UN ALGORITMO EN SEUDOCODIGO PARA EL CHEQUEO DE LA SUBSUMICION EN ALC

Eduard Gilberto Puerto C.
Estudiante X Semestre Ing. Sistemas
eduardpuerto@yahoo.com
Universidad Francisco de Paula Santander

RESUMEN

En el presente artículo se describe un evaluador de satisfactibilidad para el chequeo de la subsumición en un lenguaje de atributos de conceptos (Attribute Language Concept, *ALC*). Los lenguajes de conceptos basados en las lógicas descriptivas (Description Logics, DLs) ofrecen servicios de razonamiento que permiten hacer clasificación y recuperación de la información dentro de la base de conocimiento. Los procesos de razonamiento de subsumición y de satisfactibilidad son equivalentes y se especifican por medio del Cálculo de Predicados de Primer Orden (First Order Predicate Calculus, FOPC) y el cálculo Tableaux. FOPC permite asociar cada expresión C de conceptos a una fórmula $f_C(x)$ de la lógica de predicados, de tal forma que un modelo de una fórmula $f_C(x)$ es un modelo del concepto C y viceversa. El cálculo Tableaux de primer orden siempre termina para las fórmulas asociadas a conceptos en el FOPC. El cálculo de terminación planteado permite una interpretación si la fórmula es satisfactible o se produce una contradicción si la fórmula es insatisfactible. Se plantea un algoritmo en seudocódigo para el chequeo de la subsumición.

1. INTRODUCCION

Un área de estudio de la Inteligencia Artificial de gran actualidad por su aplicación es la sistematización del conocimiento. Para realizar esta tarea se requieren métodos que permitan representar el conocimiento junto con procedimientos efectivos y eficientes para recuperar e inferir conocimiento. Esta tarea se realiza por medio de los sistemas de representación de conocimiento (Knowledge Representation Systems, KRSs) mediante el uso de los lenguajes terminológicos o lenguajes de conceptos basados en DLs. Uno de los mecanismos básicos de la representación y procesamiento del conocimiento humano es la división del mundo en clases o conceptos los cuales usualmente están dotados de una jerarquía estructural.

Los lenguajes de conceptos son un medio para expresar conocimiento taxonómico, se conciben como sublenguajes de la lógica de predicados de primer orden con la semántica declarativa de Tarski, y permiten representar conocimiento de un domino concreto por medio de conceptos y roles, donde los conceptos son descripciones de clases de individuos y los roles modelan relaciones entre las clases (Buchheit, et al, 1993). Partiendo de conceptos atómicos y roles atómicos (que sólo son descritos por su nombre) al combinarlos se pueden construir expresiones de conceptos utilizando constructores que corresponden a operadores lógicos y a cuantificadores. Los diferentes lenguajes de conceptos son distinguibles por los constructores que ellos proveen. Los sistemas de representación de conocimiento ofrecen servicios de razonamiento que permiten inferir, ordenar, gestionar, modificar o recuperar información dentro de las bases de conocimiento y obtener respuesta a cuestiones como:

- Dado un concepto, ¿es éste satisfactible?
- Determinar la relación de subsumición entre conceptos.
- Dentro de la jerarquía de conceptos, ¿dónde se ubica un concepto determinado?
- ¿Qué hechos se deducen del conocimiento dado?
- ¿Es consistente el conocimiento?
- ¿Qué individuos son instancias de un concepto dado?
- ¿De qué conceptos un individuo es instancia?

El presente artículo está organizado así: En la sección 2 se presenta la sintaxis y la semántica de *ALC* y una relación de equivalencia entre la subsumición y la satisfactibilidad. En la sección 3 se plantea un cálculo para el chequeo de la satisfactibilidad. En la sección 4 se da una descripción de los objetos que se requieren para modelar la especificación referente al problema de satisfactibilidad. En la sección 5 se plantea un algoritmo que da solución al problema de la satisfactibilidad de una expresión de conceptos.

2 ALC

Una característica fundamental de las DLs es que ellas soportan la composición de descripciones estructuradas de conceptos con las cuales se puede razonar (Woods y Schmolze, 1992). La inferencia es posible con base en la especificación formal de la sintaxis y la semántica del lenguaje de descripción de conceptos. A continuación se describe la sintaxis y la semántica de $\mathcal{ALC}$ (Schmidt y Smolka, 1991).

