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

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?

LPO: Que podemos expresar?

- Propiedades de los numeros naturales

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

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

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

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

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.

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

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.

LPO: Que podemos expresar?

- Propiedades de los numeros naturales

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

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

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

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

- 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

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.

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.

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

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):

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.

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.

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.

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.

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

Sintaxis de LPO: El Lenguaje Logico

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

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: $\rightarrow$, $\vee$, $\leftrightarrow$)

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: $\forall$)

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)

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

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.

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

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 $\exists$: $\forall x.(\varphi) \equiv \neg\exists x.(\neg\varphi)$.)

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)

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

LPO: Semantica

- Formulas de LPO are verdaderas of falsas respecto de un modelo y una asignacion para ese modelo.

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

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 .

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.

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 $c \in CONS$, $I(c) \in D$

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)

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)

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

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.

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 ($g(x)$).

$$x^{I,g} \quad = \quad g(x)$$

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 ($g(x)$).

$$\begin{array}{lcl} x^{I,g} & = & g(x) \\ c^{I,g} & = & I(c) \end{array}$$

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 ($g(x)$).

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

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

Satisfacibilidad

- $M, g \models t_1 = t_2$ sii $t_1^{I,g} = t_2^{I,g}$

Satisfacibilidad

- $M, g \models t_1 = t_2$ sii $t_1^{I,g} = t_2^{I,g}$
- $M, g \models R(t_1, \dots t_n)$ sii $(t_1^{I,g}, \dots, t_n^{I,g}) \in I(R)$

Satisfacibilidad

- $M, g \models t_1 = t_2$ sii $t_1^{I,g} = t_2^{I,g}$
- $M, g \models R(t_1, \dots t_n)$ sii $(t_1^{I,g}, \dots, t_n^{I,g}) \in I(R)$
- $M, g \models \neg\varphi$ sii $M, g \not\models \varphi$

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 $M, g \models \varphi_2$

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

Algunas propiedades de los cuantificadores

- $\forall x.\forall y.\varphi$ es equivalente a $\forall y.\forall x.\varphi$

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 $\exists y.\exists x.\varphi$

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 $\forall y.\exists x.\varphi$

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 $\exists y.\varphi[x/y]$).

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 ($Q \in \{\forall, \exists\}$).

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

Logica de Primer Orden

- Por muchos años, decir “Logica” era en realidad decir “Logica de Primer Orden”

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.

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

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

Decidibilidad

Como probamos que un problema X es indecidible?

Decidibilidad

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

- le pedimos a alguien, mas inteligente que nosotros, que demuestre que un problema Y es indecidible

Decidibilidad

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

- le pedimos a alguien, mas inteligente que nosotros, 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.

Decidibilidad

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

- le pedimos a alguien, mas inteligente que nosotros, 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

Problemas de Tiling

Un problema de tiling es una especie de rompecabezas

Problemas de Tiling

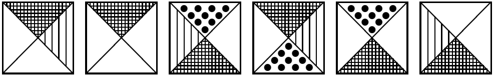
Un problema de tiling es una especie de rompecabezas

- Un tile T es un cuadrado de 1×1, con orientacion fija, y con un color fijo en cada uno de sus lados

Problemas de Tiling

Un problema de tiling es una especie de rompecabezas

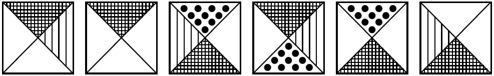
- Un tile T es un cuadrado de 1×1, con orientacion fija, y con un color fijo en cada uno de sus ladosPor ejemplo, aca hay seis tipos diferentes de tiles:



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:

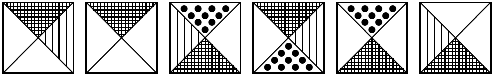


- Un problema de tiling facil, podria ser:
  Podemos usar tiles del tipo que acabamos de mostrar sobre una grilla de 2×4*, de tal forma de cubrirla completamente, bajo la condicion de que los tiles vecinos tienen lados del mismo color?*

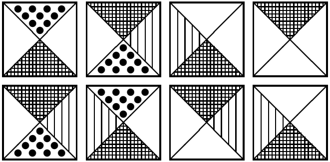
Problemas de Tiling

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

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?

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)

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 $\mathbb{N} \times \mathbb{N}$?

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).

