Una introducción al Análisis Formal de Conceptos en PVS *

José A. Alonso, J. Borrego, M.J. Hidalgo, F.J. Martín y J.L. Ruiz
www.cs.us.es/{~jalonso, ~jborrego, ~mjoseh, ~fmartin, ~jruiz}
Dpto. de Ciencias de la Computación e Inteligencia Artificial
Escuela Técnica Superior de Ingeniería Informática. Universidad de Sevilla
Avda. Reina Mercedes, s/n. 41012 Sevilla, Spain

Resumen En este trabajo, presentamos una formalización en PVS del Análisis Formal de Conceptos. Inicialmente, formalizamos la noción de concepto y establecemos que el conjunto de conceptos de un contexto formal tiene estructura de retículo completo. A continuación, realizamos una especificación en PVS de un algoritmo para obtener todos los conceptos de un contexto finito y probamos que es correcto y completo.

1 Introducción

El Análisis Formal de Conceptos (AFC) [2,7], introducido en 1982 por Rudolf Wille, es una técnica de aprendizaje capaz de extraer estructuras conceptuales de un conjunto de datos. Está basada en la idea filosófica de que un "concepto" consta de dos partes: su extensión, formada por todos los objetos que pertenecen a dicho concepto; y su intención, que comprende todos los atributos compartidos por dichos objetos.

El marco en el que se establecen los conceptos se conoce como contexto formal. Consta de un conjunto de objetos, un conjunto de atributos o propiedades, y una relación que informa sobre los atributos que posee cada objeto. El conjunto de los conceptos de un contexto formal tiene estructura de retículo completo, lo que permite representarlos gráficamente como jerarquías conceptuales, posibilitando el análisis de estructuras complejas y descubriendo dependencias entre los datos.

Esta técnica se ha aplicado con éxito en análisis de datos, recuperación de información y descubrimiento del conocimiento a partir de bases de datos (KDD) [5]. Como aplicación se ha desarrollado el programa TOSCANA [6], que permite explorar de forma interactiva bases de datos.

Por otra parte, el análisis formal de conceptos también ha sido objeto de estudio desde el campo de la deducción automática. Concretamente, el retículo de los conceptos de un contexto formal ha sido formalizado en Mizar[4]. El objetivo de nuestro trabajo es la formalización de la teoría AFC en un sistema de razonamiento más automático. Hemos elegido PVS porque combina un expresivo lenguaje de especificación con un demostrador de teoremas interactivo.

* Este trabajo ha sido financiado por el proyecto TIC2000–1368–C03–02 del Ministerio de Ciencia y Tecnología

PVS (Prototype Verification System) [1] es un sistema de razonamiento en el que se pueden realizar especificaciones formales y razonar sobre ellas. Consta de un lenguaje de especificación, algunas teorías predefinidas (contenidas en el preludio) y un demostrador de teoremas. Fue desarrollado en el SRI International Computer Science Laboratory, y ha sido ampliamente utilizado tanto en campos científicos como industriales (ver `http://pvs.csl.sri.com/users.html`).

El lenguaje de especificación de PVS está construido sobre una lógica de orden superior con tipos. Los tipos básicos incluyen tipos no interpretados, que pueden ser introducidos por el usuario, y tipos construidos, tales como enteros, reales y ordinales hasta ϵ_0, entre otros. Por otra parte, el preludio de PVS proporciona especificaciones y resultados sobre un número considerable de teorías, incluyendo teoría de conjuntos, conjuntos finitos, listas y sucesiones.

El demostrador de teoremas de PVS contiene una colección de reglas de inferencia que son aplicadas por el usuario, de forma interactiva, dentro de un marco de cálculo de secuentes. Estas reglas incluyen simplificación, inducción y procedimientos de decisión para la aritmética lineal. Además, permite al usuario combinarlas para construir estrategias de prueba de nivel superior.

En el desarrollo de este artículo, presentaremos de forma escalonada los aspectos más relevantes del AFC, así como algunas características de PVS.