La terminología $\mathcal{ALC}$ está basada en tres alfabetos de símbolos llamados nombres de conceptos, nombres de roles y nombres de individuos. Una terminología o base de conocimiento terminológico consta de un número finito de axiomas los cuales permiten introducir conceptos por medio de definiciones o de inclusiones. La terminología $\mathcal{ALC}$ está determinada por las siguientes reglas de formación:

- Axiomas de la forma $CN = C \mid CN \sqsubseteq C$, y
- Las expresiones de conceptos de la forma: $CN \mid C \sqcap D \mid C \sqcup D \mid \neg C \mid \forall P.C \mid \exists P.C$ donde CN es un nombre de concepto, C y D son expresiones de conceptos y P es un nombre de rol.

Los conceptos son interpretados como subconjuntos de un dominio y los roles son interpretados como relaciones binarias sobre un dominio (Lizcano y Ojeda, 2002). Una interpretación$I = (\Delta^I, \cdot^I)$ consta de un conjunto no vacío Δ^I llamado el dominio de I y una función^I (la función interpretación de I) que asigna cada concepto a un subconjunto de Δ^I, cada rol a un subconjunto de $\Delta^I \times \Delta^I$, y además se satisfacen las ecuaciones de la tabla 1.

$(C \sqcap D)^I = C^I \cap D^I$
$(C \sqcup D)^I = C^I \cup D^I$
$(\neg C)^I = \Delta^I - C^I$
$(\exists P.C)^I = \{d \in \Delta^I \mid \exists e \in \Delta^I,$ con $(d, e) \in P^I$ y $e \in C^I\}$
$(\forall P.C)^I = \{d \in \Delta^I \mid \forall e \in \Delta^I,$ si $(d, e) \in P^I$ entonces $e \in C^I\}$

Tabla 1. Interpretación de conceptos

La interpretación de una expresión de un concepto es derivada de las interpretaciones de sus componentes. En cada interpretación I, diferentes individuos denotan diferentes elementos del dominio Δ^I, es decir, para cada par de individuos a y b, si a es diferente de b entonces $a^I \neq b^I$. Esta propiedad corresponde al postulado de nombre único.

Una interpretación I es un modelo para un concepto C si C^I es no vacío. Un concepto es satisfactible si tiene un modelo, en otro caso se dice que no es satisfactible. Dados los conceptos C y D, se dice que C es subsumido por D (lo notamos $C \sqsubseteq D$) si $C^I \subseteq D^I$ para cada interpretación I. Finalmente se dice que C es equivalente a D si $C^I = D^I$ para cada interpretación I.

La siguiente proposición describe la equivalencia entre la subsumición y la satisfactibilidad (Nutt,1993).

Proposición 2.1 Sean C y D conceptos entonces:

1. C es subsumido por D, si y sólo si, $C \sqcap \neg D$ no es satisfactible.

2. C es satisfactible, si y sólo si, C no es subsumido por $\bot$.

Luego la satisfactibilidad y la subsumición pueden ser reducidas en un tiempo lineal la una en la otra. Por tanto los dos problemas son de igual complejidad y basta con decidir sobre uno de ellos.

3 UN CALCULO PARA EL CHEQUEO DE LA SATISFACTIBILIDAD

Con base en lo anterior, la evaluación de la subsumición entre conceptos puede ser tratada como el chequeo de satisfactibilidad de conceptos. Esto último está influenciado por el cálculo de terminación o el cálculo Tableaux para FOPC (Mac Gregor,1999).

Figura 1. Arquitectura del cálculo de satisfactibilidad de una expresión de conceptos

La técnica del cálculo básicamente consiste en aplicar el cálculo Tableaux con alguna estrategia de control a las fórmulas obtenidas de los conceptos, y está descrita como reglas que operan sobre aseveraciones correspondientes a las fórmulas de la lógica de predicados asociados a conceptos. El control es incorporado dentro de las condiciones que permiten aplicar las reglas.

