i.e., el problema P no es más complejo que el problema Q.)
- Pero no sirve para obtener Lower Bounds (i.e., el problema P es al menos tan complejo como el problema Q.). Para esto necesitaríamos traducciónes 'para el otro lado'.
- Y en realidad, muchas veces las DLs tienen mejor comportamiento computacional que los fragmentos de LPO clásicos en los que pueden traducirse naturalmente.E.g.,
  - Contrastando con muchas DLs, agregar 'roles transitivos' (requerir que ciertas relaciones binarias sean interpretadas como relaciones transitivas) a LPO^2 vuelve el fragmento indecidible.
  - LPO^2 es NExpTime-complete, mientras que la muchas DLs estan en ExpTime (o aún por debajo).

Tableaux para $\mathcal{ALC}$

- Miremos a los modelos de DL: Grafos etiquetados. Podemos pensar que un modelo de DL es
  - un conjunto de valuaciones proposicionales
  - más estructura relacional entre estas valuaciones
- Un tableau para DL entonces es
  - una colección de tableau proposicionales
  - con estructura adicional: la relacion de accesibilidad.
- Notar que esta es exactamente la información que hay en una ABox.
- Para determinar si un concepto C de $\mathcal{ALC}$ es consistente, escribir C en NNF ($(\neg\exists R.C) \rightsquigarrow (\forall R.\neg C)$ y $(\neg\forall R.C) \rightsquigarrow (\exists R.\neg C)$).
- Aplicar las siguientes reglas a $\mathcal{A} = \{a{:}\text{NNF}(C)\}$.

Reglas de Tableau para DLs

$\rightarrow_\sqcap$	Si	1. $x{:}C_1 \sqcap C_2 \in \mathcal{A}$ y
		2. $\{x{:}C_1, x{:}C_2\} \not\subseteq \mathcal{A}$
	entonces	$\mathcal{A} \rightarrow_\sqcap \mathcal{A} \cup \{x{:}C_1, x{:}C_2\}$
$\rightarrow_\sqcup$	Si	1. $x{:}C_1 \sqcup C_2 \in \mathcal{A}$ y
		2. $\{x{:}C_1, x{:}C_2\} \cap \mathcal{A} = \emptyset$
	entonces	$\mathcal{A} \rightarrow_\sqcup \mathcal{A} \cup \{x{:}D\}$ para algun $D \in \{C_1, C_2\}$
$\rightarrow_\exists$	Si	1. $x{:}\exists R.D \in \mathcal{A}$ y
		2. no hay y tq. $\{(x,y){:}R, y{:}D\} \subseteq \mathcal{A}$
	entonces	$\mathcal{A} \rightarrow_\exists \mathcal{A} \cup \{(x,y){:}R, y{:}D\}$ para y nuevo
$\rightarrow_\forall$	si	1. $x{:}\forall R.D \in \mathcal{A}$ y
		2. hay un y tq. $(x,y){:}R \in \mathcal{A}$ and $y{:}D \notin \mathcal{A}$
	entonces	$\mathcal{A} \rightarrow_\forall \mathcal{A} \cup \{y{:}D\}$

Clash: $\mathcal{A}$ tiene clash si para algun C, $\{a{:}C, a{:}\neg C\} \subseteq \mathcal{A}$.

Terminación

Por que termina este algoritmo?

- Sea $\mathcal{L}(w) = \{C \mid w{:}C \in \mathcal{A}\}$.
- Las reglas $\sqcup$, $\sqcap$, $\exists$ pueden ser aplicadas sólo una vez a una fórmula en $\mathcal{L}(w)$
- La regla $\forall$ puede ser aplicada muchas veces a una formula en $\mathcal{L}(w)$ pero sólo una vez a cada eje (w, v).
- Aplicar una regla a una fórmula φ extiende el labeling con una fórmula que es siempre estrictamente más pequeña que φ.

Correctitud y Completitud del Algoritmo

La correctitud y completitud del algoritmo se sigue de las siguientes propiedades:

1. No puede haber una sequencia infinita de aplicaciones de reglas (terminación)
2. Sea $\mathcal{A}'$ obtenido a partir de $\mathcal{A}$ por la aplicación de alguna de las reglas. Entonces $\mathcal{A}'$ es satisfacible sii $\mathcal{A}$ es satisfacible.
3. Toda Abox $\mathcal{A}$ conteniendo un clash es insatisfacible.
4. Toda Abox $\mathcal{A}$ saturada (no nueva regla puede aplicarse a $\mathcal{A}$) y sin clash es satisfacible.

Que Regla Aplicamos Primero?

- Importa que regla aplicamos primero para decidibilidad?
- Importa que regla aplicamos primero para tiempo de corrida?
- Algunas heurísticas para decidir que regla de expansión aplicar primero:
  1. Usar non-branching rules (como $\sqcap$-rule) antes que branching rules (como $\sqcup$-rule)
  2. usar reglas proposicionales (como $\sqcap$-rule y $\sqcup$-rule) antes que reglas modales (como $\exists$-rule y $\forall$-rule).

Complejidad del Algoritmo

- El algoritmo que presentamos hasta el momento puede requerir espacio (y tiempo) exponencial!
- Consideremos el concepto definido recursivamente como

$$\begin{aligned} C_1 &:= \exists R.A \sqcap \exists R.B \\ C_{n+1} &:= \exists R.A \sqcap \exists R.B \sqcap \forall R.C_n \end{aligned}$$

- El tamaño de C_n es sólo lineal en n, pero el algoritmo de tableaux construiría, al ser corrido sobre $\mathcal{A} = \{a{:}C_n\}$ una ABox conteniendo $2^{n+1} - 1$ indivíduos.