2 Conceptos en contextos formales

Definición 2.1 *Un contexto formal C es una terna (O, A, I) donde O es un conjunto de objetos, A es un conjunto de atributos e $I \subseteq O \times A$. Si $(d, a) \in I$ significa que el objeto d posee el atributo a (también lo notaremos por dIa).*

Ejemplo 2.2 *Consideremos el conjunto de objetos {g: gato, s: sanguijuela, r: rana, m: maíz, p:pez} sobre los que se han observado las propiedades siguientes {N: necesita agua, A: es acuático, M: es móvil, P: tiene patas}, obteniéndose la relación dada por la siguiente tabla:*

	N*ec. agua*	**A***cuático*	**M***óvil*	**P***atas*
g*ato*	**X**		**X**	**X**
s*anguijuela*	**X**	**X**	**X**	
r*ana*	**X**	**X**	**X**	**X**
m*aíz*	**X**			
p*ez*	**X**	**X**	**X**	

Para expresar la noción de contexto formal en PVS consideramos, en primer lugar, dos tipos no interpretados T_1 y T_2 que representarán objetos y atributos, respectivamente. Establecemos un tipo formado por estructuras con tres campos: un conjunto de elementos de tipo T_1, un conjunto de elementos de tipo T_2, y un conjunto de elementos del tipo producto cartesiano $T_1 \times T_2$.

```
CFT: TYPE =  [# obj: set[T1],
                atrib: set[T2],
                relacion: set[[T1, T2]] #]
```

Ahora, podemos representar un contexto formal como una estructura del tipo CFT, $[O, A, I]$, en la que los elementos de I son de la forma (d, a) con $d \in O$ y $a \in A$. Notemos aquí que, en PVS, un conjunto `S` con elementos en un tipo `T` está definido por el predicado que caracteriza a sus elementos (`S(x)` $\equiv x \in S$).

```
contexto_formal: TYPE =
  {C: CFT | LET Re = relacion(C) IN
            FORALL (par:(Re)): obj(C)(proj_1(par)) AND
                               atrib(C)(proj_2(par))}
```

donde si `x` es una n-tupla, entonces `proj_i(x)` es la proyección i-ésima de `x`.

En PVS, un conjunto de elementos de un tipo dado que verifique una determinada condición se denomina subtipo. Los subtipos proporcionan una parte importante de la capacidad expresiva del lenguaje, que usamos aquí para establecer el tipo con el que representaremos los contextos formales.

En particular, consideraremos contextos formales finitos como contextos en los que los conjuntos de objetos y atributos son finitos y, además, el conjunto de atributos es no vacío.

```
CFTF: TYPE = [# obj: finite_set[T1],
                atrib: non_empty_finite_set[T2],
                relacion: finite_set[[T1, T2]] #]

contexto_formal_finito: TYPE =
  {C: CFTF | LET Re = relacion(C) IN
             FORALL (par:(Re)): obj(C)(proj_1(par)) AND
                                atrib(C)(proj_2(par))}
```

Dado un contexto $C = (O, A, I)$, declaramos los tipos específicos formados por los objetos y atributos de dicho contexto, así como los tipos formados por sus conjuntos de objetos y atributos:

```
C: VAR contexto_formal

objetos(C) :    TYPE = (obj(C))
atributos(C) :  TYPE = (atrib(C))
c_objetos(C):   TYPE = set[objetos(C)]
c_atributos(C): TYPE = set[atributos(C)]
```

Para definir la noción de concepto utilizamos los *operadores de derivación* en un contexto formal que definimos a continuación.

Definición 2.3 *Sea X un conjunto de objetos de un contexto $C = (O, A, I)$. La* **intención** *de X, que notamos por X', es el conjunto de atributos comunes a todos los objetos de X:*

$$X' = \{a \in atrib(C) : dIa, \text{ para todo } d \in X\}$$

Definición 2.4 *Sea Y un conjunto de atributos de un contexto $C = (O, A, I)$. La* **extensión** *de Y, que notamos por Y', es el conjunto de objetos que poseen todos los atributos de Y:*

$$Y' = \{d \in obj(C) : dIa, \text{ para todo } a \in Y\}$$

```
intencion(C)(X: c_objetos(C)): c_atributos(C) =
   {a: (atrib(C)) | FORALL (d: (X)): relacion(C)(d, a)}

extension(C)(Y: c_atributos(C)): c_objetos(C) =
   {d: (obj(C)) | FORALL (a: (Y)): relacion(C)(d, a)}
```

Los operadores de derivación en un contexto formal (O, A, I) verifican las siguientes propiedades:

Lema 2.5 *Sea $C = (O, A, I)$ un contexto formal. Dados $X, X_1, X_2 \subseteq O$, e $Y, Y_1, Y_2 \subseteq A$, se verifica:*

(1) $X_1 \subseteq X_2 \Longrightarrow X_2' \subseteq X_1'$ (1) $Y_1 \subseteq Y_2 \Longrightarrow Y_2' \subseteq Y_1'$
(2) $X \subseteq X''$ (2′) $Y \subseteq Y''$
(3) $X = X'''$ (3′) $Y = Y'''$
(4) $X \subseteq Y' \Longleftrightarrow Y \subseteq X' \Longleftrightarrow X \times Y \subseteq I$

Estas propiedades se formalizan de manera natural y se prueban de forma casi automática en PVS. En particular, la formalización de (3) se expresa de la forma siguiente:

```
int_es_int_ext_int: LEMMA
  FORALL (X: c_objetos(C)):
   intencion(C)(X) = intencion(C)(extension(C)(intencion(C)(X)))
```

En cuanto a la prueba, se realiza de forma interactiva, proporcionándo indicaciones acerca de las reglas o propiedades que se quieren aplicar en un paso determinado. Dichas propiedades pueden estar establecidas en las teorías predefinidas del sistema (`subset_antisymmetric`, de la teoría de conjuntos `sets_lemmas`), o bien en la propia teoría que estamos construyendo (`int_ext_subset` y `ext_int_subset`, que corresponden a la propiedades (2) y (2')). Además, el sistema genera de forma automática la descripción detallada de la prueba que incluimos a continuación:

Verbose proof for `int_es_int_ext_int`.

```
|
------------------------------------------------------------
{1} ∀ (C: contexto_formal), (X: c_objetos(C)):
      intencion(C)(X) =
       intencion(C)(extension(C)(intencion(C)(X)))
```

Skolemizing and flattening,
`int_es_int_ext_int:`

```
|
------------------------------------------------------------
{1} intencion(C')(X') =
      intencion(C')(extension(C')(intencion(C')(X')))
```

Rewriting using `subset_antisymmetric`, matching in *, we get 2 subgoals:
`int_es_int_ext_int.1:`

```
|
------------------------------------------------------------
{1} (intencion(C')(X') ⊆ intencion(C')(extension(C')(intencion(C')(X'))))
{2} intencion(C')(X') =
      intencion(C')(extension(C')(intencion(C')(X')))
```

Applying `ext_int_subset`
`int_es_int_ext_int.1:`

```
|{-1} ∀ (C: contexto_formal, Y: c_atributos(C)):
|       (Y ⊆ intencion(C)(extension(C)(Y)))
------------------------------------------------------------
{1} (intencion(C')(X') ⊆ intencion(C')(extension(C')(intencion(C')(X'))))
{2} intencion(C')(X') =
      intencion(C')(extension(C')(intencion(C')(X')))
```

Instantiating the top quantifier in - with the terms: C', intencion$(C')(X')$,
This completes the proof of `int_es_int_ext_int.1`.
`int_es_int_ext_int.2:`

```
|
------------------------------------------------------------
{1} (intencion(C')(extension(C')(intencion(C')(X'))) ⊆ intencion(C')(X'))
{2} intencion(C')(X') =
      intencion(C')(extension(C')(intencion(C')(X')))
```

Using lemma `int_ext_subset`,
`int_es_int_ext_int.2:`

```
|{-1} (X' ⊆ extension(C')(intencion(C')(X')))
------------------------------------------------------------
{1} (intencion(C')(extension(C')(intencion(C')(X'))) ⊆ intencion(C')(X'))
{2} intencion(C')(X') =
      intencion(C')(extension(C')(intencion(C')(X')))
```

Forward chaining on `intencion_subset`,
This completes the proof of `int_es_int_ext_int.2`.
Q.E.D.