El cálculo Tableaux es un cálculo basado en las reglas de expansión de los constructores del lenguaje para probar la insatisfactibilidad de fórmulas del FOPC (Lizcano,2002). De una fórmula f, se dice que es insatisfactible si al descomponer paso a paso f con las reglas de expansión se tiene una contradicción en todos los caminos, en otro caso es satisfactible, es decir, hay un conjunto terminal que es un modelo.

La estructura de datos fundamental para el cálculo son las aseveraciones. Las reglas se aplican a conjuntos de aseveraciones que son entendidos como conjunciones de sus elementos. Para determinar la satisfactibilidad de un concepto C, se parte de un conjunto de aseveraciones S, cada derivación termina después de un número finito de pasos (Smullyan,1968).

Si todos los conjuntos terminales de aseveraciones contienen una contradicción, entonces C no es satisfactible. En otro caso se concluye que C es satisfactible ya que existe al menos un conjunto terminal libre de contradicción , el cual describe un modelo de C. El cálculo asume que las expresiones de conceptos están en forma normal negativa, es decir, que las negaciones sólo se aplican a nombres de conceptos y no a términos compuestos.

En el FOPC se expresan los conceptos atómicos como predicados unarios y los roles como predicados binarios. Esta identificación puede ser extendida asociando cada expresión C de conceptos a una fórmula $f_c(x)$ de la lógica de predicados, con x variable libre. Un modelo de una fórmula $f_c(x)$ es un modelo del concepto C y viceversa. En particular, C no es satisfactible, si y sólo si, $f_c(x)$ no es satisfactible.

El cálculo Tableaux de primer orden siempre termina para las fórmulas asociadas a conceptos en el FOPC y genera un modelo si la fórmula es satisfactible o una contradicción si la fórmula es insatisfactible. Con base a lo anterior se puede diseñar un chequeador de satisfactibilidad que consta de dos componentes: un evaluador de refutación de teoremas y un procedimiento que genera todas las interpretaciones finitas y evalúa cuales de ellas son un modelo (Lizcano, 2001). Si ambos procesos comienzan con la entrada $f_c(x)$ y se ejecutan en paralelo el probador de teoremas encontrará que la fórmula es insatisfactible, si ésta lo es, y el evaluador de interpretación exhibirá un modelo de la fórmula, si ésta lo tiene. El cálculo Tableaux combina las características de ambos procesos.

El cálculo para el chequeo de la satisfactibilidad parte de un sistema de aseveraciones $S = \{x : C\}$ y se desarrolla en sucesivos pasos aplicando un conjunto de reglas de expansión que corresponden a los constructores del lenguaje. El proceso termina cuando ninguna de las reglas se puede aplicar o se presente una contradicción. El cálculo se basa en un conjunto de reglas que son equivalentes a las reglas del cálculo Tableaux (Horrocks,1997).

regla-$\sqcap$: Si 1. $x : C_1 \sqcap C_2 \in S$
2. $\{ x : C_1, x : C_2 \} \notin S$
Entonces $S \rightarrow S \cup \{ x : C_1 , x : C_2 \}$

regla-⊔: Si 1. $x : C_1 \sqcup C_2 \in S$
2. $\{ x : C_1 , x : C_2 \} \cap S = \Phi$
Entonces a. Guardar S
b. Hacer $S \rightarrow S \cup \{ x : C_1 \}$ si se llega a una contradicción restaurar S, y
c. Hacer $S \rightarrow S \cup \{ x : C_2 \}$

regla-$: Si 1. $x : \exists P.C \in S$
2. Si no existe variable alguna y tal que $x P y$ en S y además $y : C \in S$.
Entonces crear una variable y, luego hacer $S \rightarrow S \cup \{ x P y, y : C \}$

regla-∀: Si 1. $x : \forall P.C \in S$
2. Si existe alguna variable y tal que $x P y$ en S y
además $y : C \in S$.
Entonces hacer $S \rightarrow S \cup \{ y : C \}$

Figura 2. Reglas de expansión del Tableaux para $\mathcal{ALC}$.

Una contradicción para una variable x y un concepto C se tiene, cuando uno de estos dos hechos se da en el sistema de aseveraciones:

- $\{ x : C, x : \neg C \}$.

