Formalización de la lógica descriptiva $\mathcal{ALC}$ en PVS

José A. Alonso Jiménez
María J. Hidalgo Doblado
Francisco J. Martín Mateos
José L. Ruiz Reina

Grupo de Lógica Computacional
Dpto. de Ciencias de la Computación e Inteligencia Artificial
Universidad de Sevilla
Sevilla, 23 de marzo de 2006

Resumen

Este trabajo presenta una formalización de la lógica descriptiva $\mathcal{ALC}$ en PVS. Está basado fundamentalmente en el capítulo 2 de la tesis de Tobies [Tob01] y en el capítulo de Baader y Nutt [BN03] en "The Description Logic Handbook" [BMNPS03]. Como bibliografía complementaria se ha usado el capítulo 1 de la tesis de Lutz [Lut02] y los manuales de RacerPro [Racb] y [Raca].

Índice general

Capítulo 1

Sintaxis de $\mathcal{ALC}$

En este capítulo se formaliza la sintaxis de $\mathcal{ALC}$.

1.1. Los conceptos de $\mathcal{ALC}$

Definición 1.1.1 (Nombres de conceptos y de roles)
Para la construcción de los conceptos se parte de un conjunto de ***nombres de conceptos****,* NC*, y un conjunto de* ***nombres de roles****,* NR*.*

Notación 1.1.2 Se usarán A y B como variables sobre nombres de conceptos y R como variables sobre nombres de roles.

Definición 1.1.3 (Constructores)
Los ***constructores*** *de los conceptos de $\mathcal{ALC}$ son*

- $\neg$ *(negación o complementario)*
- $\sqcap$ *(intersección)*
- $\sqcup$ *(unión)*
- $\forall$ *(cuantificador universal o generalizador)*
- $\exists$ *(cuantificador existencial o particularizador)*

Definición 1.1.4 (Conceptos)
El ***conjunto de los conceptos*** *de $\mathcal{ALC}$ se construye, a partir de* NC *y* NR*, mediante las siguientes reglas:*

- *Los nombres de conceptos son conceptos.*
- *Si C es un concepto, entonces $\neg C$ es un concepto.*
- *Si C y D son conceptos, entonces $C \sqcap D$ y $C \sqcup D$ son conceptos.*
- *Si C es un concepto y R es un nombre de rol, entonces $\forall R.C$ y $\exists R.C$ son conceptos.*

```
alc_concept[NC: TYPE+, NR: TYPE]: DATATYPE
 BEGIN
  alc_a(name: NC)                        : alc_atomic?
```

```
 alc_not(conc: alc_concept)            : alc_not?
 alc_and(conc1,conc2: alc_concept)     : alc_and?
 alc_or(conc1,conc2: alc_concept)      : alc_or?
 alc_all(role: NR, conc: alc_concept)  : alc_all?
 alc_some(role: NR, conc: alc_concept) : alc_some?
END alc_concept
```

Notación 1.1.5 Se usarán C, D, C_1, C_2 como variables sobre conceptos de $\mathcal{ALC}$.

Nota 1.1.6 (Constructores de conceptos) En la definición del TAD concepto se han introducido los siguientes constructores:

`alc_a(A)`	=	A
`alc_not(D)`	=	$\neg C$
`alc_and(C,D)`	=	$C \sqcap D$
`alc_or(C,D)`	=	$C \sqcup D$
`alc_all(R,C)`	=	$\forall R.C$
`alc_some(R,C)`	=	$\exists R.C$

Nota 1.1.7 (Accesores de conceptos) En la definición del TAD concepto se han introducido los siguientes accesores:

`name(alc_a(A))`	=	`A`
`conc(alc_not(D))`	=	`C`
`conc1(alc_and(C,D))`	=	`C`
`conc2(alc_and(C,D))`	=	`D`
`conc1(alc_or(C,D))`	=	`C`
`conc2(alc_or(C,D))`	=	`D`
`role(alc_all(R,C))`	=	`R`
`conc(alc_all(R,C))`	=	`C`
`role(alc_some(R,C))`	=	`R`
`conc(alc_some(R,C))`	=	`C`

Nota 1.1.8 (Reconocedores de conceptos) En la definición del TAD concepto se han introducido los siguientes reconocedores:

- `alc_atomic?(C)`, que se verifica si C es atómico (i.e. $C \in \mathsf{NC}$),
- `alc_not?(C)`, que se verifica si C es una negación (i.e. existe un concepto D tal que C es $\neg D$),
- `alc_and?(C)`, que se verifica si C es una intersección (i.e. existen conceptos C_1 y C_2 tales que C es $C_1 \sqcap C_2$),
- `alc_or?(C)`, que se verifica si C es una unión (i.e. existen conceptos C_1 y C_2 tales que C es $C_1 \sqcup C_2$),

- `alc_all?(C)`, que se verifica si C es una generalización (i.e. existe un rol R y un concepto D tales que C es $\forall R.D$),
- `alc_some?(C)`, que se verifica si C es una particularización (i.e. existe un rol R y un concepto D tales que C es $\exists R.D$),

Nota 1.1.9 (Constructores de conceptos en RACER) En la siguiente tabla se muestra la correspondencia entre las notaciones de los constructores de concepto en PVS, DL y RACER:

PVS	DL	RACER
`alc_not(D)`	$\neg C$	`(not D)`
`alc_and(C,D)`	$C \sqcap D$	`(and C D)`
`alc_or(C,D)`	$C \sqcup D$	`(or C D)`
`alc_all(R,C)`	$\forall R.C$	`(all R C)`
`alc_some(R,C)`	$\exists R.C$	`(some R C)`

Definición 1.1.10 (Conceptos atómicos y complejos)
*Un concepto C es **atómico** si $C \in$ NC y es **complejo** si $C \notin$ NC.*

1.2. Definiciones sobre la sintaxis de $\mathcal{ALC}$

En esta sección se definen constructores auxiliares y nociones sintácticas sobre los conceptos de $\mathcal{ALC}$, como las de tamaño y ocurrencia.

Nota 1.2.1 La teoría `alc_concept_defs` depende del conjunto de nombres de conceptos NC y del conjunto de nombres de roles NR.

```
alc_concept_defs[NC: TYPE+, NR: TYPE]: THEORY
 BEGIN
```

Nota 1.2.2 Se usará el tipo de datos de conceptos de $\mathcal{ALC}$ construidos con el conjunto de nombres de conceptos NC y el conjunto de nombres de roles NR.

```
  IMPORTING alc_concept_adt[NC, NR]
```

Nota 1.2.3 Se usarán A y B como variables sobre nombres de conceptos y C, D, C_1, C_2 como variables sobre conceptos.

```
  A, B:          VAR NC
  C, D, C1, C2: VAR alc_concept
```

Nota 1.2.4 Se representará mediante A_1 el nombre de un concepto (que se usará en las definiciones siguientes de $\top$ y $\bot$).

```
A1: NC
```

Definición 1.2.5 (Concepto universal)
*El **concepto universal** es la diyunción de un nombre de concepto y su complementario; es decir,* $\top = A_1 \sqcup \neg A_1$.

```
alc_top: alc_concept =
  alc_or(alc_a(A1),alc_not(alc_a(A1)))
```

Definición 1.2.6 (Concepto vacío)
*El **concepto vacío** es la conjunción de un nombre de concepto y su complementario; es decir,* $\bot = A_1 \sqcap \neg A_1$.

```
alc_bottom: alc_concept =
  alc_and(alc_a(A1),alc_not(alc_a(A1)))
```

Definición 1.2.7 (Implicación de conceptos)
Se define $C \to D$ *como una abreviatura de* $\neg C \sqcup D$.

```
alc_impl(C,D): alc_concept =
  alc_or(alc_not(C),D)
```

Definición 1.2.8 (Equivalencia de conceptos)
Se define $C \leftrightarrow D$ *como una abreviatura de* $(C \to D) \sqcap (D \to C)$.

```
alc_equiv(C,D): alc_concept =
  alc_and(alc_impl(C,D), alc_impl(D,C))
```

Definición 1.2.9 (Tamaño de un concepto)
*El **tamaño del concepto** C, $|C|$, es el número de símbolos necesarios para construirlo. Se define por recursión en la estructura de* C*:*

$$\begin{aligned}
|A| &= 1 \\
|\neg D| &= 1 + |D| \\
|C_1 \sqcap C_2| &= 1 + |C_1| + |C_2| \\
|C_1 \sqcup C_2| &= 1 + |C_1| + |C_2| \\
|\forall R.D| &= 2 + |D| \\
|\exists R.D| &= 2 + |D|
\end{aligned}$$

```
size(C): RECURSIVE nat =
 CASES C OF
  alc_a(A)      : 1,
  alc_not(D)    : 1 + size(D),
  alc_and(C1,C2): 1 + size(C1) + size(C2),
  alc_or(C1,C2) : 1 + size(C1) + size(C2),
```

```
  alc_all(R,D)  : 2 + size(D),
  alc_some(R,D) : 2 + size(D)
 ENDCASES
 MEASURE C BY <<
```

Definición 1.2.10 (Ocurrencias de nombres en conceptos)
El nombre A ***ocurre en*** *el concepto C si se usa A en la construcción de C. Se define por recursión en la estructura de C,*

- *A ocurre en B syss $A = B$*
- *A ocurre en $\neg D$ syss A ocurre en D*
- *A ocurre en $C_1 \sqcap C_2$ syss A ocurre en C_1 o en C_2*
- *A ocurre en $C_1 \sqcup C_2$ syss A ocurre en C_1 o en C_2*
- *A ocurre en $\forall R.D$ syss A ocurre en D*
- *A ocurre en $\exists R.D$ syss A ocurre en D*

```
occur_in(A,C): RECURSIVE bool =
  CASES C OF
   alc_a(B)      : A = B,
   alc_not(D)    : occur_in(A,D),
   alc_and(C1,C2): occur_in(A,C1) OR occur_in(A,C2),
   alc_or(C1,C2) : occur_in(A,C1) OR occur_in(A,C2),
   alc_all(R,D)  : occur_in(A,D),
   alc_some(R,D) : occur_in(A,D)
  ENDCASES
  MEASURE C BY <<
```

Nota 1.2.11 Con esto termina la teoría `alc_concept_defs`.

```
END alc_concept_defs
```

1.3. Conocimiento terminológico y asertivo

Los sistemas de lógicas descriptivas organizan el conocimiento en dos categorías:

- *conocimiento terminológico*, que relaciona los conceptos y se almacena en la caja–T (en inglés, "Tbox"), y
- *conocimiento asertivo*, que relaciona los individuos con los conceptos y con los roles y se almacen en la caja–A (en inglés, "Abox"),

Juntos forman las *bases de conocimiento*.