Usando estos operadores se establece la definición de concepto en un contexto formal como sigue:

Definición 2.6 *Un* **concepto** *en un contexto formal $C = (O, A, I)$ es un par (X, Y) donde X es un conjunto de objetos de C, e Y es un conjunto de atributos de C, tales que $X' = Y$ e $Y' = X$.*

Decimos que X e Y son la extensión y la intención, respectivamente, del concepto (X, Y).

```
es_concepto_c?(C)(par: [c_objetos(C), c_atributos(C)]): bool =
 LET (X, Y) = par IN intencion(C)(X) = Y AND extension(C)(Y) = X
```

En nuestro ejemplo, $(\{g, s, r, m, p\}, \{N\})$ y $(\{s, r, p\}, \{N, A, M\})$ son conceptos, pero $(\emptyset, \{N, A, M, P\})$ no lo es, pues $\{N, A, M, P\}' = \{r\} \neq \emptyset$.

Observamos que el conjunto de conceptos de un contexto formal $C = (O, A, I)$ es no vacío, puesto que (A', A'') siempre es un concepto.

```
concepto_c_no_vacio: LEMMA
  es_concepto_c?(C)(extension(C)(atrib(C)),
                    intencion(C)(extension(C)(atrib(C))))
```

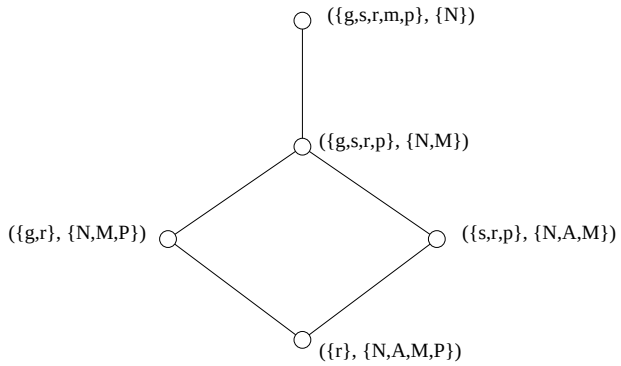
Notamos por $R(O, A, I)$ al conjunto de los conceptos de un contexto formal $C = (O, A, I)$. El tipo correspondiente en PVS es el siguiente:

```
concepto_c(C): TYPE = (es_concepto_c?(C))
```

3 El retículo de los conceptos de un contexto formal

Los conceptos de un contexto pueden ser parcialmente ordenados de forma natural: un concepto C_1 es "menor" que otro C_2 cuando todos los objetos de C_1 lo son también de C_2.

En nuestro ejemplo, podemos visualizar la relación entre los conceptos del contexto mediante el siguiente diagrama:



En esta sección definimos la relación de orden entre los conceptos de un contexto formal y probamos en PVS, que el conjunto de los conceptos $R(O, A, I)$, junto con esta relación tiene estructura de retículo completo (Teorema 3.5).

Definición 3.1 *Sean* (X_1, Y_1) *y* (X_2, Y_2) *conceptos del contexto formal* $C = (O, A, I)$. *El concepto* (X_1, Y_1) *es subconcepto de* (X_2, Y_2), *y se representa por* $(X_1, Y_1) \leq (X_2, Y_2)$, *si* $X_1 \subseteq X_2$.

```
subconcepto_c?(C)(par1, par2: concepto_c(C)): bool =
  subset?(proj_1(par1), proj_1(par2))
```

Lema 3.2 *Sean* (X_1, Y_1) *y* (X_2, Y_2) *conceptos del contexto formal* $C = (O, A, I)$. *Entonces,* $(X_1, Y_1) \leq (X_2, Y_2)$ *si y sólo si* $Y_2 \subseteq Y_1$.

```
cond_equiv_subconcepto_c: LEMMA
  FORALL (par1, par2: concepto_c(C)):
    subconcepto_c?(C)(par1, par2) IFF
     subset?(proj_2(par2), proj_2(par1))
```

Lema 3.3 *La relación* $\leq$ *es un orden parcial en el conjunto de los conceptos de un contexto* $R(O, A, I)$.

```
subconcepto_c_es_orden_parcial: LEMMA
   partial_order?(subconcepto_c?(C))
```

Para probar que $R(O, A, I)$ es un retículo completo, tenemos que establecer que todo conjunto de conceptos posee ínfimo y supremo. Previamente, probamos que la intersección arbitraria de intenciones (resp. extensiones) es una intención (resp. extensión).