4 MODELADO DE OBJETOS PARA EL CALCULO DE LA SATISFACTIBILIDAD DE UNA EXPRESION DE CONCEPTOS

En un análisis del proceso de chequeo para la subsumición se ha hecho la abstracción de los siguientes tres objetos que permiten modelar el comportamiento del sistema. ALMACEN, que es la agregación de INDIVIDUO y EXPRESION, y por la cual se determina una taxonomía en una base de conocimiento.

A continuación se da una descripción de estos objetos.

ALMACEN. Es la clase que contiene los datos alusivos a la expresión de conceptos y otros que resultan de aplicar el algoritmo de Tableaux para evaluar la satisfactibilidad de una expresión de conceptos.

Atributos:
Cod_Exp: Establece el código de la expresión que es un valor consecutivo.

Expresion: Expresión de conceptos generados al aplicar las reglas de expansión.

Tipo_Const: Constructor lógico que caracteriza la expresión de conceptos en cuestión.

Cod_Ind: Determina el código del individuo en que se instancia la expresión de conceptos.

Cod_Exp_Prec: Es el código de la expresión del cual ha sido generada la expresión en cuestión.

Estado: Identifica expresiones de conceptos a las cuales se le ha aplicado una regla de expansión.

Camino: Define una trayectoria para las expresiones de conceptos que han sido derivadas.

Operaciones:

Crear( ): Crea un nuevo registro a partir de una expresión de conceptos.

Chequear( ): Verifica si la expresión de conceptos del registro en cuestión es la negación de alguna expresión de conceptos de los registros anteriores.

Expandir( ): Ejecuta la operación de expansión determinada por la operación Estrategia( ).

Estrategia( ): Determina la regla a aplicar de las expresiones de conceptos factibles a expandir.

Figura 3. Diagrama de clases

4.1 SEUDOCODIGO DE LAS OPERACIONES

CREAR( )
Entradas: Expresion, Cod_Exp_Original, Cod_Ind, Camino.
Salidas: Ninguna

Tipo = Analizar(Expresion)
Estado = DefinirEstado(Tipo)
IF(Almacen = Nil) THEN
CREAR PRIMER REGISTRO
Cod_Exp = 1
Expresion = Expresión
Tipo_Const = Tipo
Cod_Ind = 1
Cod_Exp_Prec = 0
Estado = Estado
Camino = Camino
ELSE
CREAR REGISTRO
Cod_Exp = MaxCod_Exp+1
Expresion = Expresión
Tipo_Const = Tipo
Cod_Ind = Cod_Ind
Cod_Exp_Prec = Cod_Exp_Original
Estado = Estado
Camino = Camino

FIN ELSE
FIN IF
FIN CREAR

DefinirEstado()
Entrada: Tipo
Salida: Entero

IF(Tipo = 0 | Tipo = 1) THEN
RETURN 1
ELSE
RETURN 0
FIN IF
FIN DefinirEstado

Analizar()
Entrada: Expresion
Salida: Entero

Ultimo Caracter de Exp.Infijo
CASE 0: Concepto atómico
CASE 1: Concepto atómico negado
CASE 2: AplicarRegla-⊓
CASE 3: AplicarRegla-⊔
CASE 4: AplicarRegla-$
CASE 5: AplicarRegla- ∀
FIN Analizar

ESTRATEGIA( )
Entrada: ninguna
Salida: Entero // Si es cero no hay regla a aplicar y si es mayor que cero encontró una regla a aplicar en un registro en Cod_Exp.