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

Codificando tilin en LPO

- Teorema: El problema de decidir si una formula de LPO dada es satisfacible 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:

$$h(v(x)) = v(h(x)),$$

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:

$$h(v(x)) = v(h(x)),$$
$$\bigvee\{t_i(x) = x \mid 1 \leq i \leq n\},$$

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:

$$h(v(x)) = v(h(x)),$$
$$\bigvee\{t_i(x) = x \mid 1 \leq i \leq n\},$$
$$\bigwedge\{t_i(x) = x \rightarrow \neg(t_j(x) = x) \mid 1 \leq i < j \leq n\},$$

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:

$$
\begin{gathered}
h(v(x)) = v(h(x)), \\
\bigvee\{t_i(x) = x \mid 1 \leq i \leq n\}, \\
\bigwedge\{t_i(x) = x \rightarrow \neg(t_j(x) = x) \mid 1 \leq i < j \leq n\}, \\
\bigvee\{t_i(x) = x \wedge t_j(h(x)) = h(x) \mid H(t_i, t_j)\},
\end{gathered}
$$

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:

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

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:

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

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

Demostracion automatica para LPO

Demostracion automatica para LPO

- Pero LPO es indecidible???!!!!

Demostracion automatica para LPO

- Pero LPO es indecidible???!!!! Si, Y?

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

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?

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

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).

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.

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.

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?

Que es unificacion?

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

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 $f(b, y)$.

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?

Que es unificacion?

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

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.

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?

Que es unificacion?

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

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.

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.

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 que sirve unificacion?

- Para hacer un paso de inferencia en demostracion automatica para LPO.

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 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 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 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 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 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 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

Convencion y Notacion

- x, y, z son variables.
- a, b, c son constantes.
- f , g, h son funciones.
- s, t, r son terminos arbitrarios.

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.

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

Sustituciones

Sustitution

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

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

Aplicacion de Sustituciones

Aplicamos una sustitucion σ a un termino t:

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

Aplicacion de Sustituciones

Aplicamos una sustitucion σ a un termino t:

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

Ejemplo

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

Aplicacion de Sustituciones

Aplicamos una sustitucion σ a un termino t:

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

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

Unificador, unificador mas general

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

Unificador, unificador mas general

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

- Algunos de los unificadores:

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

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

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 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:

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;

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):

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";

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

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

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

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

Resolucion en logica proposicional

Resolucion en logica proposicional

- Forma Clausal. Escribir φ en forma normal conjuntiva

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

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

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

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?

Podemos hacer “lo mismo” para LPO?

Tenemos dos problemas

Podemos hacer “lo mismo” para LPO?

Tenemos dos problemas

- Que hacemos con los cuantificadores?

Podemos hacer “lo mismo” para LPO?

Tenemos dos problemas

- Que hacemos con los cuantificadores?

  Los eliminamos $\Rightarrow$ Skolemnizacion

Podemos hacer “lo mismo” para LPO?

Tenemos dos problemas

- Que hacemos con los cuantificadores?

  Los eliminamos $\Rightarrow$ Skolemnizacion

- La regla (RES) es demasiado debil

Podemos hacer “lo mismo” para LPO?

Tenemos dos problemas

- Que hacemos con los cuantificadores?

  Los eliminamos $\Rightarrow$ Skolemnizacion

- 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

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

Forma Clausal y Skolemnizacion

- Escribimos φ en forma nomral prenexa, con la matriz en forma normal conjuntiva:

$$\varphi = \mathcal{Q}.\psi \text{ donde } \psi = \bigwedge_{l \in L} \bigvee_{m \in M} \psi_{(l,m)}$$

Forma Clausal y Skolemnizacion

- Escribimos φ en forma nomral prenexa, con la matriz en forma normal conjuntiva:

$$\varphi = \mathcal{Q}.\psi \text{ donde } \psi = \bigwedge_{l \in L} \bigvee_{m \in M} \psi_{(l,m)}$$

- Sea *Sko*(φ) la "skolemnizacion" de φ, obtenida como

Forma Clausal y Skolemnizacion

- Escribimos φ en forma nomral prenexa, con la matriz en forma normal conjuntiva:

$$\varphi = \mathcal{Q}.\psi \text{ donde } \psi = \bigwedge_{l \in L} \bigvee_{m \in M} \psi_{(l,m)}$$

- Sea *Sko*(φ) 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 $\exists x_i$.

Forma Clausal y Skolemnizacion

- Escribimos φ en forma nomral prenexa, con la matriz en forma normal conjuntiva:
  $$\varphi = \mathcal{Q}.\psi \text{ donde } \psi = \bigwedge_{l\in L} \bigvee_{m\in M} \psi_{(l,m)}$$
- Sea *Sko*(φ) 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.