Satisfacibilidad de $\mathcal{ALC}$ esta en PSPACE

- Toda fórmula satisfacible φ puede ser satisfecha en la raíz de un árbol finito de profundidad a lo sumo $\deg(\varphi) + 1$
- El tamaño total del modelo puede ser exponencial en $|\varphi|$, pero
  - no es necesario mantener toda esta informacion en memoria.
  - en cada momento, sólo necesitamos mantener la información correspondiente a una rama del modelo.

El Algoritmo

$\mathcal{ALC}\text{-SAT}(C) := \text{sat}(x_0, \{x_0{:}C\})$

sat$(x, \mathcal{A})$:
while ($\rightarrow_\sqcap$ o $\rightarrow_\sqcup$ pueden aplicarse) y ($\mathcal{A}$ no tiene clash) do
    aplicar $\rightarrow_\sqcap$ o $\rightarrow_\sqcup$ a $\mathcal{A}$
if $\mathcal{A}$ tiene clash then return UNSAT.
$\mathbf{E}$:=$\{x{:}\exists R.D \mid x{:}\exists R.D \in \mathcal{A}\}$
while $\mathbf{E} \neq \emptyset$ do
    elegir $x{:}\exists R.D \in \mathbf{E}$ arbitrario
    $\mathcal{A}_{new} := \{(x, y){:}R, y{:}D\}$ donde y es un nuevo individuo.
    while ($\rightarrow_\forall$ puede aplicarse a $\mathcal{A} \cup \mathcal{A}_{new}$) do
        aplicar $\rightarrow_\forall$ a $\mathcal{A} \cup \mathcal{A}_{new}$
    if $\mathcal{A} \cup \mathcal{A}_{new}$ tiene clash then return UNSAT.
    if sat$(y, \mathcal{A} \cup \mathcal{A}_{new})$ = UNSAT then return UNSAT
    $\mathbf{E}$:= $\mathbf{E} \backslash \{x{:}\exists R.D\}$
    eliminar $\mathcal{A}_{new}$ de la memoria
return SAT

Terminologías

- El algoritmo que vimos hasta el momento sólo trata consistencia de conceptos
- Cómo agregamos ABoxes?
- Cómo agregamos TBoxes?
  - Consideremos una definición $C \sqsubseteq D$
  - Basta asegurarnos que cada nodo del tableaux contenga $\neg C \sqcup D$
  - Pero eso implica que el tamaño de los conceptos en un nodo (y en particuar la profundidad máxima de cuantificación) no disminuye.
  - El argumento anterior de terminación (y por lo tanto el de complejidad) se 'caen'.

Falla de Terminación: Terminologías

- Como dijimos, el algoritmo básico de tableaux no termina para $\mathcal{ALC}$ con T-boxes generales.
- E.g., si Human $\sqsubseteq$ $\exists$has-mother.Human $\in \mathcal{T}$, entonces $\neg$Human $\sqcup$ $\exists$has-mother.Human se agregaría a todo nodo del tableaux de w:Human.

w $\mathcal{L}(w) = \{\text{Human}, (\neg\text{Human} \sqcup \exists\text{has-mother.Human}), \exists\text{has-mother.Human}\}$

has-mother

x $\mathcal{L}(x) = \{\text{Human}, (\neg\text{Human} \sqcup \exists\text{has-mother.Human}), \exists\text{has-mother.Human}\}$

has-mother

y $\mathcal{L}(y) = \{\text{Human}, (\neg\text{Human} \sqcup \exists\text{has-mother.Human}), \exists\text{has-mother.Human}\}$

Blocking

- Cuando un nuevo nodo es creado, chequear los ancestros por un etiqueta idéntica (o un superconjunto).
- Si un nodo de este tipo existe, entonces el nuevo nodo esta bloqueado y ninguna regla puede aplicarse a el

w $\mathcal{L}(w) = \{\text{Human}, (\neg\text{Human} \sqcup \exists\text{has-mother.Human}), \exists\text{has-mother.Human}\}$

has-mother

x $\mathcal{L}(x) = \{\text{Human}\}$

Implementaciones Naive

Problemas típicos

- Problemas de Espacio
  - Espacio requerido para las estructuras de datos que prepresentan el tableaux.
  - Raramente un problema serio en la práctica
- Problemas de Tiempo
  - Necesitamos búsqueda dada la naturaleza no determinística del algoritmo de tableaux.
  - Un problema serio en la práctica
  - puede ser mitigado mediante
    - La elección cuidadosa del algoritmo
    - Una implementacion altamente optimizada

Tableaux para $\mathcal{I}$ y $\mathcal{N}$

- Como modificamos el tableaux para $\mathcal{ALC}$ para que funcione también para $\mathcal{I}$?
- Reglas de Tableaux para $\mathcal{N}$

$\rightarrow_\geq$ Si 1. $x{:}(\leq nR) \in \mathcal{A}$ y
2. no hay individuos $z_1, \ldots, z_n$ tal que $(x, z_i){:}R \in \mathcal{A}$ y $z_i \dot{\neq} z_j \in \mathcal{A}$ $(1 \leq i < j \leq n)$
entonces $\mathcal{A} \rightarrow_\geq \mathcal{A} \cup \{(x, y_i){:}R \mid 1 \leq i \leq n\} \cup$ $\{y_i \dot{\neq} y_j \mid 1 \leq i < j \leq n\}$ para y_i nuevos