Nota 1.3.1 La teoría `kb_syntax` formaliza la sintaxis de las bases de conocimiento y depende del conjunto de nombres de conceptos NC, del conjunto de nombres de roles NR y del conjunto de nombres de individuos, NI (cuya función se explicará en la definición de los axiomas asertivos (1.3.24 de la página 13)).

```
kb_syntax[NC:TYPE+, NR: TYPE, NI: TYPE]: THEORY
 BEGIN
```

Nota 1.3.2 Se usarán las definiciones sobre conceptos de $\mathcal{ALC}$ construidas con el conjunto de nombres de conceptos NC y el conjunto de nombres de roles NR.

```
  IMPORTING alc_concept_defs[NC,NR]
```

Nota 1.3.3 Se usarán A y B como variables sobre nombres de conceptos.

```
  A, B: VAR NC
```

1.3.1. Conocimiento terminológico

Definición 1.3.4 (Axiomas terminológicos)
Si C y D son conceptos, entonces

- *$C \sqsubseteq D$ es un axioma terminológico general, que se llama **axioma de inclusión conceptual general** (en inglés, "general concept inclusions (GCI)") o, simplemente, **inclusión** y*
- *$C \equiv D$ es un axioma terminológico general, que se llama **axioma de igualdad conceptual general** o, simplemente, **igualdad**.*

```
  alc_gtax: DATATYPE
    BEGIN
      gci(antecedent,consequent: alc_concept)        : gci?
      concept_eq(antecedent,consequent: alc_concept) : concept_eq?
    END alc_gtax
```

Nota 1.3.5 En la definición del TAD se han introducido los siguientes

- constructores:
  - `gci(C,D)` que representa $C \sqsubseteq D$ y
  - `concept_eq(C,D)` que representa $C \equiv D$.
- accesores:
  - `antecedent(gci(C,D)) = C`,

  - `consequent(gci(C,D)) = D`,
  - `antecedent(concept_eq(C,D)) = C`,
  - `consequent(concept_eq(C,D)) = D`,

- reconocedores:
  - `gci?(X)`, que se verifica si `X` es un axioma de inclusión conceptual general y
  - `concept_eq?(X)`, que se verifica si `X` es un axioma de igualdad conceptual general.

Nota 1.3.6 En RACER,

- $C \sqsubseteq D$ se representa por `(implies C D)`,
- $C \equiv D$ se representa por `(equivalent C D)`,

Notación 1.3.7 Se usarán Ax, Ax_1, Ax_2 como variables sobre axiomas terminológicos generales.

```
Ax, Ax1, Ax2: VAR alc_gtax
```

Definición 1.3.8 (Especialización)
*Una **especialización** es un axioma de inclusión cuyo antecedente es un concepto atómico, i.e. $A \sqsubseteq C$ (ref. [BMNPS03] pág. 16).*

```
specialization?(Ax): bool =
  gci?(Ax) AND (alc_atomic?(antecedent(Ax)))
```

Nota 1.3.9 En RACER la especialización $A \sqsubseteq C$ se representa por
`(define-primitive-concepto A C)`
y se llama definición de concepto primitivo.

Definición 1.3.10 (Definición)
*Una **definición** es un axioma de igualdad cuyo antecedente es un concepto atómico, i.e. $A \equiv C$ (ref. [BMNPS03] pág. 9).*

```
definition?(Ax): bool =
  concept_eq?(Ax) AND (alc_atomic?(antecedent(Ax)))
```

Nota 1.3.11 En RACER la definición $A \equiv C$ se representa por
`(define-concept A C)`.

Definición 1.3.12 (Caja–T)
*Una **caja–T** (terminología o TBox) es un conjunto finito de axiomas.*

```
TBox: TYPE = finite_set[alc_gtax]
```

Notación 1.3.13 Se usarán $\mathcal{T}$, $\mathcal{T}_1$ y $\mathcal{T}_2$ como variables sobre cajas–T.

```
T, T1, T2: VAR TBox
```

Definición 1.3.14 (Caja simple)
*Una caja–T $\mathcal{T}$ es **simple** si:*

- *los antecedentes de los axiomas de $\mathcal{T}$ son atómicos; es decir están formados sólo por nombres de conceptos;*
- *cada nombre de concepto ocurre como máximo una vez en los antecedentes de los axiomas de $\mathcal{T}$ y*
- *$\mathcal{T}$ es acíclica.*

Nota 1.3.15 Los siguientes conceptos se introducen para facilitar la formalización de caja–T simple. La definición de caja simple es la 1.3.23 en la página 13.

Definición 1.3.16
Caja–T cuyos antecedentes son atómicos.

```
all_antecedents_atomics(T): bool =
  FORALL Ax: member(Ax,T) IMPLIES alc_atomic?(antecedent(Ax))
```

Definición 1.3.17
Conjunto de los antecedentes atómicos de una caja–T.

```
set_atomic_antecedents(T): finite_set[alc_concept] =
  {C: alc_concept | EXISTS Ax: (member(Ax,T) AND
                                alc_atomic?(antecedent(Ax)) AND
                                C = antecedent(Ax))}
```

Definición 1.3.18
Caja–T cuyos antecedentes son atómicos y tal que cada uno de sus antecedentes ocurre como máximo una vez como antecedente de algún axioma.

```
antecedents_atomic_once(T): bool =
  all_antecedents_atomics(T) AND card(set_atomic_antecedents(T)) = card(T)
```

Definición 1.3.19
*Un nombre de concepto A **usa directamente** el nombre de concepto B, respecto de $\mathcal{T}$, si existe algún axioma en $\mathcal{T}$ cuyo antecedente es A y B ocurre en el consecuente de dicho axioma.*

```
directely_use_tbox(T)(A, B): bool =
  EXISTS Ax: member(Ax,T) AND
             alc_atomic?(antecedent(Ax)) AND
             name(antecedent(Ax)) = A AND
             occur_in(B, consequent(Ax))
```

Nota 1.3.20 Se usa la teoría correspondiente a la clausura transitiva de una relación.

```
IMPORTING wf@cl_tr[NC]
```

Definición 1.3.21
*La relación **usa** es la clausura reflexiva–transitiva de la relación* usa directamente.

```
use_tbox(T): pred[[NC,NC]] = tr_cl(directely_use_tbox(T))
```

Definición 1.3.22 (Caja acíclica)
*Una caja–T se denomina **acíclica** si no existe ningún nombre de concepto que se use a sí mismo.*

```
acyclic(T): bool =
  NOT EXISTS A: use_tbox(T)(A,A)
```

Definición 1.3.23 (Caja simple)
*Una caja–T $\mathcal{T}$ es **simple** si:*

- *los antecedentes de los axiomas de $\mathcal{T}$ son atómicos; es decir están formados sólo por nombres de conceptos;*
- *cada nombre de concepto ocurre como máximo una vez en los antecedentes de los axiomas de $\mathcal{T}$ y*
- *$\mathcal{T}$ es acíclica.*

```
simple(T): bool =
 antecedents_atomic_once(T) AND acyclic(T)
```

1.3.2. Conocimiento asertivo

Definición 1.3.24 (Axiomas asertivo)
*Los **axiomas asertivos** son las expresiones de la forma $x : C$, $(x, y) : R$ y $x \neq y$, donde x, y son nombres de individuos, C es un concepto y R es un nombre de rol.*

```
assertional_ax: DATATYPE
  BEGIN
    instanceof(left:NI, right:alc_concept)       :instanceof?
    related(left:NI, role:NR, right:NI)          :related?
    different_ni(left:NI, right:NI)              :different_ni?
  END assertional_ax
```

Nota 1.3.25 En la definición del TAD se han introducido los siguientes

- constructores:

  - `instanceof(x,C)` que representa $x : C$,
  - `related(x,R,y)` que representa $(x, y) : R$,
  - `different_ni(x,y)` que representa $x \neq y$.

- accesores:
  - `left(instanceof(x,C)) = x`,
  - `right(instanceof(x,C)) = C`,
  - `left(related(x,R,y)) = x`,
  - `role(related(x,R,y)) = R`,
  - `right(related(x,R,y)) = y`,
  - `left(different_ni(x,C)) = x`,
  - `right(different_ni(x,C)) = y`.
- reconocedores:
  - `instanceof?(X)`, que se verifica si `X` es una expresión de la forma $x : C$,
  - `related?(X)`, que se verifica si `X` es una expresión de la forma $(x, y) : R$,
  - `different_ni?(X)`, que se verifica si `X` es una expresión de la forma $x \neq y$.

Nota 1.3.26 En RACER,

- $x : C$ se representa por `(instance x C)` y
- $(x, y) : R$ se representa por `(related x y R)`.

Además, usa la hipótesis de nombres únicos.