Lema 3.4 *Sea* $\mathcal{F}$ *una familia de conjuntos de objetos y* $\mathcal{G}$ *una familia de conjuntos de atributos en un contexto formal* C. *Se tiene que*

$$(\bigcup \mathcal{F})' = \bigcap \{X' : X \in \mathcal{F}\}$$

$$(\bigcup \mathcal{G})' = \bigcap \{Y' : Y \in \mathcal{G}\}$$

La formalización en PVS queda como sigue:

```
intencion_union_gen: LEMMA
 FORALL (F: set[c_objetos(C)]):
  intencion(C)(union(F)) = intersection(conj_intenciones_c(C)(F))

extension_union_gen: LEMMA
FORALL (G: set[c_atributos(C)]):
 extension(C)(union(G)) = intersection(conj_extensiones_c(C)(G))
```

donde `conj_intenciones_c(C)(F)` es la imagen por `intencion(C)` de `F` y `conj_extensiones_c(C)(G)` es la imagen por `extension(C)` de `G`.

Así, estamos en condiciones de establecer en PVS el teorema siguiente:

Teorema 3.5 *$(R(O,A,I),\leq)$ es un retículo completo en el que, para cualquier conjunto de conceptos $\{(X_k,Y_k) : k \in K\}$, el supremo y el ínfimo vienen dados por*

$$\sup(\{(X_k,Y_k) : k \in K\}) = \Big((\bigcup_{k\in K} X_k)'', \bigcap_{k\in K} Y_k \Big)$$

$$\inf(\{(X_k,Y_k) : k \in K\}) = \Big(\bigcap_{k\in K} X_k, (\bigcup_{k\in K} Y_k)'' \Big)$$

Para ello, definimos en PVS las funciones que obtienen el supremo y el ínfimo de un conjunto de conceptos:

```
supremo(C)(S:set[concepto_c(C)]): concepto_c(C) =
 (extension(C)(intencion(C)(union(conj_ext_conceptos(C)(S)))),
  intersection(conj_int_conceptos(C)(S)))

infimo(C)(S:set[concepto_c(C)]): concepto_c(C) =
 (intersection(conj_ext_conceptos(C)(S)),
  intencion(C)(extension(C)(union(conj_int_conceptos(C)(S)))))
```

donde `conj_ext_conceptos(C)(S)` (resp. `conj_int_conceptos(C)(S)`) es el conjunto de las extensiones (resp. intenciones) de los conceptos de `S`.

Obsérvese que, en la definición de estas funciones, se ha hecho uso de subtipos para restringir tanto los dominios como los rangos. Por tanto, el "comprobador de tipos" de PVS genera condiciones obligatorias como la siguiente:

```
supremo_TCC1: OBLIGATION
  FORALL (C: contexto_formal, S: set[concepto_c(C)]):
   es_concepto_c?(C)
       (extension(C)
           (intencion(C)
              (union[objetos(C)]
                 (conj_ext_conceptos(C)(S)))),
        intersection[atributos(C)](conj_int_conceptos(C)(S)))
```

Esta condición garantiza que el supremo de un conjunto de conceptos pertenece al retículo.

Por otra parte, hemos de usar en PVS los conceptos de ínfimo y supremo en un conjunto parcialmente ordenado. Para ello, construimos la teoría `op_defs`, en la que establecemos las definiciones correspondientes sobre un conjunto D, con un orden parcial $\leq$.

```
op_defs[D: TYPE+, <=: (partial_order?[D])]: THEORY

BEGIN
 x, y: VAR D
 A   : VAR set[D]

 csup?(x, A): bool = (FORALL (a: (A)): a <= x)
 cinf?(x, A): bool = (FORALL (a: (A)): x <= a)

 CSUP(A): set[D] = { x: D | csup?(x, A) }
 CINF(A): set[D] = { x: D | cinf?(x, A) }

 supremo?(x,A): bool = csup?(x,A) AND FORALL (y:(CSUP(A))): x <= y
 infimo?(x,A): bool = cinf?(x,A) AND FORALL (y:(CINF(A))): y <= x

END op_defs
```

Utilizando esta teoría se demuestra el siguiente resultado:

Lema 3.6 *Para todo conjunto de objetos S, $infimo(S)$ es la mayor de las cotas inferiores de S, y $supremo(S)$ es la menor de las cotas superiores de S.*

```
infimo_es_infimo_c: LEMMA
 FORALL (S:set[concepto_c(C)]):
  infimo?[concepto_c(C), subconcepto_c?(C)](infimo(C)(S), S)

supremo_es_supremo_c: LEMMA
 FORALL (S:set[concepto_c(C)]):
  supremo?[concepto_c(C), subconcepto_c?(C)](supremo(C)(S), S)
```