$\rightarrow_\leq$ Si 1. $x{:}(\leq nR) \in \mathcal{A}$ y
2. $\{(x, y_i){:}R \mid 1 \leq n \leq n+1\} \subseteq \mathcal{A}$ and $y_i \dot{\neq} y_j \notin \mathcal{A}$ para algun i,j $1 \leq i < j \leq n+1$
entonces $\mathcal{A} \rightarrow_\leq \mathcal{A}[y_i/y_j]$ para algun par $y_i \dot{\neq} y_j$

Clash (para $\mathcal{N}$): $\mathcal{A}$ tiene un clash también si $\{x{:}(\leq nR)\} \cup$ $\{(x, y_i) : R \mid 1 \leq i \leq n+1\} \cup \{y_i \dot{\neq} y_j \mid 1 \leq i < j \leq n+1\} \subseteq \mathcal{A}$.

System Demo: RACER

- RACER (http://www.sts.tu-harburg.de/ r.f.moeller/racer/), desarrollado en la University of Hamburg por Haarslev y Möller en Common Lisp. Puede trabajar en $\mathcal{ALCFHIQ}(\mathcal{D}^-)_{\mathcal{R}^+}$.
- RACER es un razonador automático para DL con soporte para TBoxes, ABoxes y Concrete domains (e.g., (in)ecuaciones lineares sobre los reales)
- Es también una herramienta de inferencia para la web semántica que permite el desarrollo de ontologías, consultas de documentos RDF y ontologias RDFS/DAML que permite el registro permanente de queries con notificación automática de nuevos resultados
- Está implementado en Lisp y existen versiones para Linux, Macintosh y Windows

Lógica Computacional y Demostración Automática

Carlos Areces

areces@loria.fr

http://www.loria.fr/~areces

INRIA Nancy Grand Est, France

Diciembre 2008

El programa de hoy

- Logica de Primer Orden.
  - Que podemos expresar?
  - Sintaxis y Semantica
  - Indecibilidad
- Unificacion
  - Definicion
  - Propiedades
  - Algoritmo simple
- Resolucion para LPO
  - Forma Clausal. Skolemizacion.
  - Unificacion
  - Falla de Terminacion
  - El Algoritmo de Clausula Dada (Given Clause Algorithm)
  - Demo de SPASS

LPO: Que podemos expresar?

- Propiedades de los numeros naturales

$$\forall x.(nat(x) \rightarrow (x + 0 = x)).$$

$$\forall x.(nat(x) \rightarrow 0 * x = 0).$$

$$\forall x.\forall y.(nat(x) \wedge nat(y) \rightarrow x + succ(y) = succ(x + y)).$$

$$\forall x.\forall y.(nat(x) \wedge nat(y) \rightarrow x * y = y * x).$$

- Los axiomas de Zermelo-Fraenkel para teoria de conjuntos FO.

$$\forall x.\forall y.((x = y) \leftrightarrow (x \subseteq y \wedge y \subseteq x)$$

- Una parte importante del lenguaje natural.
- Modelos Infinitos.

Modelos Infinitos

- Proposicion: Sea φ la conjuncion de las siguientes formulas

Serialidad	$\forall x.\exists y.R(x, y)$
Transitividad	$\forall x.\forall y.\forall z.(R(x, y) \wedge R(y, z) \rightarrow R(x, z))$
Irreflexividad	$\forall x.\neg R(x, x)$.

  Entonces, $\mathcal{M} \models \varphi$ implica $\mathcal{M}$ es infinito.
- En otras palabras, existen formulas de LPO que son satisfacibles pero no tienen modelos finitos.
- La busqueda exhaustiva por un modelo (como hicimos con DP) no funciona para LPO-Sat.

Sintaxis de LPO: Elementos basicos

Un lenguaje de primer orden se define en terminos de su conjunto de variables, simbolos de constante, simbolos de predicado, y simbolos de funcion (la signatura):

- (S. de) Constantes: π, 2, Carlos, ID223, B230, LORIA, etc.
  Van a representar elementos especiales en un modelo dado.
- (S. de) Relaciones: Hermano, $\geq$, Alto, etc.
  Representan propiedades (y relaciones) entre elementos de un modelo dado.
- (S. de) Funciones: RaizCuadrada, PiernaIzquierdaDe, TamañoDe, Suc, etc.
  Representan funciones sobre elementos de un modelo lado.
- Variables: x, y, z, a, b, etc.
  Representan elementos arbitrarios en un modelo dado.

Estos simbolos se llaman tambien el lenguaje no logico y su significado tiene que ser especificado por cada modelo.

Sintaxis de LPO: El Lenguaje Logico

Ademas del lenguaje no logica, usamos tambien un lenguaje logico que tiene una interpretacion fija:

- Conectivos Booleanos: $\neg$, $\wedge$ (definibles: $\rightarrow$, $\vee$, $\leftrightarrow$)
- Quantificadores: $\exists$, (definible: $\forall$)
- Igualdad: $=$ (opcional)
- Punctuacion:), (, (. opcional)

Sintaxis de LPO: Terminos y formulas atomicas

Terminos:

una constante	E.g.: Carlos, Graciela
una variables	E.g.: x
funcion(termino$_1$,..., termino$_n$)	E.g.: PiernaIzquierdaDe(Carlos)

Los terminos pueden pensarse como 'nombres complejos' para elementos en una determinada situacion.