Forma Clausal y Skolemnizacion

- Escribimos φ en forma nomral prenexa, con la matriz en forma normal conjuntiva:
  $$\varphi = \mathcal{Q}.\psi \text{ donde } \psi = \bigwedge_{l \in L} \bigvee_{m \in M} \psi_{(l,m)}$$
- 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

Ejemplos de Skolemnizacion

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

Ejemplos de Skolemnizacion

1. $\exists x.\forall y.\exists z.(P(x,y) \land P(y,z) \rightarrow P(x,z))$
   Sk: $P(c,y) \land P(y,f(y)) \rightarrow P(c,fy)$

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

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: $P(x, y) \rightarrow (P(x, f(x, y)) \rightarrow P(f(x, y), y))$

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. $\forall x.\exists x.P(x,x)$

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: $P(f(x),f(x))$

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. $\exists x.\forall x.P(x,x)$

Ejemplos de Skolemnizacion

1. $\exists x.\forall y.\exists z.(P(x,y) \wedge P(y,z) \to P(x,z))$
   Sk: $P(c,y) \wedge P(y,f(y)) \to P(c,fy)$

2. $\forall x.\forall y.(P(x,y) \to \exists z.(P(x,z) \to P(z,y)))$
   Sk: $P(x,y) \to (P(x,f(x,y)) \to P(f(x,y),y))$

3. $\forall x.\exists x.P(x,x)$
   Sk: $P(f(x),f(x))$

4. $\exists x.\forall x.P(x,x)$
   Sk: $P(x,x)$

Unificacion

Unificacion

- Unificar dos atomos φ_1 y φ_2 es encontrar una sustitucion θ de variables por terminos, tal que $(\varphi_1)\theta = (\varphi_2)\theta$

Unificacion

- Unificar dos atomos φ_1 y φ_2 es encontrar una sustitucion 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 que vimos antes. 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.

Resolucion para LPO

Resolucion para LPO

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

Resolucion para LPO

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

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

Resolucion para LPO

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

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

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

Resolucion para LPO

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

$$[\mathit{RES}]\ \frac{\mathit{Cl}_1 \cup \{N\} \in \mathit{ClSet}^*(\varphi) \qquad \mathit{Cl}_2 \cup \{\neg M\} \in \mathit{ClSet}^*(\varphi)}{(\mathit{Cl}_1 \cup \mathit{Cl}_2)\theta \in \mathit{ClSet}^*(\varphi)}$$

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

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

Ejemplo

Ejemplo

1. $\neg((\forall x(P(x) \rightarrow Q(x)) \wedge \forall x(\neg Q(x))) \rightarrow \forall x(\neg P(x)))$

Ejemplo

1. $\neg((\forall x(P(x) \rightarrow Q(x)) \wedge \forall x(\neg Q(x))) \rightarrow \forall x(\neg P(x)))$
2. $\neg(\neg(\forall x(\neg P(x) \vee Q(x)) \wedge \forall x(\neg Q(x))) \vee \forall x(\neg P(x)))$

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. $((\forall x(\neg P(x) \vee Q(x)) \wedge \forall x(\neg Q(x))) \wedge \exists x(P(x)))$

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. $(((\neg P(x) \vee Q(x)) \wedge \neg Q(x)) \wedge P(c))$

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. $\{\{\neg P(x), Q(x)\}, \{\neg Q(x)\}, \{P(c)\}\}$

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?

Terminacion?

Consideremos la formula

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

Terminacion?

Consideremos la formula

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

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

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

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

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

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

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

Explosion de estados

- Como vimos en el ejemplo anterior, generar nuevas clausulas es facil.

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.

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 clausula que es directamente implicada por una clausula ya presente en el conjunto de clausulas estamos desperdiciando tiempo.