Definición 1.3.27 (Caja–A)
*Una **caja–A** es un conjunto finito de axiomas asertivos.*

```
ABox: TYPE = finite_set[assertional_ax]
```

Notación 1.3.28 Se usarán A, A_1, A_2 como variables sobre axiomas asertivos.

```
Aa, Aa1, Aa2 : VAR assertional_ax
```

Definición 1.3.29 (Base de conocimiento)
*Una **base de conocimiento** es un par formado por una caja–T y una caja–A.*

```
KnowledgeBase: TYPE = [TBox, ABox]
```

Notación 1.3.30 Se usará $\mathcal{K}$ como variable sobre bases de conocimiento.

```
KB: VAR KnowledgeBase
```

Nota 1.3.31 Con esto finaliza la descripción de la teoría `kb_syntax`.

```
END kb_syntax
```

Capítulo 2

Semántica de $\mathcal{ALC}$

En este capítulo se formaliza la semántica de $\mathcal{ALC}$.

2.1. Semántica de los conceptos de $\mathcal{ALC}$

En esta sección se define la noción de interpretación y las nociones de subsunción y satisfacibilidad de conceptos.

Nota 2.1.1 La formalización de la teoría `semantics_alc_concept` está parametrizada por el conjunto de nombres de conceptos, NC, el conjunto de nombres de roles, NR, el conjunto de nombres de individuos, NI y el conjunto de elementos de los dominios, U.

```
semantics_alc_concept[NC:TYPE+, NR: TYPE, NI: TYPE, U: TYPE+]: THEORY
 BEGIN
```

Nota 2.1.2 Se usarán las definiciones de conceptos de $\mathcal{ALC}$ construido con el conjunto de nombres de conceptos NC y el conjunto de nombres de roles NR.

```
  IMPORTING alc_concept_defs[NC, NR]
```

Nota 2.1.3 Se usarán A y B como variables sobre nombres de conceptos y C, D, C_1, C_2 como variables sobre conceptos.

```
  A, B:          VAR NC
  C, D, C1, C2: VAR alc_concept
```

Nota 2.1.4 Para facilitar la demostración de la existencia de interpretaciones (definición 2.1.10) introducimos los objetos auxiliares a_1, S_1, f_1, f_2 y f_3.

Definición 2.1.5
a_1 *es un elemento del universo.*

```
a_1: U
```

Definición 2.1.6
El conjunto S_1 cuyo único elemento es a_1 es un subconjunto no vacío del universo.

```
S_1: (nonempty?[U]) =
  singleton(a_1)
```

Definición 2.1.7
La función f_1 que aplica cada nombre de concepto en el conjunto vacío es una función de NC *en* 2^{S_1}.

```
f_1: [NC -> (powerset(S_1))] =
  lambda (A:NC): emptyset[U]
```

Definición 2.1.8
La función f_2 que aplica cada nombre de rol en el conjunto vacío es una función de NR *en* $2^{S_1 \times S_1}$.

```
f_2: [NR -> PRED[[(S_1),(S_1)]]] =
  lambda (R:NR): emptyset[[U,U]]
```

Definición 2.1.9
La función f_3 que aplica cada nombre de individuo en a_1 es una función de NI *en* S_1.

```
f_3: [NI -> (S_1)] =
  lambda (x:NI): a_1
```

Definición 2.1.10 (Interpretación)
*Una **interpretación** $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ consta de un conjunto no vacío $\Delta^{\mathcal{I}}$ (el **dominio** de la interpretación) y una aplicación $\cdot^{\mathcal{I}}$ (la **función de interpretación**) que asigna a cada nombre de concepto A un conjunto $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$, a cada nombre de rol R una relación binaria $R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$ y a cada nombre de individuo a un elemento del dominio $a^{\mathcal{I}} \in \Delta^{\mathcal{I}}$.*

```
interpretation: NONEMPTY_TYPE =
  [# int_domain:        (nonempty?[U]),
     int_names_concept: [NC -> (powerset(int_domain))],
     int_names_roles:   [NR -> PRED[[(int_domain),(int_domain)]]],
     int_names_ind:     [NI -> (int_domain)] #]
```

Nota 2.1.11 Una interpretación $\mathcal{I}$ sobre el universo U se representa por una estructura con cuatro campos:

1. el dominio de $\mathcal{I}$, `int_domain(I)`, que es un subconjunto no vacío de U,

2. una función `int_names_concept(I)` que interpreta los nombres de conceptos como subconjuntos del dominio de $\mathcal{I}$,

3. una función `int_names_roles(I)` que interpreta los nombres de roles en relaciones binarias sobre el dominio de $\mathcal{I}$,

4. una función `int_names_ind(I)` que interpreta los nombres de individuos como elementos del dominio de $\mathcal{I}$.

Nota 2.1.12 Al definir la interpretación como un tipo no vacío se genera una obligación de prueba.

Notación 2.1.13 Se usará $\mathcal{I}$ como variable sobre las interpretaciones.

```
I: VAR interpretation
```

2.1.1. Valor de un concepto en una interpretación. Modelos

El objetivo de esta sección es definir el valor de los conceptos en las interpretaciones y distinguir las interpretaciones que son modelos de un concepto.

Definición 2.1.14 (Valor de un concepto)
Sea $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ una interpretación. La función de interpretación se extiende a los conceptos por recursión en la estructura de los conceptos:

$$\begin{array}{lcl} (\neg D)^{\mathcal{I}} & = & \Delta^{\mathcal{I}} \setminus D^{\mathcal{I}} \\ (C_1 \sqcap C_2)^{\mathcal{I}} & = & C_1^{\mathcal{I}} \cap C_2^{\mathcal{I}} \\ (C_1 \sqcup C_2)^{\mathcal{I}} & = & C_1^{\mathcal{I}} \cup C_2^{\mathcal{I}} \\ (\forall R.D)^{\mathcal{I}} & = & \{a \in \Delta^{\mathcal{I}} : (\forall b \in \Delta^{\mathcal{I}})[(a,b) \in R^{\mathcal{I}} \rightarrow b \in D^{\mathcal{I}}]\} \\ (\exists R.D)^{\mathcal{I}} & = & \{a \in \Delta^{\mathcal{I}} : (\exists b \in \Delta^{\mathcal{I}})[(a,b) \in R^{\mathcal{I}} \wedge b \in D^{\mathcal{I}}]\} \end{array}$$

*$C^{\mathcal{I}}$ se llama el **valor** de C en $\mathcal{I}$.*

```
int_concept(C,I): RECURSIVE set[U] =
  CASES C OF
    alc_a(B):       int_names_concept(I)(B),
    alc_not(D):     difference(int_domain(I), int_concept(D,I)),
    alc_and(C1,C2): intersection(int_concept(C1,I),int_concept(C2,I)),
    alc_or(C1,C2):  union(int_concept(C1,I),int_concept(C2,I)),
    alc_all(R,D):   {a:(int_domain(I)) |
                       FORALL (b:(int_domain(I))):
                         int_names_roles(I)(R)(a,b) IMPLIES
                         int_concept(D,I)(b)},
    alc_some(R,D):  {a:(int_domain(I)) |
                       EXISTS (b:(int_domain(I))):
                         int_names_roles(I)(R)(a,b) AND
                         int_concept(D,I)(b)}
  ENDCASES
MEASURE C BY <<
```

Lema 2.1.15 (Valor del falso) $\bot^{\mathcal{I}} = \emptyset$.

```
int_concept_alc_bottom_emptyset: LEMMA
  int_concept(alc_bottom,I) = emptyset
```

Demostración: Trivial.

□

Lema 2.1.16 (Valor de la intersección) $(C \sqcap D)^{\mathcal{I}} = C^{\mathcal{I}} \cap D^{\mathcal{I}}$.

```
int_concept_alc_and: LEMMA
  int_concept(alc_and(C,D),I) = intersection(int_concept(C,I),
                                             int_concept(D,I))
```

Demostración: Trivial.

□

Lema 2.1.17 (Valor de la negación) $(\neg C)^{\mathcal{I}} = \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$.

```
int_concept_alc_not: LEMMA
  int_concept(alc_not(C),I) = difference(int_domain(I), int_concept(C,I))
```

Demostración: Trivial.

□

Lema 2.1.18 *El valor de un concepto C mediante una interpretación $\mathcal{I}$ es un subconjunto del dominio de $\mathcal{I}$.*

```
int_concept_alc_subset_int_domain: LEMMA
  subset?(int_concept(C,I), int_domain(I))
```

Demostración: Por inducción en la estructura de C.

□

Corolario 2.1.19 *Si $u \in C^{\mathcal{I}}$, entonces $u \in \Delta^{\mathcal{I}}$.*

```
member_int_concept_member_int_domain: COROLLARY
  FORALL (u:U): member(u,int_concept(C,I)) IMPLIES
                member(u,int_domain(I))
```

Definición 2.1.20 (Modelo de un concepto)
*La interpretación $\mathcal{I}$ es **modelo** del concepto C si $C^{\mathcal{I}} \neq \emptyset$. Se representa por $\mathcal{I} \models C$.*

```
is_model_concept(I,C): bool =
  nonempty?(int_concept(C,I));

|=(I,C): bool =
  is_model_concept(I,C)
```

2.1.2. Problemas de razonamiento

2.1.2.1. Satisfacibilidad, insatisfacibilidad y tautologicidad

El objeto de esta sección es distinguir los conceptos satisfacibles, insatisfacibles y tautológicos.

Definición 2.1.21 (Concepto satisfacible)
*El concepto C es **satisfacible** si posee algún modelo.*

```
concept_satisfiable?(C): bool =
  EXISTS I: I |= C
```

Definición 2.1.22 (Concepto insatisfacible)
*El concepto C es **insatisfacible** si para toda interpretación $\mathcal{I}$, $C^{\mathcal{I}} = \emptyset$.*

```
concept_unsatisfiable?(C): bool =
  FORALL I: empty?(int_concept(C,I))
```

Lema 2.1.23 *C es insatisfacible syss C no es satisfacible.*

```
unsatisfiable_IFF_NOT_satisfiable: LEMMA
  concept_unsatisfiable?(C) IFF NOT concept_satisfiable?(C)
```

Demostración: Trivial.