Formulas Atomicas:

termino$_1$ = termino$_2$

E.g. Tamaño(PiernaIzqDe(Carlos)) = Tamaño(PiernaDerDe(Carlos))

relacion(termino$_1$, ..., termino$_2$)

E.g. HermanoDe(Carlos,Graciela)

Las formulas atomicas son las unidades basicas sobre las que podemos indicar verdad o falsedad en un modelo dado.

Sintaxis de LPO: Formulas Complejas

Las formulas complejas se contruyen a partir de las formulas atomicas usando los conectivos Booleanos, los cuantificadores y los simbolos de puntuacion.

$$\neg\varphi \mid (\varphi_1 \wedge \varphi_2) \mid \exists x.(\varphi)$$

($\forall$ se define en terminos de $\neg$ y $\exists$: $\forall x.(\varphi) \equiv \neg\exists x.(\neg\varphi)$.)

Ejemplos:

HermanoDe(Carlos,Graciela) $\wedge$ Mujer(Graciela)
$\rightarrow$ HermanaDe(Graciela,Carlos)

$$\forall x.(\forall y.(>(x, y) \vee (x = y) \vee >(y, x)))$$

LPO: Semantica

- Formulas de LPO are verdaderas of falsas respecto de un modelo y una asignacion para ese modelo.
- Un modelo es un par $\langle D, I\rangle$ donde
  - D es el dominio: un conjunto no vacio de objetos
  - I es la interpretacion: una funcion asignando significado a los s. de constantes, relaciones y funcion del lenguaje .
- Una asignacion es una funcion que a cada variable asigna un elemento de D.

Formalmente, dada una signatura $S = \langle VAR, CONS, REL, FUN\rangle$ a modelo para S es $M = \langle D, I\rangle$ tal que,

- $D \neq \emptyset$
- para $c \in CONS$, $I(c) \in D$
- para $R \in REL$ n-ario, $I(R) \subseteq D^n$ (una relacion n-aria en D)
- para $F \in FUN$ n-ario, $I(F) : D^n \mapsto D$ (una funcion n-aria en D)

Una asignacion g para M es una funcion (total) $g : VAR \mapsto D$.

Satisfacibilidad

- Dado un modelo $M = \langle D, I\rangle$ y una asignacion g para M queremos definir cuando una formula de LPO φ es verdadera o falsa en M, g ($M, g \models \varphi$). Lo hacemos caso por caso.
- Definimos primero el significado de terminos complejos. La interpretacion nos da el significado de constantes ($I(c)$) y funciones ($I(F)$). La asignacion nos da el significado de las variables ($g(x)$).

$$\begin{array}{rcl} x^{I,g} & = & g(x) \\ c^{I,g} & = & I(c) \\ (f(t_1,\ldots,t_n))^{I,g} & = & I(f)(t_1^{I,g},\ldots,t_n^{I,g}) \end{array}$$

- Ahora podemos definir $M, g \models \varphi$ para formulas arbitrarias. . .

Satisfacibilidad

- $M, g \models t_1 = t_2$ sii $t_1^{I,g} = t_2^{I,g}$
- $M, g \models R(t_1, \ldots t_n)$ sii $(t_1^{I,g}, \ldots, t_n^{I,g}) \in I(R)$
- $M, g \models \neg\varphi$ sii $M, g \not\models \varphi$
- $M, g \models (\varphi_1 \wedge \varphi_2)$ sii $M, g \models \varphi_1$ y $M, g \models \varphi_2$
- $M, g \models \forall x.(\varphi)$ sii $M, g' \models \varphi$ para toda asignacion g' identica a g excepto quizas en $g(x)$.

Algunas propiedades de los cuantificadores

- $\forall x.\forall y.\varphi$ es equivalente a $\forall y.\forall x.\varphi$
- $\exists x.\exists y.\varphi$ es equivalente a $\exists y.\exists x.\varphi$
- $\exists x.\forall y.\varphi$ no es lo mismo que $\forall y.\exists x.\varphi$
- $\forall x.\varphi$ es lo mismo que $\forall y.\varphi[x/y]$ si y no aparece en φ, (lo mismo para $\exists x.\varphi$ y $\exists y.\varphi[x/y]$).
- $\varphi \wedge Qx.\psi$ es lo mismo que $Qx.(\varphi \wedge \psi)$ si x no aparece en φ ($Q \in \{\forall, \exists\}$).
- $\neg\exists x.\varphi$ es equivalente a $\forall x.\neg\varphi$ and $\neg\forall y.\varphi$ es equivalente a $\exists x.\neg\varphi$.

Logica de Primer Orden

- Por muchos años, decir "Logica" era en realidad decir "Logica de Primer Orden"
  Y habia buenos motivos
  - Alto poder expresivo.
  - Simplicidad.
  - Comportamiento excepcional tanto semantica como sintacticamente.
  - Una teoria de modelos muy bien desarrollada.
- Pero desde el punto de vista computaciona, LPO tiene una gran desventaja

LPO-SAT es indecidible.

Decidibilidad

Como probamos que un problema X es indecidible?
Ona forma es

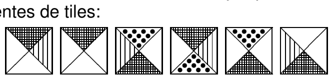
- le pedimos a alguien, mas inteligente que nosotors, que demuestre que un problema Y es indecidible
- Demostramos que si X es decidible entonces Y tambien lo seria, dando una codificacion de toda instancia del problema Y como alguna instancia del problema X.