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 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)

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 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

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 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:

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

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

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

El algoritmo de la clausula dada

input: init: set of clasues;

El algoritmo de la clausula dada

input: init: set of clasues;
var active, passive, new: set of clauses; current: clause;

El algoritmo de la clausula dada

input: init: set of clasues;
var active, passive, new: set of clauses; current: clause;
active := $\emptyset$; passive := init;

El algoritmo de la clausula dada

input: init: set of clasues;
var active, passive, new: set of clauses; current: clause;
active := $\emptyset$; passive := init;
while passive $\neq \emptyset$ **do**

El algoritmo de la clausula dada

input: init: set of clasues;
var active, passive, new: set of clauses; current: clause;
active := $\emptyset$; passive := init;
while passive $\neq \emptyset$ **do**
    current := select(passive);

El algoritmo de la clausula dada

input: init: set of clasues;
var active, passive, new: set of clauses; current: clause;
active := $\emptyset$; passive := init;
while passive $\neq \emptyset$ **do**
    current := select(passive);
    passive := passive – {current};

El algoritmo de la clausula dada

input: init: set of clasues;
var active, passive, new: set of clauses; current: clause;
active := $\emptyset$; passive := init;
while passive $\neq \emptyset$ **do**
    current := select(passive);
    passive := passive – {current};
    active := active $\cup$ {current};

El algoritmo de la clausula dada

input: init: set of clasues;
var active, passive, new: set of clauses; current: clause;
active := $\emptyset$; passive := init;
while passive $\neq \emptyset$ **do**
  current := select(passive);
  passive := passive – {current};
  active := active $\cup$ {current};
  new := infer(current, active);

El algoritmo de la clausula dada

input: init: set of clasues;
var active, passive, new: set of clauses; current: clause;
active := $\emptyset$; passive := init;
while passive $\neq \emptyset$ **do**
    current := select(passive);
    passive := passive – {current};
    active := active $\cup$ {current};
    new := infer(current, active);
    if {} $\in$ new **then return** unsat;

El algoritmo de la clausula dada

input: init: set of clasues;
var active, passive, new: set of clauses; current: clause;
active := $\emptyset$; passive := init;
while passive $\neq \emptyset$ **do**
    current := select(passive);
    passive := passive – {current};
    active := active $\cup$ {current};
    new := infer(current, active);
    if {} $\in$ new **then return** unsat;
    inner_simplify(new); simplify(new, active $\cup$ passive);

El algoritmo de la clausula dada

input: init: set of clasues;
var active, passive, new: set of clauses; current: clause;
active := $\emptyset$; passive := init;
while passive $\neq \emptyset$ **do**
  current := select(passive);
  passive := passive – {current};
  active := active $\cup$ {current};
  new := infer(current, active);
  if {} $\in$ new **then return** unsat;
  inner_simplify(new); simplify(new, active $\cup$ passive);
  if {} $\in$ new **then return** unsat;

El algoritmo de la clausula dada

input: init: set of clasues;
var active, passive, new: set of clauses; current: clause;
active := $\emptyset$; passive := init;
while passive $\neq \emptyset$ **do**
    current := select(passive);
    passive := passive – {current};
    active := active $\cup$ {current};
    new := infer(current, active);
    if {} $\in$ new **then return** unsat;
    inner_simplify(new); simplify(new, active $\cup$ passive);
    if {} $\in$ new **then return** unsat;
    simplify(active, new); simplify(passive, new);

El algoritmo de la clausula dada

input: init: set of clasues;
var active, passive, new: set of clauses; current: clause;
active := $\emptyset$; passive := init;
while passive $\neq \emptyset$ **do**
  current := select(passive);
  passive := passive – {current};
  active := active $\cup$ {current};
  new := infer(current, active);
  if {} $\in$ new **then return** unsat;
  inner_simplify(new); simplify(new, active $\cup$ passive);
  if {} $\in$ new **then return** unsat;
  simplify(active, new); simplify(passive, new);
  if {} $\in$ (active $\cup$ passive) **then return** unsat;

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;
```

El algoritmo de la clausula dada

input: init: set of clasues;
var active, passive, new: set of clauses; current: clause;
active := $\emptyset$; passive := init;
while passive $\neq \emptyset$ **do**
  current := select(passive);
  passive := passive – {current};
  active := active $\cup$ {current};
  new := infer(current, active);
  if {} $\in$ new **then return** unsat;
  inner_simplify(new); simplify(new, active $\cup$ passive);
  if {} $\in$ new **then return** unsat;
  simplify(active, new); simplify(passive, new);
  if {} $\in$ (active $\cup$ passive) **then return** unsat;
  passive := passive $\cup$ new
od;
return sat

Spass

Spass

- Un demostrador basado en resolucion.
- Desarrollado por el Max-Planck-Institut fur Informatik, Saarbruecken, Alemania.
- `http://www.spass-prover.org/`
- Proveen sources y binarios para 2000/XP, linux, y la version MAC hasta tiene GUI!!!
- Estan muy orgullosos de su logo