4 Generación de todos los conceptos de un contexto finito

En esta sección realizamos una especificación en PVS de un algoritmo para obtener todos los conceptos de un contexto formal finito, y verificamos las propiedades de corrección y completitud de dicho algoritmo.
La idea para determinarlo se basa en las siguientes propiedades:

(i) Todo concepto de un contexto $CF = (O, A, I)$ es de la forma (Y', Y'') para algún conjunto de atributos Y. Recíprocamente, todos los pares de esta forma son conceptos del contexto CF.
(ii) Para todo conjunto de atributos Y, se tiene $Y' = \bigcap_{a \in Y} \{a\}'$

El algoritmo sería, pues, como sigue:

1. Generar el conjunto de todas las extensiones del contexto, usando (ii). Para ello:

- Comenzamos tomando $\mathcal{S} = \{O\}$, pues $\emptyset' = O$
- Para cada $a \in A$: añadimos a $\mathcal{S}$ el conjunto $D \cap \{a\}'$, para cada $D \in \mathcal{S}$.

Si $A = \{a_1, \ldots, a_k\}$, entonces:

- Paso 0: $\mathcal{S} = \{O\}$
- Paso 1: $\mathcal{S} = \{O, \{a_1\}'\}$, pues $O \cap \{a_i\}' = \{a_i\}'$
- Paso 2: $\mathcal{S} = \{O, \{a_1\}', \{a_2\}', \{a_1, a_2\}'\}$, pues $\{a_1, a_2\}' = \{a_1\}' \cap \{a_2\}'$
- Paso 3: $\mathcal{S} = \{O, \{a_1\}', \{a_2\}', \{a_1, a_2\}', \{a_3\}', \{a_1, a_3\}', \{a_2, a_3\}', \{a_1, a_2, a_3\}'\}$

En nuestro ejemplo, la generación de las extensiones sería:

- Paso 0: $\mathcal{S} = \{\{g, s, r, m, p\}\}$
- Paso 1: $\mathcal{S} = \{\{g, s, r, m, p\}\}$ (int. con $\{N\}'$).
- Paso 2: $\mathcal{S} = \{\{g, s, r, m, p\}, \{s, r, p\}\}$ (int. con $\{A\}'$)
- Paso 3: $\mathcal{S} = \{\{g, s, r, m, p\}, \{s, r, p\}, \{g, s, r, p\}\}$ (int. con $\{M\}'$)
- Paso 4: $\mathcal{S} = \{\{g, s, r, m, p\}, \{s, r, p\}, \{g, s, r, p\}, \{g, r\}, \{r\}\}$ (int. con $\{P\}'$)

2. Para cada elemento del conjunto generado en el apartado (1), obtener su intención y formar el concepto correspondiente. De esta forma, se tendría el conjunto de todos los conceptos del contexto. En el ejemplo, los conceptos son:

   - $(\{g, s, r, m, p\}, \{N\})$
   - $(\{s, r, p\}, \{N, A, M\})$
   - $(\{g, s, r, p\}, \{N, M\})$
   - $(\{g, r\}, \{N, M, P\})$
   - $(\{r\}, \{N, A, M, P\})$

El algoritmo para generar las extensiones de los conceptos en un contexto finito $CF = (O, A, I)$ es un proceso recursivo sobre el conjunto de atributos del contexto. En PVS, los procesos recursivos sobre un conjunto finito B, los especificamos haciendo uso de la función de elección, `choose(B)`, y del conjunto resultante de eliminar de `B` el elemento elegido, `rest(B)`. Para usar dicha función de elección sobre un conjunto B de tipo `set[T]`, es necesario que T sea un tipo no vacío. Por esta razón, en la especificación de `contexto_formal_finito` hemos exigido que el conjunto de atributos fuera del tipo `non_empty_finite_set`$[T_2]$.