El problema de la parada para maquinas de Turing es el ejemplo clasico de un problema indecidible. El comportamiento de una maquina de Turing, y la propiedad que dice que una dada maquina de Turing termina para todo input posible pueden ser expresados en LPO.

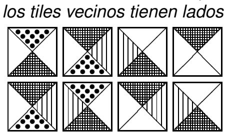
Problemas de Tiling

Un problema de tiling es una especie de rompecabezas

- Un tile T es un cuadrado de 1×1, con orientacion fija, y con un color fijo en cada uno de sus ladosPor ejemplo, aca hay seis tipos diferentes de tiles:



- Un problema de tiling facil, podria ser:
  Podemos usar tiles del tipo que acabamos de mostrar sobre una grilla de 2×4, de tal forma de cubrirla completamente, bajo la condicion de que los tiles vecinos tienen lados del mismo color?



Problemas de Tiling

- La forma general de un problema de tiling es:

  Dado un numero finito de tipos de tiles $\mathcal{T}$, podemos cubrir una parte de $\mathbb{Z} \times \mathbb{Z}$ de tal forma que los tiles adyacentes tienen el mismo color en los lados vecinos?

  (Usualmente el conjunto $\mathcal{T}$ viene especificado en terminos de relaciones binarias V y H sobre un dominio finito $\{t_1, \ldots, t_n\}$ que especifican que tiles pueden colocarse en forma adyacente horizontalmente y verticalmente)
- En algunos casos, se pueden indicar ademas condiciones que determinan que consideramos un tiling correcto.

Cubriendo $\mathbb{N} \times \mathbb{N}$

- Cubriendo $\mathbb{N} \times \mathbb{N}$
  - Tiling $\mathbb{N} \times \mathbb{N}$: dado un conjunto finito de tiles $\mathcal{T}$, puede $\mathcal{T}$ cubrir $\mathbb{N} \times \mathbb{N}$?
  - Este problema es indecidible (puede mostrarse que es equivalente al problema de la parada en maquinas de Turing. (Ver Harel, Algorithms. The Essence of Computing).
  - En el siguiente slide vamos a usar este problema para mostrar que LPO-SAT es indecidible.

Codificando tilin en LPO

- Teorema: El problema de decidir si una formula de LPO dada es satisfacible es indecidible.
  Demostracion: [Gurevich, 1976] Sean $h, v, t_1, \ldots, t_n$ simbolos de funcion unaria. Sea φ la conjuncion de las siguientes formulas:

$$h(v(x)) = v(h(x)),$$
$$\bigvee\{t_i(x) = x \mid 1 \leq i \leq n\},$$
$$\bigwedge\{t_i(x) = x \to \neg(t_j(x) = x) \mid 1 \leq i < j \leq n\},$$
$$\bigvee\{t_i(x) = x \wedge t_j(h(x)) = h(x) \mid H(t_i, t_j)\},$$
$$\bigvee\{t_i(x) = x \wedge t_j(v(x)) = v(x) \mid V(t_i, t_j)\}.$$

  Entonces $\forall x.\varphi(x)$ es satisfacible sii $\mathcal{T}$ cubre $\mathbb{N} \times \mathbb{N}$.

Demostracion automatica para LPO

- Pero LPO es indecidible???!!!! Si, Y?
- Como vimos en el caso de LP, aun problemas NP pueden tener casos imposibles de resolver en un tiempo razonable: Aun un algoritmo optimizado va a andar bien en ciertos casos y mal en otros
- Quizas entonces, un algoritmo adecuado funcione “bien” (en en caso general) para LPO. Quien sabe?

Estado del arte para demostradores de LPO

- usan algoritmos de gran complejidad y muy optimizados (generalmente basados en resolucion).
- Usan tipos de datos eficientes.
- Necesitan heuristicas.
- Ningun demostrador puede manejar igual de bien todas las formulas de LPO.

Que es unificacion?

- Objetivo: Identificar (hacer identicas) dos expresiones.
- Metodo: Reemplazar ciertas subexpresiones (variables) por otras expresiones.

Ejemplo

- Objetivo: Identififcar $f(x, a)$ y $f(b, y)$.
- Metodo: reemplazar la variable x por b, y la variable y por a. Las expresiones se transforman en $f(b, a)$.
- La substitucion $\{x \mapsto b, y \mapsto a\}$ unifica los terminos $f(x, a)$ y $f(b, y)$.
- Obviamente, debemos saber que significa ‘identificar’, que expresiones queremos manejar, y que cosas son variables y cuales no.

Que es unificacion?

- Goal: Identificar dos expresiones simbolicas.
- Metodo: Reemplazar ciertas subexpresiones (variables) por otras expresiones.

Dependiendo de que significa “identificar” (identidad sintactica, o igualdad modulo ciertas ecuaciones) podemos hablar de unificacion sintactica o de unificacion ecuacional.

Ejemplo

- Los terminos $f(x, a)$ and $g(a, x)$ no son unificables sintacticamente.
- Pero, son unificables modula la equacion $f(a, a) = g(a, a)$ con la substitucion $x \mapsto a$.

Que es unificacion?

- Objetivo: Identificar dos expresiones simbolicas.
- Metodo: Reemplazar ciertas subexpresiones (variables) por otras expresiones.

Dependiendo en que posiciones pueden aparecer las variables y que tipo de expresiones pueden reemplazarlas, hablamos de unificacion de primer orden o de unificacion de alto orden.