□

Definición 2.1.24 (Concepto tautológico)
*Un concepto C es **tautológico** si, para toda interpretación $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$, $C^{\mathcal{I}} = \Delta^{\mathcal{I}}$.*

```
concept_tautological?(C): bool =
  FORALL I: int_concept(C,I) = int_domain(I)
```

Lema 2.1.25 *$\top$ es un concepto tautológico.*

```
top_is_tautological: LEMMA
  concept_tautological?(alc_top)
```

Demostración: Trivial

□

Corolario 2.1.26 *Existen conceptos tautológicos.*

```
exists_tautological: COROLLARY
  EXISTS C: concept_tautological?(C)
```

Lema 2.1.27 *$\bot$ es un concepto insatisfacible.*

```
bottom_is_unsatisfiable: LEMMA
  concept_unsatisfiable?(alc_bottom)
```

Demostración: Trivial

□

Corolario 2.1.28 *Existen conceptos insatisfacibles.*

```
exists_unsatisfiable: COROLLARY
  EXISTS C: concept_unsatisfiable?(C)
```

2.1.2.2. Subsunción y equivalencia

En esta sección se introduce las relaciones de subsunción y equivalencia entre conceptos.

Definición 2.1.29 (Subsunción de conceptos)
*El concepto C está **subsumido** en el concepto D si, para cualquier interpretación $\mathcal{I}$, $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$. Se representa por $C \sqsubseteq D$.*

```
concept_subsumed?(C,D): bool =
  FORALL I: subset?(int_concept(C,I), int_concept(D,I))
```

Definición 2.1.30 (Equivalencia de conceptos)
*Los conceptos C y D son **equivalentes** si $C \sqsubseteq D$ y $D \sqsubseteq C$. Se representa por $C \equiv D$.*

```
concept_equivalent?(C,D): bool =
  concept_subsumed?(C,D) AND concept_subsumed?(D,C)
```

Nota 2.1.31 Con esto se termina la teoría `semantics_alc_concept`.

```
END semantics_alc_concept
```

2.2. Semántica de las bases de conocimiento

En esta sección se establece la semántica de las bases de conocimiento. Además, probamos que los problemas de inferencia que se plantean en las bases de conocimiento se reducen al problema de la satisfacibilidad de conceptos, respecto de la base de conocimiento.

Nota 2.2.1 La formalización de la teoría `kb_semantic` está parametrizada por el conjunto de nombres de conceptos, NC, el conjunto de nombres de roles, NR, el conjunto de nombres de individuos, NI y el conjunto de elementos de los dominios, U.

```
kb_semantic[NC:TYPE+, NR: TYPE, NI: TYPE+, U: TYPE+, x_0: NI]: THEORY
 BEGIN
```

Nota 2.2.2 Se usarán las teorías de la sintaxis de las bases de conocimiento y la de la semántica de los conceptos.

```
  IMPORTING kb_syntax[NC,NR,NI]
  IMPORTING semantics_alc_concept[NC,NR,NI,U]
```

Notación 2.2.3 Se usarán las siguientes variables:

- C, D, C_1, C_2 para conceptos,
- $\mathcal{I}$ para interpretaciones,
- u para elementos del dominio de interpretaciones,
- x para nombres de individuos,
- Ax, Ax_1, Ax_2 para axiomas terminológicos generales,
- $\mathcal{T}, \mathcal{T}_1, \mathcal{T}_2$ para cajas–T,
- Aa, Aa_1, Aa_2 para axiomas asertivos,
- $\mathcal{A}$ para cajas–A,
- $\mathcal{K}$ para bases de conocimiento.

```
  C, D, C1, C2 : VAR alc_concept
  I            : VAR interpretation
  u            : VAR U
  x            : VAR NI
  Ax, Ax1, Ax2 : VAR alc_gtax
  T, T1, T2    : VAR TBox
  Aa, Aa1, Aa2 : VAR assertional_ax
  AB           : VAR ABox
  KB           : VAR KnowledgeBase
```

Definición 2.2.4 (Satisfacibilidad de axiomas terminológicos)
Sea $\mathcal{I}$ una interpretación.

- *$\mathcal{I}$ **satisface** el axioma terminológico $C \sqsubseteq D$ si $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$.*
- *$\mathcal{I}$ **satisface** el axioma terminológico $C \equiv D$ si $C^{\mathcal{I}} = D^{\mathcal{I}}$.*

```
satisfies_gtax(I,Ax): bool =
  CASES Ax OF
    gci(C,D)        : subset?(int_concept(C,I),int_concept(D,I)),
    concept_eq(C,D) : int_concept(C,I) = int_concept(D,I)
  ENDCASES
```

Definición 2.2.5 (Modelo de una caja–T)
*La interpretación $\mathcal{I}$ es **modelo** de una caja–T $\mathcal{T}$ si satisface todos sus axiomas. Se representa por $\mathcal{I} \models \mathcal{T}$.*

```
is_model_TBox(I,T): bool =
  FORALL Ax: member(Ax,T) IMPLIES satisfies_gtax(I,Ax)
```

Definición 2.2.6 (Consistencia de las cajas–T)
*Una caja–T es **consistente** si tiene algún modelo.*

```
tbox_consistent(T): bool =
  EXISTS I: is_model_TBox(I,T)
```

Definición 2.2.7 (Satisfacibilidad relativa de un concepto)
*El concepto C es **satisfacible respecto** de la caja–T $\mathcal{T}$ si existe algún modelo de $\mathcal{T}$ que es modelo de C. En caso contrario, C es **insatisfacible respecto** de $\mathcal{T}$.*

```
satisfiable_tbox(C,T): bool =
  EXISTS I: is_model_TBox(I,T) AND nonempty?(int_concept(C,I))

unsatisfiable_tbox(C,T): bool =
  NOT satisfiable_tbox(C,T)

satisfiable_tbox(T)(C): bool =
  satisfiable_tbox(C,T)

unsatisfiable_tbox(T)(C): bool =
  unsatisfiable_tbox(C,T)
```

Lema 2.2.8 *C es satisfacible repecto de $\emptyset$ syss C es satisfacible.*

```
satisfiable_tbox_empty: LEMMA
  empty?(T) IMPLIES
  (satisfiable_tbox(T)(C) IFF concept_satisfiable?(C))
```

Demostración: Trivial, usando 2.1.20, 2.1.21, 2.2.18 y 2.2.7.

□

Definición 2.2.9 (Subsunción relativa de conceptos)
*El concepto C está **subsumido** en el concepto D respecto de $\mathcal{T}$ si, para cualquier modelo $\mathcal{I}$ de $\mathcal{T}$, $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$. Se representa por $C \sqsubseteq_{\mathcal{T}} D$.*

```
subsumed_tbox(C,D,T): bool =
  FORALL I: is_model_TBox(I,T) IMPLIES
            subset?(int_concept(C,I), int_concept(D,I))

subsumed_tbox(T)(C,D): bool =
  subsumed_tbox(C,D,T)
```

Lema 2.2.10 *$C \sqsubseteq_{\emptyset} D$ syss $C \sqsubseteq D$.*

```
subsumed_tbox_empty: LEMMA
  empty?(T) IMPLIES
  (subsumed_tbox(T)(C,D) IFF concept_subsumed?(C,D))
```

Demostración: Directamente, usando 2.1.29, 2.2.18 y 2.2.9.

□

Definición 2.2.11 (Equivalencia relativa de conceptos)
*Los conceptos C y D son **equivalentes** respecto de $\mathcal{T}$ si, cada uno está subsumido en el otro respecto de $\mathcal{T}$. Se representa por $C \equiv_{\mathcal{T}} D$.*

```
equivalent_tbox(C,D,T): bool =
  subsumed_tbox(C,D,T) AND subsumed_tbox(D,C,T)

equivalent_tbox(T)(C,D): bool =
  equivalent_tbox(C,D,T)
```

Nota 2.2.12 El siguiente lema es un lema técnico para PVS.

Lema 2.2.13 *$C \equiv_{\mathcal{T}} D$ syss $C \sqsubseteq_{\mathcal{T}} D$ y $D \sqsubseteq_{\mathcal{T}} C$*

```
equivalent_tbox_rw: LEMMA
  equivalent_tbox(T)(C,D) IFF
  (subsumed_tbox(T)(C,D) AND subsumed_tbox(T)(D,C))
```

Demostración: Por las definiciones 2.2.11 y 2.2.9.

□

Lema 2.2.14 *$C \equiv_{\emptyset} D$ syss $C \equiv D$.*

```
equivalent_tbox_empty: LEMMA
  empty?(T) IMPLIES
  (equivalent_tbox(T)(C,D) IFF concept_equivalent?(C,D))
```

Demostración: Por 2.2.13, 2.1.30 y 2.2.10.

□

Definición 2.2.15 (Satisfacibilidad de axiomas asertivos)
Sea $\mathcal{I}$ una interpretación.