Por otra parte, al realizar en PVS especificaciones de algoritmos recursivos hemos de proporcionar una función de medida adecuada para el argumento sobre el que se hace la recursión, que garantice la terminación del mismo. En este caso, la medida será el cardinal del conjunto. Además, para razonar sobre la especificación, necesitamos manejar esquemas de inducción sobre conjuntos finitos, así como condiciones necesarias y suficientes para que un determinado conjunto sea finito. Para ello, utilizamos las teorías correspondientes (`finite_sets_inductions` y `finite_sets_eq`) de la biblioteca de PVS relativa a conjuntos finitos.

La especificación en PVS del algoritmo quedaría, pues, como sigue:

```
CF: VAR contexto_formal_finito

agrega_extension_elto(CF)(a: atributos(CF),
                          S: finite_set[set[objetos(CF)]]):
                          finite_set[set[objetos(CF)]] =
  union(S, {X: set[objetos(CF)] |EXISTS (D: (S)):
               X=intersection(D, extension(CF)(singleton(a)))})

genera_extensiones_aux(CF)(A: finite_set[atributos(CF)],
                           S: finite_set[set[objetos(CF)]]):
           RECURSIVE finite_set[set[objetos(CF)]] =
  IF empty?(A) THEN S ELSE
  genera_extensiones_aux(CF)
      (rest(A),
       agrega_extension_elto(CF)(choose(A), S))
  ENDIF
  MEASURE card(A)

genera_extensiones(CF): finite_set[set[objetos(CF)]] =
  genera_extensiones_aux(CF)
     (atrib(CF),
      singleton[set[objetos(CF)]](obj(CF)))

genera_conceptos(CF): set[[set[objetos(CF)],
                           set[atributos(CF)]]] =
  {par: [set[objetos(CF)], set[atributos(CF)]] |
    EXISTS (X: (genera_extensiones(CF))):
               par = (X, intencion(CF)(X))}
```

Queremos establecer que esta especificación es correcta y completa.

Teorema 4.1 *Sean $CF = (O, A, I)$ un contexto formal finito, X un conjunto de objetos e Y un conjunto de atributos. Entonces,*
(X, Y) *es un concepto de* $CF \iff (X, Y) \in$ `genera_conceptos(CF)`

En primer lugar, probamos que el algoritmo es correcto

```
genera_conceptos_correcto: LEMMA
 FORALL (par: [set[objetos(CF)], set[atributos(CF)]]):
  member(par, genera_conceptos(CF)) =>
  es_concepto_c?(CF)(par)
```

para lo cual establecemos la propiedad de corrección de `genera_extensiones`.

```
genera_extensiones_correcto: LEMMA
  es_conj_extensiones?(CF)(genera_extensiones(CF))
```

En segundo lugar, probamos la completitud

```
genera_conceptos_completo: LEMMA
 FORALL(par: concepto_c(CF)):
  member(par, genera_conceptos(CF))
```

a partir, también, de la completitud de `genera_extensiones`

```
genera_extensiones_completo: LEMMA
 FORALL(Y: finite_set[atributos(CF)]):
  member(extension(CF)(Y), genera_extensiones(CF))
```

De entre las propiedades necesarias para la prueba de este lema, destacamos la siguiente:

Lema 4.2 *Sea Y un conjunto finito de atributos y $\mathcal{S}$ un conjunto finito de conjuntos de objetos. Se tiene que dados Z, conjunto finito de atributos, y $D \in \mathcal{S}$, si $Z \subseteq Y$, entonces $D \cap Z' \in$* `genera_extensiones_aux(CF)(Y,S)`

```
genera_extensiones_aux_completo_l3:LEMMA
 FORALL (Y: finite_set[atributos(CF)],
         S: finite_set[set[objetos(CF)]]):
  FORALL (Z: finite_set[atributos(CF)], D: (S)):
    subset?(Z, Y) => member(intersection(D, extension(CF)(Z)),
                     genera_extensiones_aux(CF)(Y, S))
```

La prueba se realiza por inducción en `Y` según el siguiente esquema:

```
finite_set_induction_rest: THEOREM
   P(emptyset[T])
   AND (FORALL SS : P(rest(SS)) IMPLIES P(SS))
         IMPLIES (FORALL S : P(S))
```

5 Conclusiones y trabajo futuro

La especificación formal que se ha hecho del algoritmo para generar todos los conceptos de un contexto formal finito `CF`, usa operaciones sobre conjuntos finitos que no son evaluables. Por ejemplo, respecto a la función de elección `choose`, se tiene que si el conjunto `A` es no vacío, `choose(A)` es un elemento de `A`, pero no está determinado qué elemento se elige, ni el conjunto que queda, `rest(A)`. Lo mismo podemos decir respecto de la operación `union` o de la propia función `genera_conceptos`.