Example

- Si G y x son variables, los terminos $f(x, a)$ y $G(a, x)$ no puden identificarse usando unificacion de primer orden.
- $G(a, x)$ no es un termino de primer-orden: G tiene ‘parametros’.
- Pero, $f(x, a)$ y $G(a, x)$ puden ser unificadas usando unificacion de alto orden usando la substitucion $\{x \mapsto a, G \mapsto f\}$.

Para que sirve unificacion?

- Para hacer un paso de inferencia en demostracion automatica para LPO.
- Para hacer un paso de solucion en programacion logica.
- Para hacer un paso de reescritura en el area de reescritura de terminos.
- Para extraer una parte de informacion estructurada o semi-estructurada (e.g. de un documento XML).
- Para hacer inferencia de tipos en lenguajes de programacion.
- Para hacer matching en lenguajes con pattern-matching.
- Para manipulacion de esquemas de programa en ingenieria de software.
- Para varios formalismos en linguistica computacional.

Convencion y Notacion

- x, y, z son variables.
- a, b, c son constantes.
- f, g, h son funciones.
- s, t, r son terminos arbitrarios.
- Terminos cerrados (Ground terms): terminos sin variables.

Ejemplos:

- $f(x, g(x, a), y)$ es un termino, donde f es ternaria, g es binaria, a es una constante, x e y son variables.
- $f(b, g(b, a), c)$ es un termino ground.

Sustituciones

Sustitution

- Es un mapeo de variables a terminos, donde todas excepto un numero finito de variables se mapean a si mismas.

Ejemplo

Una substitucion puede representarse como una lista de bindings:

- $\{x \mapsto f(a, b), y \mapsto z\}$.
- $\{x \mapsto f(x, y), y \mapsto f(x, y)\}$.

Todas las variables excepto x e y se mapean a si mismas en estas dos substituciones.

Aplicacion de Sustituciones

Aplicamos una sustitucion σ a un termino t:

$$t\sigma = \begin{cases} \sigma(x) & \text{si } t = x \\ f(t_1\sigma, \ldots, t_n\sigma) & \text{si } t = f(t_1, \ldots, t_n) \end{cases}$$

Ejemplo

- $\sigma = \{x \mapsto f(x, y), y \mapsto g(a)\}$.
- $t = f(x, g(f(x, f(y, z))))$.
- $t\sigma = f(f(x, y), g(f(f(x, y), f(g(a), z))))$.

Una sustitucion σ es un unificador de los terminos s y t si $s\sigma = t\sigma$.

Unificador, unificador mas general

Un problema de unification: $f(x, z) \doteq^? f(y, g(a))$.

- Algunos de los unificadores:

$$\{x \mapsto y, z \mapsto g(a)\}$$
$$\{y \mapsto x, z \mapsto g(a)\}$$
$$\{x \mapsto a, y \mapsto a, z \mapsto g(a)\}$$
$$\{x \mapsto g(a), y \mapsto g(a), z \mapsto g(a)\}$$
$$\{x \mapsto f(x, y), y \mapsto f(x, y), z \mapsto g(a)\}$$
$$\ldots$$

Most General Unifiers (mgu):

$$\{x \mapsto y, z \mapsto g(a)\}, \{y \mapsto x, z \mapsto g(a)\}.$$

- mgu es unico modulo renombre de variables.

Algoritmo de Unificacion

Objetivo: Diseñar un algoritmo que para un problema de unificacion dado $s \doteq^? t$

- termine retornando un mgu de s y t si son unificables,
- termine con un mensaje de "NO UNIFICABLES" si no.

Algoritmo: Escribimos los dos terminos uno abajo del otro, y ponemos marcadores al comienzo de los dos terminos:

1. Movemos los marcadores simultaneamente, de a un simbolo, hasta que ambos estan al final de los terminos (exito), o hasta que apuntan a un simbolo diferente;
2. Si ambos simbolos son no-variables, terminamos reportando "no unificables"; si no, uno es una variable (x), el otro es el comienzo de un subtermino (t):
   - 2.1 Si x aparece en t, terminamos con "no unificables";
   - 2.2 Si no, reemplazamos todas las ocurrencias de x por t, imprimimos "$x \mapsto t$" como solucion parcial. Volvemos a 1.

Algoritmo de Unificacion

- Encuentra diferencias entre los dos terminos que queremos unificar.
- Intenta reparar la diferencia, linqueando variables a terminos.
- Falla cuando existe un clash de simbolos de funcion, o cuando intentamos unificar una variable con un termino conteniendo esa variable.

Ejemplo

$f(x, g(a), g(z))$

$f(g(y), g(y), g(g(x)))$

Ejercicios

- Dar la demostracion de que Serialidad + Transitividad + Irreflexividad $\Rightarrow$ Modelo Infinito.
- Decir si los siguientes pares de terminos son unificables o no (letras mayusculas representan variables):
  1. $a(X, c(d, X)) =^? a(2, c(d, Y))$
  2. $a(X, Y) =^? a(b(c, Y), Z)$
  3. $p(X, g(X), Y) =^? p(a, g(X), b)$
  4. $p(a, X) =^? p(X, g(X))$
  5. $p(X) =^? p(g(X))$

Resolucion en logica proposicional

- Forma Clausal. Escribir φ en forma normal conjuntiva

$$\varphi = \bigwedge_{l \in L} \bigvee_{m \in M} \psi_{(l,m)},$$

y sea el conjunto de clausulas asociadas a φ

$$ClSet(\varphi) = \{\{\psi_{(l,m)} \mid m \in M\} \mid l \in L\}.$$

- Sea $ClSet^*(\varphi)$ el minimo conjunto incluyendo $ClSet(\varphi)$ y cerrado bajo la regla (RES):

$$\frac{Cl_1 \cup \{N\} \in ClSet^*(\varphi) \qquad Cl_2 \cup \{\neg N\} \in ClSet^*(\varphi)}{Cl_1 \cup Cl_2 \in ClSet^*(\varphi)}$$

Podemos hacer "lo mismo" para LPO?