WHILE(EOF != Nil & Estado = 0 & Tipo_Cons =2)
RETURN Cod_Exp FIN WHILE


```
WHILE(EOF != Nil & Estado = 0 & Tipo_Cons =3)
DO
        i = i+1  // Actualizar el indice
        CEND[i] = Cod_Exp_Actual
        RETURN Cod_Exp

FIN WHILE
WHILE(EOF != Nil & Estado = 0 & Tipo_Cons =4)
RETURN Cod_Exp FIN WHILE
WHILE(EOF != Nil & Estado = 0 & Tipo_Cons =5)
RETURN Cod_Exp FIN WHILE
FIN ESTRATEGIA
```

EXPANDIR()
Entrada: Cod_Exp_Actual
Salida: Expresión o Expresiones

```
READ Expresion, Tipo_Const, Cod_Ind, Camino.
        Estado = 1  // cambiar el valor del campo
        Estado de cero a uno
        IF(Tipo_Const = 3) THEN
        Camino = Camino + .1   //  Camino es
        alfanumérico.
        AplicarRegla(Expresion, Cod_Exp.Actual,
        3,Camino)
        FIN IF
        IF(Tipo_Const = 4 | 5) THEN
        AplicarRegla(Expresion, Cod_Exp.Actual,
        Tipo_Const, Camino)
        ELSE AplicarRegla(Expresion, Cod_Exp.Actual,
        2,Camino)
        FIN IF
FIN EXPANDIR
```

AplicarRegla()
Entrada: Expresion, Cod_Exp, Entero, Camino
Salida: Ninguna

```
READ Cod_Exp & Cod_Ind

CASE 2:
N1 = Primera parte Exp.Infija
N2 = Segunda parte Exp_Infija
Crear(N1, Cod_Exp, Cod_Ind, Camino )
Crear(N2, Cod_Exp, Cod_Ind, Camino )
CASE 3:
N1 = Primera parte Exp.Infija
Crear(N1, Cod_Exp, Cod_Ind, Camino )
CASE 4:
C = Concepto de Exp.Infija
READ Reg.Cod_Ind == Mayor.Cod_Ind
Cod_Ind = Reg.Cod_Ind +1
Crear(C, Cod_Exp, Cod_Ind, Camino )
CASE 5:
C = Concepto de Exp.Infija
Crear(C, Cod_Exp, Cod_Ind, Camino )
FIN AplicarRegla
```

CHEQUEAR()
Entrada: UltimoRegistro
Salida: Contradicción o no-contradicción

```
READ  Cod_Exp  // leer el codigo de expresión del
ultimo registro
Contradicción = no  // iniciar con no contradicción
Marca = Izquierda
READ Expresión & Camino
GOTO Registro inicial de Almacen
      WHILE(EOF != NIL & Contradiccion = no &
      RegLeido.Camino != 0 ) DO
         IF(Reg.Leido.Expresion  =  ¬ Expresión)
         THEN
         Contradicción = si
         IF (Camino = par) THEN Marca = Derecha
      FIN IF
      FIN WHILE
IF(Contradiccion = si) THEN
      WHILE(EOF != NIL)  DO
         IF  RegLeido.Camino  =  Camino  THEN
         RegLeido.Camino = 0
      FIN WHILE
         IF(Marca = Derecha) THEN
         Camino = Cam.CEND[i]
      WHILE EOF != NIL DO
         IF  RegLeido.Camino  =  Camino  THEN
         RegLeido.Camino =0
```

```
        FIN WHILE
        FIN IF
FIN IF
RETURN Contradiccion.
FIN CHEQUEAR
```

INDIVIDUO Representa el objeto que se instancia en una expresión de conceptos.

Atributo:

Cod_Ind: Descrito antes.

Operaciones:

Crear(): Crea el código asociado a un individuo

EXPRESION Es la clase que se le asocia una expresión de conceptos y un constructor lógico.

Atributos:

Expresion: Descrito antes.

Tipo-Const: Descrito antes. Asignado los siguientes valores:

Para un concepto atómico = 0 y para un concepto atómico negado = 1.
Constructor $\sqcap$ = 2
Constructor $\sqcup$ = 3
Constructor $ = 4
Constructor $\forall$ = 5

Operaciones:

Crear(): Crea el valor del campo de Expresión y el Tipo_Const.

Analizar(): Determina el constructor asociado a la expresión de conceptos.