- *$\mathcal{I}$ **satisface** el axioma $x : C$ si $x^{\mathcal{I}} \in C^{\mathcal{I}}$.*
- *$\mathcal{I}$ **satisface** el axioma $(x, y) : R$ si $(x^{\mathcal{I}}, y^{\mathcal{I}}) \in R^{\mathcal{I}}$.*
- *$\mathcal{I}$ **satisface** el axioma $x \neq y$ si $x^{\mathcal{I}} \neq y^{\mathcal{I}}$.*

```
satisfies_aax(I,Aa): bool =
  CASES Aa OF
    instanceof(ni,C)     : member(int_names_ind(I)(ni),int_concept(C,I)),
    related(ni1,R,ni2)   : int_names_roles(I)(R)(int_names_ind(I)(ni1),
                                                 int_names_ind(I)(ni2)),
    different_ni(ni1,ni2): int_names_ind(I)(ni1) /= int_names_ind(I)(ni2)
  ENDCASES
```

Definición 2.2.16 (Modelo de una caja–A)
*La interpretación $\mathcal{I}$ es **modelo** de una caja–A $\mathcal{A}$ si satisface todos sus axiomas. Se representa por $\mathcal{I} \models \mathcal{A}$.*

```
is_model_ABox(I,AB): bool =
  FORALL Aa: member(Aa,AB) IMPLIES satisfies_aax(I,Aa)
```

Definición 2.2.17 (Satisfacibilidad de las cajas–A)
*Una caja–A es **satisfacible** si tiene algún modelo.*

```
abox_satisfiable(AB:ABox): bool =
  EXISTS I: is_model_ABox(I,AB)
```

Definición 2.2.18 (Modelo de una base de conocimiento)
*La interpretación $\mathcal{I}$ es **modelo** de la base de conocimiento $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ si es modelo de $\mathcal{T}$ y de $\mathcal{A}$. Se representa por $\mathcal{I} \models \mathcal{K}$.*

```
is_model_kb(I,KB): bool =
  LET (T,AB) = KB IN
    is_model_TBox(I,T) AND is_model_ABox(I,AB)
```

Definición 2.2.19 (Consistencia de las bases de conocimiento)
*Una base de conocimiento es **consistente** si tiene algún modelo e **inconsistente** en caso contrario.*

```
kb_consistent(KB): bool =
  EXISTS I: is_model_kb(I,KB)

kb_inconsistent(KB): bool =
  NOT kb_consistent(KB)
```

Nota 2.2.20 Con los siguientes resultados se muestra cómo se reducen unos problemas de razonamiento a otros.

Teorema 2.2.21 (Reducción de insatisfacibilidad a subsunción) *Un concepto C es insatisfacible respecto de una caja-T $\mathcal{T}$ syss $C \sqsubseteq_{\mathcal{T}} \bot$.*

```
subsumption_to_satisfiability: THEOREM
  unsatisfiable_tbox(T)(C) IFF subsumed_tbox(T)(C,alc_bottom)
```

Demostración: Por 2.2.7, 2.2.9 y 2.1.15.

□

Teorema 2.2.22 (Reducción de subsunción a insatisfacibilidad) *$C \sqsubseteq_{\mathcal{T}} D$ syss $C \cap \neg D$ es insatisfacible respecto de $\mathcal{T}$.*

```
satisfiability_to_subsumtion: THEOREM
  subsumed_tbox(T)(C,D) IFF
  unsatisfiable_tbox(T)(alc_and(C,alc_not(D)))
```

Demostración: Por las definiciones 2.2.9 y 2.2.7 y los lemas 2.1.16, 2.1.17 y 2.1.18.

□

Nota 2.2.23 Los siguientes resultados son auxiliares para la reduccción de la satisfacibilidad a la consistencia de bases de conocimiento (teorema 2.2.28 de la página 26).

Definición 2.2.24 ($\mathcal{I}_u$)
Dada una interpretación $\mathcal{I}$ y un elemento $u \in \Delta^{\mathcal{I}}$, la interpretación $\mathcal{I}_u$ se obtiene a partir de $\mathcal{I}$ interpretando todos los individuos en u.

```
interpretation_all_ni_same(I,(u: (int_domain(I)))): interpretation =
  (# int_domain        := int_domain(I),
     int_names_concept := int_names_concept(I),
     int_names_roles   := int_names_roles(I),
     int_names_ind     := (lambda(x):u) #)
```

Lema 2.2.25 *Si $u \in \Delta^{\mathcal{I}}$, entonces $C^{\mathcal{I}_u} = C^{\mathcal{I}}$.*

```
int_concept_interpretation_all_ni_same: LEMMA
  member(u, int_domain(I)) IMPLIES
  int_concept(C,interpretation_all_ni_same(I,u)) = int_concept(C,I)
```

Demostración: Por inducción en C.

□

Lema 2.2.26 *Si $u \in \Delta^{\mathcal{I}}$, entonces $\mathcal{I}_u$ satisface todos los axiomas terminológicos válidos en $\mathcal{I}$.*

```
interpretation_all_ni_same_preserve_satisfies_gtax: LEMMA
  member(u, int_domain(I)) IMPLIES
  (satisfies_gtax(I,Ax) IMPLIES
   satisfies_gtax(interpretation_all_ni_same(I,u),Ax))
```

Demostración: Por 2.2.4 y 2.2.25.

□

Lema 2.2.27 *Si $\mathcal{I}$ es modelo de la caja $\mathcal{T}$ y $u \in \Delta^{\mathcal{I}}$, entonces $\mathcal{I}_u$ es modelo de $\mathcal{T}$.*

```
interpretation_all_ni_same_preserve_model_tbox: LEMMA
  member(u, int_domain(I)) IMPLIES
  (is_model_TBox(I,T)
   IMPLIES
   is_model_TBox(interpretation_all_ni_same(I,u),T))
```

Demostración: Por la definición 2.2.18 y el lema 2.2.26.

□

Teorema 2.2.28 (Reduccción de la satisfacibilidad a la consistencia de bases de conocimiento) *El concepto C es satisfacible con respecto a la caja-T $\mathcal{T}$ syss la base de conocimiento $\mathcal{K} = (\mathcal{T}, \{x : C\})$ es consistente.*

```
satisfiable_tbox_iff_kb_consistent: THEOREM
  satisfiable_tbox(T)(C) IFF kb_consistent(T, singleton(instanceof(x,C)))
```

Demostración:

($\Longrightarrow$) Si C es satisfacible respecto de $\mathcal{T}$, existe un modelo $\mathcal{I}$ de $\mathcal{T}$, tal que $C^{\mathcal{I}} \neq \emptyset$. Sea $u \in C^{\mathcal{I}}$. Entonces, $\mathcal{I}_u$ es un modelo de $(\mathcal{T}, \{x : C\})$ (por el lema 2.2.27).

($\Longleftarrow$) Si $(\mathcal{T}, \{x : C\})$ es consistente, entonces tiene un modelo, $\mathcal{I}$. Basta comprobar que $\mathcal{I}$ es modelo de C respecto de $\mathcal{T}$.

□

Definición 2.2.29 (Instancia relativa)
*El nombre de individuo x es una **instancia** del concepto C con respecto de la base de conocimiento $\mathcal{K}$ si para todo modelo $\mathcal{I}$ de $\mathcal{K}$, $x^{\mathcal{I}} \in C^{\mathcal{I}}$.*

```
instance_of(KB)(x,C): bool =
  FORALL I: is_model_kb(I,KB) IMPLIES
            member(int_names_ind(I)(x), int_concept(C,I))
```

Teorema 2.2.30 (Reducción de instancia relativa a consistencia) *x es una instancia de C con respecto a $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ syss la base de conocimiento $(\mathcal{T}, \mathcal{A} \cup \{x : \neg C\})$ es inconsistente.*

```
instance_of_iff_kb_inconsistent: THEOREM
  instance_of(KB)(x,C) IFF
  kb_inconsistent((proj_1(KB), add(instanceof(x,alc_not(C)),proj_2(KB))))
```

Demostración:

($\Longrightarrow$) Sea x una instancia de C con respecto a $\mathcal{K}$. Veamos que $(\mathcal{T}, \mathcal{A} \cup \{x : \neg C\})$ es inconsistente. En efecto, supongamos que

$$\mathcal{I} \models (\mathcal{T}, \mathcal{A} \cup \{x : \neg C\}) \tag{2.1}$$

De 2.1 se sigue que $\mathcal{I} \models \mathcal{K}$ y, por la hipótesis, $x^{\mathcal{I}} \in C^{\mathcal{I}}$. De 2.1 también se sigue que $\mathcal{I} \models \mathcal{A} \cup \{x : \neg C\}$ y, por tanto, $x^{\mathcal{I}} \notin C^{\mathcal{I}}$ que contradice lo anterior.

($\Longleftarrow$) Sea $\mathcal{I} \models \mathcal{K} = (\mathcal{T}, \mathcal{A})$. Como $(\mathcal{T}, \mathcal{A} \cup \{x : \neg C\})$ es inconsistente, $\mathcal{I} \not\models x : \neg C$ y, por tanto, $\mathcal{I} \models x : C$

□

Nota 2.2.31 Para determinar la satisfacibilidad de un concepto a partir de la satisfacibilidad de una caja–A, consideraremos un individuo inicial x_0.