Tenemos dos problemas

- Que hacemos con los cuantificadores?

  Los eliminamos $\Rightarrow$ Skolemizacion

- La regla (RES) es demasiado debil

  $\{\{\forall x.P(x)\}, \{\neg P(a)\}\}$ es inconsistente
  pero $\{\}$ no pude derivarse mediante (RES)
  $\Rightarrow$
  Unificacion

Algunas propiedades de los cuantificadores

- $\forall x.\forall y.\varphi$ es equivalente a $\forall y.\forall x.\varphi$
- $\exists x.\exists y.\varphi$ es equivalente a $\exists y.\exists x.\varphi$
- $\exists x.\forall y.\varphi$ no es equivalente a $\forall y.\exists x.\varphi$
- $\forall x.\varphi$ es equivalente a $\forall y.\varphi[x/y]$ si y no aparece en φ, (lo mismo para $\exists x.\varphi$ y $\exists y.\varphi[x/y]$).
- $\varphi \wedge Qx.\psi$ es equivalente a $Qx.(\varphi \wedge \psi)$ si x no aparece en φ ($Q \in \{\forall, \exists\}$).
- $\neg\exists x.\varphi$ es equivalente a $\forall x.\neg\varphi$ y $\neg\forall y.\varphi$ es equivalente a $\exists x.\neg\varphi$.

Forma Clausal y Skolemnizacion

- Escribimos φ en forma nomral prenexa, con la matriz en forma normal conjuntiva:
  $\varphi = \mathcal{Q}.\psi$ donde $\psi = \bigwedge_{l\in L} \bigvee_{m\in M} \psi_{(l,m)}$
- Sea $Sko(\varphi)$ la "skolemnizacion" de φ, obtenida como
  - Mientras hay un cuantificador existencia en $\mathcal{Q}$, sea $\bar{x}$ la lista de variables universalmente cuantificadas en $\mathcal{Q}$ que aparecen en frente del primer cuantificador existencial $\exists x_i$.
  - Eliminamos $\exists x_i$ de $\mathcal{Q}$ y reemplazamos ψ por $\psi[f(\bar{x})/x_i]$ donde f es una nueva funcion $|\bar{x}|$-aria no usada hasta el momento.
- Luego de eliminar todos los cuantificadores existenciales, eliminamos $\mathcal{Q}$, y consideramos la matriz asi obtenida como una formula proposicional en forma normal conjuntiva, y definimos *ClSet* como hicimos anteriormente.

Ejemplos de Skolemnizacion

1. $\exists x.\forall y.\exists z.(P(x,y) \wedge P(y,z) \rightarrow P(x,z))$
   Sk: $P(c,y) \wedge P(y,f(y)) \rightarrow P(c,fy)$
2. $\forall x.\forall y.(P(x,y) \rightarrow \exists z.(P(x,z) \rightarrow P(z,y)))$
   Sk: $P(x,y) \rightarrow (P(x,f(x,y)) \rightarrow P(f(x,y),y))$
3. $\forall x.\exists x.P(x,x)$
   Sk: $P(f(x),f(x))$
4. $\exists x.\forall x.P(x,x)$
   Sk: $P(c,x)$

Unificacion

- Unificar dos atomos φ_1 y φ_2 es encontrar una systutycuib θ de variables por terminos, tal que $(\varphi_1)\theta = (\varphi_2)\theta$.
- Para unificar los atomos φ_1 y φ_2 vamos a considerar los simbolos de predicado y el simbolo de identidad como simbolos de funcion. De forma de poder usar el algoritmo de la clase pasada. Ademas, podemos asumir que φ_1 y φ_2 usan variables distintas (veremos despues por que)
- El algoritmo de unificacion (de complejidad lineal) es una de las caracteristicas principales (y una de las causas del exito) del algoritmo de resolucion para LPO.

Unificador Mas General

- Una sustitucion θ es una funcion de VAR $\rightarrow$ TERMS (donde solo un numero finito. . .).
- Un termino t es una instancia de substitucion de t' si existe una sustitucion θ tal que $\theta(t) = t'$.
- Una unificacion u de dos terminos s y t, si existe, es un termino que es una instancia de sustitucion de s y t para alguna sustitucion θ. θ se llama unificador de s y t.
- Si toda instancia de sustitucion de s y t es tambien una instancia de u, entonces u esta llamada unification minimal, y la sustitucion θ tal que $u = \theta(t) = \theta(s)$ es el most general unifier.
- **Teorema:** Si s y t son unificables, entonces hay un unico unificador mas general (mgu).