5 ALGORITMO DE SATISFACTIBILIDAD DE UNA EXPRESIÓN DE CONCEPTOS C

Sea A un ABox, el árbol n-ario G_A inducido por A, donde cada nodo es una instancia de la clase Expresión, los nodos descendientes son instancias generadas por el algoritmo ALC de la aplicación de las reglas de terminación del Cálculo de Tableaux, partiendo de una ABox inicial {xo : C}.

```
Entrada: Expresión de Conceptos
Salida: Insatisfactible | Satisfactible
Proceso:  SAT(C): Cadena
      SAT(C): = Sat (x , C )//Crear(C,1,1,2) creación
      de la tabla Almacén y el primer registro.
      i = 0              //índice para ⊔ camino  y para
      invalidarlo si presenta contradicción
WHILE(EOF != NIL) DO
      IF (Estado = 0) THEN
         Count = MaxCod_Exp +1
         Expandir(Estrategia(Cod_Exp))
      ELSE(Estado = 1) Count = MaxCod_Exp
      FIN IF
         Aux =1  // Auxiliar es una variable que mira
         si existe algun registro por analizar
      WHILE(EOF != NIL) DO
         IF Estado = 0  THEN Aux = 0
      FIN WHILE
      WHILE (Count <= MaxCod_Exp) DO
         IF (Chequear(Count) = NoContradicción &
         Aux = 1) RETURN Satisfactible
         IF (Chequear(Count) = Contradicción &
         Count.Camino = C.Impar) THEN
            Camino = CEND[i]+.2
            Crear (ExDerCEND[i], CEND[i],
            CI.CEND[i], Camino)
            i = i -1    // CEND: Código de
            expresión no determinista CI: código de
            individuo
         FIN IF
         IF (Chequear(Count) = Contradicción &
         Count.Camino = C.par) THEN
         IF (i >0) THEN
```


```
            Camino = CEND[i]+.2
            Crear(ExDerCEND[i],CEND[i],CI.CEND[i],
            Camino )    i = i -1
            FIN IF
        FIN IF
        Count = Count +1
    FIN WHILE
FIN WHILE
RETURN Insatisfactible.
```

CONCLUSIONES

En este artículo se ha presentado un cálculo para verificar la relación de subsumición de un concepto en el lenguaje $\mathcal{ALC}$, utilizando un paradigma matemático sobre la equivalencia entre la subsumición y satisfactibilidad. Se ha construido una especificación que permite desarrollar una implementación de dicho cálculo abordando el diseño de las operaciones Crear, Estrategia, Expandir y Chequear en seudocódigo del proceso SAT(C) y presentando un algoritmo que evalúa la satisfactibilidad de conceptos. Creo que estos resultados no solo tienen una importancia teórica, sino que el hecho de haber desarrollado estos algoritmos permiten pensar en la implementación de los mismos y una ampliación de este trabajo es dotar el algoritmo, de técnicas de optimización (Lizcano, 2001).

BIBLIOGRAFIA

Buchheit M., Domini F. and Schaerf A. Decidable reasoning in terminological knowledge representation systems. Journal of artificial Intelligence Research, 1 :109-138, 1993.

Horrocks I., Optimising Tableaux Decision Procedures for Description Logics. Ph. D. thesis, University of Manchester, 1997.

Lizcano L., Calculo para evaluar la Satisfactibilidad en ALC. Primera CIAMAC. Primera Conferencia Iberoamericana de Matemática Computacional., Bogotá 2001

Lizcano L. y Ojeda R., Fundamento formal de las Lógicas Descriptivas. Revista de Ingeniería de la Universidad Nacional de Colombia, 2002. En Imprenta.

Lizcano L., Razonamiento en una Expresiva Abox con Restricciones de Número. CISCI 2002 (Conferencia Iberoamericana en Sistemas, Cibernética e Informática) Orlando, Florida, EE.UU.

Mac Gregor R., Integrating Descriptions and Classification into a Predicate Calculus Framework. Collected Papers from the International Description Logics Workshop (DL'99), Linköping, 1999.

Nutt W., Algorithms for Constraints in Deduction and Knowledge Representation. PhD. thesis, University of Saarlandes, 1993.

Schmidt-Schaub M. and Smolka G., Attributive concept descriptions with complements. Artificial Intelligence, 48, 1-26, 1991.

Smullyan R. M. First Order Logic, Springer-verlag Berlin, 1968.

Woods W. and Schmolze J., The KL-ONE Family. Computers and Mathematics with Aplications, Vol 23, No. 2-5:133-177, 1992.