Lema 2.2.32 *Si dos interpretaciones coinciden en su dominio y el la forma de interpretar los nombres de concepto y los nombres de roles, entonces también coinciden en la interpretación de los conceptos.*

```
same_int_concept_cs: LEMMA
 FORALL (I,J:interpretation):
  int_domain(I) = int_domain(J) AND
  int_names_concept(I) = int_names_concept(J) AND
  int_names_roles(I) = int_names_roles(J)
  IMPLIES
  FORALL C: int_concept(C,I) = int_concept(C,J)
```

Teorema 2.2.33 (Reducción de la satisfacibilidad de un concepto a la satisfacibilidad de una caja–A) *Un concepto C es satisfacible si la caja–A $\mathcal{A} = \{x_0 : C\}$ es satisfacible.*

```
concept_satisfiable_equiv_abox_satisfiable: LEMMA
  concept_satisfiable?(C)
  IFF
  abox_satisfiable(singleton(instanceof(x_0,C)))
```

```
concept_satisfiable_equiv_abox_satisfiable_all: LEMMA
  concept_satisfiable?(C)
  IFF
  FORALL x: abox_satisfiable(singleton(instanceof(x,C)))
```

```
END kb_semantic
```

Apéndice A

Teorías PVS de la formalización de $\mathcal{ALC}$

A.1. Sintaxis de $\mathcal{ALC}$

A.1.1. Definición del tipo abstracto de datos de conceptos de $\mathcal{ALC}$

```
alc_concept[NC: TYPE+, NR: TYPE]: DATATYPE
 BEGIN
  alc_a(name: NC): alc_atomic?
  alc_not(conc: alc_concept): alc_not?
  alc_and(conc1, conc2: alc_concept): alc_and?
  alc_or(conc1, conc2: alc_concept): alc_or?
  alc_all(role: NR, conc: alc_concept): alc_all?
  alc_some(role: NR, conc: alc_concept): alc_some?
 END alc_concept
```

A.1.2. Definiciones sobre la sintaxis de $\mathcal{ALC}$

```
alc_concept_defs[NC: TYPE+, NR: TYPE]: THEORY
 BEGIN
  ASSUMING
   NC_TCC1: ASSUMPTION EXISTS (x: NC): TRUE;
  ENDASSUMING

  IMPORTING alc_concept_adt[NC, NR]

  A, B: VAR NC

  C, D, C1, C2: VAR alc_concept

  A1: NC

  alc_top: alc_concept = alc_or(alc_a(A1), alc_not(alc_a(A1)))
```

```
alc_bottom: alc_concept = alc_and(alc_a(A1), alc_not(alc_a(A1)))

alc_impl(C, D): alc_concept = alc_or(alc_not(C), D)

alc_equiv(C, D): alc_concept = alc_and(alc_impl(C, D), alc_impl(D, C))

% Termination TCC generated (at line 94, column 24) for  size(D)
size_TCC1: OBLIGATION
  FORALL (D: alc_concept[NC, NR], C: alc_concept[NC, NR]):
    C = alc_not(D) IMPLIES «[NC, NR](D, C);

% Termination TCC generated (at line 95, column 24) for  size(C1)
size_TCC2: OBLIGATION
  FORALL (C1, C2: alc_concept[NC, NR], C: alc_concept[NC, NR]):
    C = alc_and(C1, C2) IMPLIES «[NC, NR](C1, C);

% Termination TCC generated (at line 95, column 35) for  size(C2)
size_TCC3: OBLIGATION
  FORALL (C1, C2: alc_concept[NC, NR], C: alc_concept[NC, NR]):
    C = alc_and(C1, C2) IMPLIES «[NC, NR](C2, C);

% Termination TCC generated (at line 96, column 24) for  size(C1)
size_TCC4: OBLIGATION
  FORALL (C1, C2: alc_concept[NC, NR], C: alc_concept[NC, NR]):
    C = alc_or(C1, C2) IMPLIES «[NC, NR](C1, C);

% Termination TCC generated (at line 96, column 35) for  size(C2)
size_TCC5: OBLIGATION
  FORALL (C1, C2: alc_concept[NC, NR], C: alc_concept[NC, NR]):
    C = alc_or(C1, C2) IMPLIES «[NC, NR](C2, C);

% Termination TCC generated (at line 97, column 24) for  size(D)
size_TCC6: OBLIGATION
  FORALL (D: alc_concept[NC, NR], R: NR, C: alc_concept[NC, NR]):
    C = alc_all(R, D) IMPLIES «[NC, NR](D, C);

% Termination TCC generated (at line 98, column 24) for  size(D)
size_TCC7: OBLIGATION
  FORALL (D: alc_concept[NC, NR], R: NR, C: alc_concept[NC, NR]):
    C = alc_some(R, D) IMPLIES «[NC, NR](D, C);

size(C): RECURSIVE nat =
  CASES C
```

```
      OF alc_a(A): 1,
         alc_not(D): 1 + size(D),
         alc_and(C1, C2): 1 + size(C1) + size(C2),
         alc_or(C1, C2): 1 + size(C1) + size(C2),
         alc_all(R, D): 2 + size(D),
         alc_some(R, D): 2 + size(D)
      ENDCASES
     MEASURE C BY <<

  occur_in(A, C): RECURSIVE bool =
    CASES C
      OF alc_a(B): A = B,
         alc_not(D): occur_in(A, D),
         alc_and(C1, C2): occur_in(A, C1) OR occur_in(A, C2),
         alc_or(C1, C2): occur_in(A, C1) OR occur_in(A, C2),
         alc_all(R, D): occur_in(A, D),
         alc_some(R, D): occur_in(A, D)
      ENDCASES
     MEASURE C BY <<
 END alc_concept_defs
```