Por ello, abordamos la posibilidad de refinar esta especificación del algoritmo por otra que se pueda evaluar en el evaluador básico de PVS (pvs-ground-evaluator). Observamos que necesitamos realizar, previamente, un refinamiento

de los conjuntos finitos y de las operaciones sobre ellos. Usándolo podremos realizar el correspondiente refinamiento evaluable de `genera_conceptos`.

Hasta ahora, hemos realizado un refinamiento específico de la teoría previamente desarrollada, que nos ha permitido la obtención de todos los conceptos de un contexto formal finito concreto. Por otra parte, estamos desarrollando en PVS un marco general en el que realizar refinamientos de teorías, de forma que las especificaciones de los algoritmos se realicen en un nivel abstracto, donde podamos razonar sobre ellos de forma elegante. Y, posteriormente, poder realizar distintos refinamientos, de forma que las propiedades sobre los algoritmos se conserven y estos últimos sean evaluables.

En el trabajo desarrollado hasta ahora se ha puesto de manifiesto la utilidad de un sistema como PVS para formalizar la teoría AFC. En esta formalización se ha hecho un amplio uso de la teoría de conjuntos finitos de PVS, poniéndose de manifiesto que las pruebas llevadas a cabo en PVS han sido de la misma naturaleza que las habituales en la teoría AFC.

En cuanto al trabajo futuro:

- Como ya hemos comentado, estamos desarrollando un marco general para el refinamiento de teorías en PVS. Y, en particular, en refinamientos evaluables de la teoría de conjuntos finitos que, posteriormente, conducirán a refinamientos de otras teorías basadas en ella.
- El algoritmo establecido aquí para determinar todos los conceptos no es adecuado para contextos grandes, ya que requiere recorrer la lista todas las veces. En [2] se describe otro algoritmo más rápido basado en la generación de todas las clausuras de un determinado operador (en nuestro caso, el operador sería $A \rightarrow A''$). Queremos ampliar la teoría AFC, incluyendo la formalización, verificación y refinamiento evaluable de dicho algoritmo.
- En las aplicaciones prácticas del AFC, a veces es necesario clasificar un gran número de objetos respecto de un número relativamente pequeño de atributos, lo que puede hacer impracticable manejar el contexto completo para la determinación de los conceptos. En tales casos, el retículo de los conceptos puede ser inferido a partir de las "implicaciones entre atributos". En este sentido, se puede estudiar el sistema de las implicaciones entre atributos desde un punto de vista semántico, y obtener un conjunto de implicaciones no redundante y completo. Estamos trabajando también en su formalización dentro de PVS.

Referencias

1. *PVS (Prototype Verification System), 2002.* Ver `http://pvs.csl.sri.com`
2. Ganter, B. y Wille, R. *Formal Concept Analysis.* Springer–Verlag, 1999.
3. Owre, S.; Rushby, J.M.; Shankar, N. y Stringer-Calvert, D.W.J. . *PVS System Guide.* `http://pvs.csl.sri.com/`
4. Schwarzweller, C. *Mizar Formalization of Concept Lattices.* En Mechanizae Mathematics and its Applications, vol. 1, 2000

5. Stumme, G.; Wille, R. y Wille, U. *Conceptual Knowledge Discovery in Databases Using Formal Concept Analysis Methods*. En: J. M. Zytkow, M. Quafofou (eds.): Principles of Data Mining and Knowledge Discovery. Proc. 2nd European Symposium on PKDD '98, LNAI 1510, Springer, Heidelberg 1998, 450-458
6. Vogt, F. y Wille, R. *TOSCANA– a Graphical Tool for Analyzing and Exploring Data*. Lecture Notes of Computer Science, vol 894, Heidelberg, 1995. Springer.
7. Wille, R. *Knowledge Acquisition by Methods of Formal Concept Analysis*. En E. Diday, editor, Data Analysis, Learning Symbolic and Numeric Knowledge, pp. 365-380. 1989.