Algoritmo de Unificacion

- Par de diferenciacion (D): dados dos terminos, identificar la posicion mas a la izquierda en la que los dos terminos se diferencian. El Par de diferenciacion es el par de terminos que comienzan en esa posicion.
  E.g., para $g(a,f(y))$ y $g(a,x)$, $D = (f(y),x)$
- Algoritmo de Unificacion: Dados los terminos t_1, t_2 con variables disjuntas:
  1. $k := 0$; $\theta_0 := \{\}$ // La funcion Identidad
  2. IF $\theta_k(t_1) = \theta_k(t_2)$ THEN θ_k is mgu
     ELSE find the disagreement pair D_k between $\theta_k(t_1)$ and $\theta_k(t_2)$
  3. IF one of the terms in D_k is a variable v and the otheris a term t such that v dos not occur in t
     THEN $\theta_{k+1} := \theta_k + \{v \leftarrow t\}$; $k := k+1$, GOTO 2
     ELSE return "Not Unifiable"

Ejemplos de Unificacion

1.	$f(x,f(x))$ con $g(x,f(x))$	Not Unifiable
2.	$f(x,f(y))$ con $f(a,y)$	Not Unifiable
3.	$f(x,f(x))$ con $f(y,y)$	Not Unifiable
4.	$f(x,f(x))$ con $f(g(y),f(g(y)))$	Unifiable

Resolucion para LPO

- Sea $ClSet^*(\varphi)$ bel menor conjunto conteniendo $ClSet(\varphi)$ y clausurado bajo las reglas (RES) y (FAC): (θ es el mgu de M y N).

$$[RES]\ \frac{Cl_1 \cup \{N\} \in ClSet^*(\varphi) \qquad Cl_2 \cup \{\neg M\} \in ClSet^*(\varphi)}{(Cl_1 \cup Cl_2)\theta \in ClSet^*(\varphi)}$$

$$[FAC]\ \frac{Cl \cup \{N, M\} \in ClSet^*(\varphi)}{(Cl \cup \{N\})\theta \in ClSet^*(\varphi)}$$

- **Teorema:** $\forall\varphi$, $ClSet^*$ (φ) es inconsistente sii $\{\} \in ClSet^*(\varphi)$.

Ejemplo

1. $\neg((\forall x(P(x) \rightarrow Q(x)) \wedge \forall x(\neg Q(x))) \rightarrow \forall x(\neg P(x)))$
2. $\neg(\neg(\forall x(\neg P(x) \vee Q(x)) \wedge \forall x(\neg Q(x))) \vee \forall x(\neg P(x)))$
3. $((\forall x(\neg P(x) \vee Q(x)) \wedge \forall x(\neg Q(x))) \wedge \exists x(P(x)))$
4. $(((\neg P(x) \vee Q(x)) \wedge \neg Q(x)) \wedge P(c))$
5. $\{\{\neg P(x), Q(x)\}, \{\neg Q(x)\}, \{P(c)\}\}$
6. $\{\{\neg P(x), Q(x)\}, \{\neg Q(x)\}, \{P(c)\}, \{\neg P(x)\}, \{Q(c)\}, \{\}\}$

Terminacion?

Consideremos la formula

$$\exists x \forall y(R(x, y) \rightarrow (P(y) \rightarrow \exists z.(R(y, z) \wedge P(z))))$$

1. $\{\neg R(c, y),\ \neg P(y),\ R(y, f(y))\}$
2. $\{\neg R(c, z),\ \neg P(z),\ P(f(z))\}$

Con un paso de resolucion obtenemos

3. $\{\neg R(c, c),\ \neg P(c),\ \neg P(f(c)),\ P(f^2(c))\}$
4. $\{\neg R(c, f(z)),\ R(f(z), f^2(z)),\ \neg R(c, z),\ \neg P(z)\}$.

Las clauses 2 y 4 resuelven para

5. $\{\neg R(c, f^2(z)),\ R(f^2(z), f^3(z)),\ \neg R(c, f(z)),\ \neg R(c, z),\ \neg P(z)\}$.

Explosion de estados

- Como vimos en el ejemplo anterior, generar nuevas clausulas es facil.
- A decir verdad, si no se tiene cuidado, es facil generar millones de clausulas inutiles.
- Notemos que cada vez que generamos una una clausula que es directamente implicada por una clausula ya presente en el conjunto de clausulas estamos desperdiciando tiempo.
- Identificar cuando esto podria pasar (y evitarlo) es una de las principales prioridades de los demostradores de LPO (algoritmos de simplification y subsumption)
- La condicion de "no redundancia" ayuda a mantener el conjunto de clausulas ayuda a mantenerlo bajo control y a alcanzar antes el punto de saturacion (o derivar la clausula vacia).

El metodo de la clausula dada

- La mayoria de los demostradores de LPO basados en resolucion implementan una versin del "Algoritmo de Clausula Dada" (given clause algorithm).
- El algoritmo se caracteriza por diferentes implementaciones de los siguientes procedimientos:
  - `select`, Decide que clausulas usaremos en el siguiente paso de inferencia
  - `infer`, aplica las reglas de inferencia
  - `simplify(set,by)`, elimina las redundancias de `set` con la nueva informacion obtenida en `infer` (e.g., usando subsumcion).

El algoritmo de la clausula dada

```
input: init: set of clasues;
var active, passive, new: set of clauses; current: clause;
active := ∅; passive := init;
while passive ≠ ∅ do
    current := select(passive);
    passive := passive – {current};
    active := active ∪ {current};
    new := infer(current, active);
    if {} ∈ new then return unsat;
    inner_simplify(new); simplify(new, active ∪ passive);
    if {} ∈ new then return unsat;
    simplify(active, new); simplify(passive, new);
    if {} ∈ (active ∪ passive) then return unsat;
    passive := passive ∪ new
od;
return sat
```