A.1.3. Conocimiento terminológico y asertivo

```
kb_syntax[NC: TYPE+, NR: TYPE, NI: TYPE]: THEORY
 BEGIN
  ASSUMING
   NC_TCC1: ASSUMPTION EXISTS (x: NC): TRUE;
  ENDASSUMING

  IMPORTING alc_concept_defs[NC, NR]

  A, B: VAR NC

  alc_gtax: DATATYPE
   BEGIN
    gci(antecedent, consequent: alc_concept): gci?
    concept_eq(antecedent, consequent: alc_concept): concept_eq?
   END alc_gtax

  alc_gtax: TYPE

  gci?, concept_eq?: [alc_gtax -> boolean]

  gci: [[alc_concept, alc_concept] -> (gci?)]
```

```
concept_eq: [[alc_concept, alc_concept] -> (concept_eq?)]

antecedent: [alc_gtax -> alc_concept[NC, NR]]

consequent: [alc_gtax -> alc_concept[NC, NR]]

alc_gtax_ord: [alc_gtax -> upto(1)]

alc_gtax_ord_defaxiom: AXIOM
  (FORALL (antecedent: alc_concept, consequent: alc_concept):
     alc_gtax_ord(gci(antecedent, consequent)) = 0)
   AND
   (FORALL (antecedent: alc_concept, consequent: alc_concept):
      alc_gtax_ord(concept_eq(antecedent, consequent)) = 1);

ord(x: alc_gtax): upto(1) =
    CASES x
      OF gci(gci1_var, gci2_var): 0,
         concept_eq(concept_eq1_var, concept_eq2_var): 1
      ENDCASES

alc_gtax_gci_extensionality: AXIOM
  FORALL (gci?_var: (gci?), gci?_var2: (gci?)):
    antecedent(gci?_var) = antecedent(gci?_var2) AND
     consequent(gci?_var) = consequent(gci?_var2)
     IMPLIES gci?_var = gci?_var2;

alc_gtax_gci_eta: AXIOM
  FORALL (gci?_var: (gci?)):
    gci(antecedent(gci?_var), consequent(gci?_var)) = gci?_var;

alc_gtax_concept_eq_extensionality: AXIOM
  FORALL (concept_eq?_var: (concept_eq?),
          concept_eq?_var2: (concept_eq?)):
    antecedent(concept_eq?_var) = antecedent(concept_eq?_var2) AND
     consequent(concept_eq?_var) = consequent(concept_eq?_var2)
     IMPLIES concept_eq?_var = concept_eq?_var2;

alc_gtax_concept_eq_eta: AXIOM
  FORALL (concept_eq?_var: (concept_eq?)):
    concept_eq(antecedent(concept_eq?_var), consequent(concept_eq?_var))
     = concept_eq?_var;
```

```
alc_gtax_antecedent_gci: AXIOM
  FORALL (gci1_var: alc_concept, gci2_var: alc_concept):
    antecedent(gci(gci1_var, gci2_var)) = gci1_var;

alc_gtax_consequent_gci: AXIOM
  FORALL (gci1_var: alc_concept, gci2_var: alc_concept):
    consequent(gci(gci1_var, gci2_var)) = gci2_var;

alc_gtax_antecedent_concept_eq: AXIOM
  FORALL (concept_eq1_var: alc_concept, concept_eq2_var: alc_concept):
    antecedent(concept_eq(concept_eq1_var, concept_eq2_var)) =
     concept_eq1_var;

alc_gtax_consequent_concept_eq: AXIOM
  FORALL (concept_eq1_var: alc_concept, concept_eq2_var: alc_concept):
    consequent(concept_eq(concept_eq1_var, concept_eq2_var)) =
     concept_eq2_var;

alc_gtax_inclusive: AXIOM
  FORALL (alc_gtax_var: alc_gtax):
    gci?(alc_gtax_var) OR concept_eq?(alc_gtax_var);

alc_gtax_induction: AXIOM
  FORALL (p: [alc_gtax -> boolean]):
    ((FORALL (gci1_var: alc_concept, gci2_var: alc_concept):
        p(gci(gci1_var, gci2_var)))
      AND
      (FORALL (concept_eq1_var: alc_concept,
               concept_eq2_var: alc_concept):
         p(concept_eq(concept_eq1_var, concept_eq2_var))))
     IMPLIES (FORALL (alc_gtax_var: alc_gtax): p(alc_gtax_var));

subterm(x, y: alc_gtax):  boolean = x = y;

«:  (well_founded?[alc_gtax]) = LAMBDA (x, y: alc_gtax): FALSE;

alc_gtax_well_founded: AXIOM well_founded?[alc_gtax](«);

reduce_nat(gci?_fun,
           concept_eq?_fun:
             [[alc_concept[NC, NR], alc_concept[NC, NR]] -> nat]):
      [alc_gtax -> nat] =
```

```
    LAMBDA (alc_gtax_adtvar: alc_gtax):
      LET red: [alc_gtax -> nat] = reduce_nat(gci?_fun, concept_eq?_fun)
        IN
        CASES alc_gtax_adtvar
          OF gci(gci1_var, gci2_var): gci?_fun(gci1_var, gci2_var),
             concept_eq(concept_eq1_var, concept_eq2_var):
               concept_eq?_fun(concept_eq1_var, concept_eq2_var)
          ENDCASES;

REDUCE_nat(gci?_fun,
           concept_eq?_fun:
             [[alc_concept[NC, NR], alc_concept[NC, NR], alc_gtax] ->
                nat]):
      [alc_gtax -> nat] =
    LAMBDA (alc_gtax_adtvar: alc_gtax):
      LET red: [alc_gtax -> nat] = REDUCE_nat(gci?_fun, concept_eq?_fun)
        IN
        CASES alc_gtax_adtvar
          OF gci(gci1_var, gci2_var):
               gci?_fun(gci1_var, gci2_var, alc_gtax_adtvar),
             concept_eq(concept_eq1_var, concept_eq2_var):
               concept_eq?_fun(concept_eq1_var, concept_eq2_var,
                               alc_gtax_adtvar)
          ENDCASES;

reduce_ordinal(gci?_fun,
               concept_eq?_fun:
                 [[alc_concept[NC, NR], alc_concept[NC, NR]] ->
                    ordinal]):
      [alc_gtax -> ordinal] =
    LAMBDA (alc_gtax_adtvar: alc_gtax):
      LET red: [alc_gtax -> ordinal] =
            reduce_ordinal(gci?_fun, concept_eq?_fun)
        IN
        CASES alc_gtax_adtvar
          OF gci(gci1_var, gci2_var): gci?_fun(gci1_var, gci2_var),
             concept_eq(concept_eq1_var, concept_eq2_var):
               concept_eq?_fun(concept_eq1_var, concept_eq2_var)
          ENDCASES;

REDUCE_ordinal(gci?_fun,
               concept_eq?_fun:
                 [[alc_concept[NC, NR], alc_concept[NC, NR], alc_gtax] ->
```

```
                    ordinal]):
      [alc_gtax -> ordinal] =
    LAMBDA (alc_gtax_adtvar: alc_gtax):
      LET red: [alc_gtax -> ordinal] =
            REDUCE_ordinal(gci?_fun, concept_eq?_fun)
        IN
        CASES alc_gtax_adtvar
          OF gci(gci1_var, gci2_var):
               gci?_fun(gci1_var, gci2_var, alc_gtax_adtvar),
             concept_eq(concept_eq1_var, concept_eq2_var):
               concept_eq?_fun(concept_eq1_var, concept_eq2_var,
                               alc_gtax_adtvar)
          ENDCASES;

Ax, Ax1, Ax2: VAR alc_gtax

specialization?(Ax): bool = gci?(Ax) AND (alc_atomic?(antecedent(Ax)))

definition?(Ax): bool = concept_eq?(Ax) AND (alc_atomic?(antecedent(Ax)))

TBox: TYPE = finite_set[alc_gtax]

T, T1, T2: VAR TBox

all_antecedents_atomics(T): bool =
    FORALL Ax: member(Ax, T) IMPLIES alc_atomic?(antecedent(Ax))

% Subtype TCC generated (at line 180, column 4) for
  % {C: alc_concept |
  %    EXISTS Ax:
  %      (member(Ax, T) AND
  %        alc_atomic?(antecedent(Ax)) AND C = antecedent(Ax))}
  % expected type  finite_set[alc_concept]
set_atomic_antecedents_TCC1: OBLIGATION
  FORALL (T: TBox):
    is_finite[alc_concept[NC, NR]]
        ({C: alc_concept[NC, NR] |
            EXISTS Ax:
              (member[alc_gtax](Ax, T) AND
                alc_atomic?[NC, NR](antecedent(Ax)) AND
                 C = antecedent(Ax))});

set_atomic_antecedents(T): finite_set[alc_concept] =
```

```
    {C: alc_concept |
       EXISTS Ax:
         (member(Ax, T) AND
           alc_atomic?(antecedent(Ax)) AND C = antecedent(Ax))}

antecedents_atomic_once(T): bool =
    all_antecedents_atomics(T) AND
     card(set_atomic_antecedents(T)) = card(T)

directely_use_tbox(T)(A, B): bool =
    EXISTS Ax:
      member(Ax, T) AND
       alc_atomic?(antecedent(Ax)) AND
        name(antecedent(Ax)) = A AND occur_in(B, consequent(Ax))

CLOUSURE_TR: LIBRARY = "../auxiliares/wf"

IMPORTING CLOUSURE_TR@cl_tr[NC]

use_tbox(T): pred[[NC, NC]] = tr_cl(directely_use_tbox(T))

acyclic(T): bool = NOT (EXISTS A: use_tbox(T)(A, A))

simple(T): bool = antecedents_atomic_once(T) AND acyclic(T)

assertional_ax: DATATYPE
 BEGIN
  instanceof(left: NI, right: alc_concept): instanceof?
  related(left: NI, role: NR, right: NI): related?
  different_ni(left: NI, right: NI): different_ni?
 END assertional_ax

assertional_ax: TYPE

instanceof?, related?, different_ni?: [assertional_ax -> boolean]

instanceof: [[NI, alc_concept] -> (instanceof?)]

related: [[NI, NR, NI] -> (related?)]

different_ni: [[NI, NI] -> (different_ni?)]

left: [assertional_ax -> NI]
```

```
right: [(instanceof?) -> alc_concept[NC, NR]]

role: [(related?) -> NR]

right: [{x: assertional_ax | related?(x) OR different_ni?(x)} -> NI]

JUDGEMENT (related?) SUBTYPE_OF
    {x: assertional_ax | related?(x) OR different_ni?(x)}

JUDGEMENT (different_ni?) SUBTYPE_OF
    {x: assertional_ax | related?(x) OR different_ni?(x)}

assertional_ax_ord: [assertional_ax -> upto(2)]

assertional_ax_ord_defaxiom: AXIOM
  (FORALL (left: NI, right: alc_concept):
     assertional_ax_ord(instanceof(left, right)) = 0)
   AND
   (FORALL (left: NI, role: NR, right: NI):
      assertional_ax_ord(related(left, role, right)) = 1)
    AND
    (FORALL (left: NI, right: NI):
       assertional_ax_ord(different_ni(left, right)) = 2);

ord(x: assertional_ax): upto(2) =
    CASES x
      OF instanceof(instanceof1_var, instanceof2_var): 0,
         related(related1_var, related2_var, related3_var): 1,
         different_ni(different_ni1_var, different_ni2_var): 2
      ENDCASES

assertional_ax_instanceof_extensionality: AXIOM
  FORALL (instanceof?_var: (instanceof?),
          instanceof?_var2: (instanceof?)):
    left(instanceof?_var) = left(instanceof?_var2) AND
     right(instanceof?_var) = right(instanceof?_var2)
     IMPLIES instanceof?_var = instanceof?_var2;

assertional_ax_instanceof_eta: AXIOM
  FORALL (instanceof?_var: (instanceof?)):
    instanceof(left(instanceof?_var), right(instanceof?_var)) =
     instanceof?_var;
```

```
assertional_ax_related_extensionality: AXIOM
  FORALL (related?_var: (related?), related?_var2: (related?)):
    left(related?_var) = left(related?_var2) AND
     role(related?_var) = role(related?_var2) AND
      right(related?_var) = right(related?_var2)
     IMPLIES related?_var = related?_var2;

assertional_ax_related_eta: AXIOM
  FORALL (related?_var: (related?)):
    related(left(related?_var), role(related?_var), right(related?_var))
     = related?_var;

assertional_ax_different_ni_extensionality: AXIOM
  FORALL (different_ni?_var: (different_ni?),
          different_ni?_var2: (different_ni?)):
    left(different_ni?_var) = left(different_ni?_var2) AND
     right(different_ni?_var) = right(different_ni?_var2)
     IMPLIES different_ni?_var = different_ni?_var2;

assertional_ax_different_ni_eta: AXIOM
  FORALL (different_ni?_var: (different_ni?)):
    different_ni(left(different_ni?_var), right(different_ni?_var)) =
     different_ni?_var;

assertional_ax_left_instanceof: AXIOM
  FORALL (instanceof1_var: NI, instanceof2_var: alc_concept):
    left(instanceof(instanceof1_var, instanceof2_var)) = instanceof1_var;

assertional_ax_right_instanceof: AXIOM
  FORALL (instanceof1_var: NI, instanceof2_var: alc_concept):
    right(instanceof(instanceof1_var, instanceof2_var)) = instanceof2_var;

assertional_ax_left_related: AXIOM
  FORALL (related1_var: NI, related2_var: NR, related3_var: NI):
    left(related(related1_var, related2_var, related3_var)) =
     related1_var;

assertional_ax_role_related: AXIOM
  FORALL (related1_var: NI, related2_var: NR, related3_var: NI):
    role(related(related1_var, related2_var, related3_var)) =
     related2_var;
```

```
assertional_ax_right_related: AXIOM
  FORALL (related1_var: NI, related2_var: NR, related3_var: NI):
    right(related(related1_var, related2_var, related3_var)) =
     related3_var;

assertional_ax_left_different_ni: AXIOM
  FORALL (different_ni1_var: NI, different_ni2_var: NI):
    left(different_ni(different_ni1_var, different_ni2_var)) =
     different_ni1_var;

assertional_ax_right_different_ni: AXIOM
  FORALL (different_ni1_var: NI, different_ni2_var: NI):
    right(different_ni(different_ni1_var, different_ni2_var)) =
     different_ni2_var;

assertional_ax_inclusive: AXIOM
  FORALL (assertional_ax_var: assertional_ax):
    instanceof?(assertional_ax_var) OR
     related?(assertional_ax_var) OR different_ni?(assertional_ax_var);

assertional_ax_induction: AXIOM
  FORALL (p: [assertional_ax -> boolean]):
    ((FORALL (instanceof1_var: NI, instanceof2_var: alc_concept):
        p(instanceof(instanceof1_var, instanceof2_var)))
      AND
      (FORALL (related1_var: NI, related2_var: NR, related3_var: NI):
         p(related(related1_var, related2_var, related3_var)))
       AND
       (FORALL (different_ni1_var: NI, different_ni2_var: NI):
          p(different_ni(different_ni1_var, different_ni2_var))))
     IMPLIES
     (FORALL (assertional_ax_var: assertional_ax): p(assertional_ax_var));

subterm(x, y: assertional_ax):  boolean = x = y;

«:  (well_founded?[assertional_ax]) =
    LAMBDA (x, y: assertional_ax): FALSE;

assertional_ax_well_founded: AXIOM well_founded?[assertional_ax](«);

reduce_nat(instanceof?_fun: [[NI, alc_concept[NC, NR]] -> nat],
           related?_fun: [[NI, NR, NI] -> nat],
           different_ni?_fun: [[NI, NI] -> nat]):
```

```
      [assertional_ax -> nat] =
    LAMBDA (assertional_ax_adtvar: assertional_ax):
      LET red: [assertional_ax -> nat] =
            reduce_nat(instanceof?_fun, related?_fun, different_ni?_fun)
        IN
        CASES assertional_ax_adtvar
          OF instanceof(instanceof1_var, instanceof2_var):
               instanceof?_fun(instanceof1_var, instanceof2_var),
             related(related1_var, related2_var, related3_var):
               related?_fun(related1_var, related2_var, related3_var),
             different_ni(different_ni1_var, different_ni2_var):
               different_ni?_fun(different_ni1_var, different_ni2_var)
          ENDCASES;

REDUCE_nat(instanceof?_fun:
             [[NI, alc_concept[NC, NR], assertional_ax] -> nat],
           related?_fun: [[NI, NR, NI, assertional_ax] -> nat],
           different_ni?_fun: [[NI, NI, assertional_ax] -> nat]):
      [assertional_ax -> nat] =
    LAMBDA (assertional_ax_adtvar: assertional_ax):
      LET red: [assertional_ax -> nat] =
            REDUCE_nat(instanceof?_fun, related?_fun, different_ni?_fun)
        IN
        CASES assertional_ax_adtvar
          OF instanceof(instanceof1_var, instanceof2_var):
               instanceof?_fun(instanceof1_var, instanceof2_var,
                               assertional_ax_adtvar),
             related(related1_var, related2_var, related3_var):
               related?_fun(related1_var, related2_var, related3_var,
                            assertional_ax_adtvar),
             different_ni(different_ni1_var, different_ni2_var):
               different_ni?_fun(different_ni1_var, different_ni2_var,
                                 assertional_ax_adtvar)
          ENDCASES;

reduce_ordinal(instanceof?_fun: [[NI, alc_concept[NC, NR]] -> ordinal],
               related?_fun: [[NI, NR, NI] -> ordinal],
               different_ni?_fun: [[NI, NI] -> ordinal]):
      [assertional_ax -> ordinal] =
    LAMBDA (assertional_ax_adtvar: assertional_ax):
      LET red: [assertional_ax -> ordinal] =
            reduce_ordinal(instanceof?_fun, related?_fun,
                           different_ni?_fun)
```

```
          IN
          CASES assertional_ax_adtvar
            OF instanceof(instanceof1_var, instanceof2_var):
                 instanceof?_fun(instanceof1_var, instanceof2_var),
               related(related1_var, related2_var, related3_var):
                 related?_fun(related1_var, related2_var, related3_var),
               different_ni(different_ni1_var, different_ni2_var):
                 different_ni?_fun(different_ni1_var, different_ni2_var)
            ENDCASES;

  REDUCE_ordinal(instanceof?_fun:
                   [[NI, alc_concept[NC, NR], assertional_ax] -> ordinal],
                 related?_fun: [[NI, NR, NI, assertional_ax] -> ordinal],
                 different_ni?_fun:
                   [[NI, NI, assertional_ax] -> ordinal]):
        [assertional_ax -> ordinal] =
      LAMBDA (assertional_ax_adtvar: assertional_ax):
        LET red: [assertional_ax -> ordinal] =
              REDUCE_ordinal(instanceof?_fun, related?_fun,
                             different_ni?_fun)
          IN
          CASES assertional_ax_adtvar
            OF instanceof(instanceof1_var, instanceof2_var):
                 instanceof?_fun(instanceof1_var, instanceof2_var,
                                 assertional_ax_adtvar),
               related(related1_var, related2_var, related3_var):
                 related?_fun(related1_var, related2_var, related3_var,
                              assertional_ax_adtvar),
               different_ni(different_ni1_var, different_ni2_var):
                 different_ni?_fun(different_ni1_var, different_ni2_var,
                                   assertional_ax_adtvar)
            ENDCASES;

  ABox: TYPE = finite_set[assertional_ax]

  Aa, Aa1, Aa2: VAR assertional_ax

  KnowledgeBase: TYPE = [TBox, ABox]

  KB: VAR KnowledgeBase
END kb_syntax
```

A.2. Semántica de $\mathcal{ALC}$

A.2.1. Semántica de los conceptos de $\mathcal{ALC}$

```
semantics_alc_concept[NC: TYPE+, NR: TYPE, NI: TYPE, U: TYPE+]: THEORY
 BEGIN
  ASSUMING
   NC_TCC1: ASSUMPTION EXISTS (x: NC): TRUE;

   U_TCC1: ASSUMPTION EXISTS (x: U): TRUE;
  ENDASSUMING

  IMPORTING alc_concept_defs[NC, NR]

  A, B: VAR NC

  C, D, C1, C2: VAR alc_concept

  a_1: U

  S_1: (nonempty?[U]) = singleton(a_1)

  % Subtype TCC generated (at line 63, column 19) for  emptyset[U]
    % expected type  (powerset(S_1))
  f_1_TCC1: OBLIGATION powerset[U](S_1)(emptyset[U]);

  f_1: [NC -> (powerset(S_1))] = LAMBDA (A: NC): emptyset[U]

  f_2: [NR -> PRED[[(S_1), (S_1)]]] =
      LAMBDA (R: NR):
        restrict[[U, U], [(S_1), (S_1)], boolean](emptyset[[U, U]])

  % Subtype TCC generated (at line 81, column 19) for  a_1
    % expected type  (S_1)
  f_3_TCC1: OBLIGATION FORALL (x: NI): S_1(a_1);

  f_3: [NI -> (S_1)] = LAMBDA (x: NI): a_1

  % Existence TCC generated (at line 95, column 2) for
    % interpretation: TYPE+ =
    %       [# int_domain: (nonempty?[U]),
    %          int_names_concept: [NC -> (powerset(int_domain))],
    %          int_names_roles: [NR -> PRED[[(int_domain), (int_domain)]]],
    %          int_names_ind: [NI -> (int_domain)] #]
```

```
interpretation_TCC1: OBLIGATION
  EXISTS (x:
            [# int_domain: (nonempty?[U]),
               int_names_concept: [NC -> (powerset[U](int_domain))],
               int_names_ind: [NI -> (int_domain)],
               int_names_roles:
                 [NR -> PRED[[(int_domain), (int_domain)]]] #]):
    TRUE;

interpretation: TYPE+ =
      [# int_domain: (nonempty?[U]),
         int_names_concept: [NC -> (powerset(int_domain))],
         int_names_roles: [NR -> PRED[[(int_domain), (int_domain)]]],
         int_names_ind: [NI -> (int_domain)] #]

I: VAR interpretation

% Termination TCC generated (at line 157, column 48) for  int_concept(D, I)
int_concept_TCC1: OBLIGATION
  FORALL (D: alc_concept[NC, NR], C: alc_concept[NC, NR]):
    C = alc_not(D) IMPLIES «[NC, NR](D, C);

% Termination TCC generated (at line 158, column 35) for  int_concept(C1, I)
int_concept_TCC2: OBLIGATION
  FORALL (C1, C2: alc_concept[NC, NR], C: alc_concept[NC, NR]):
    C = alc_and(C1, C2) IMPLIES «[NC, NR](C1, C);

% Termination TCC generated (at line 158, column 53) for  int_concept(C2, I)
int_concept_TCC3: OBLIGATION
  FORALL (C1, C2: alc_concept[NC, NR], C: alc_concept[NC, NR]):
    C = alc_and(C1, C2) IMPLIES «[NC, NR](C2, C);

% Termination TCC generated (at line 159, column 28) for  int_concept(C1, I)
int_concept_TCC4: OBLIGATION
  FORALL (C1, C2: alc_concept[NC, NR], C: alc_concept[NC, NR]):
    C = alc_or(C1, C2) IMPLIES «[NC, NR](C1, C);

% Termination TCC generated (at line 159, column 46) for  int_concept(C2, I)
int_concept_TCC5: OBLIGATION
  FORALL (C1, C2: alc_concept[NC, NR], C: alc_concept[NC, NR]):
    C = alc_or(C1, C2) IMPLIES «[NC, NR](C2, C);

% Termination TCC generated (at line 163, column 26) for  int_concept(D, I)
```

```
int_concept_TCC6: OBLIGATION
  FORALL (D: alc_concept[NC, NR], R: NR, C: alc_concept[NC, NR],
          I: interpretation):
    C = alc_all(R, D) IMPLIES
     (FORALL (a: (int_domain(I)), b: (int_domain(I))):
        int_names_roles(I)(R)(a, b) IMPLIES «[NC, NR](D, C));

% Termination TCC generated (at line 167, column 26) for  int_concept(D, I)
int_concept_TCC7: OBLIGATION
  FORALL (D: alc_concept[NC, NR], R: NR, C: alc_concept[NC, NR],
          I: interpretation):
    C = alc_some(R, D) IMPLIES
     (FORALL (a: (int_domain(I)), b: (int_domain(I))):
        int_names_roles(I)(R)(a, b) IMPLIES «[NC, NR](D, C));

int_concept(C, I): RECURSIVE set[U] =
  CASES C
    OF alc_a(B): int_names_concept(I)(B),
       alc_not(D): difference(int_domain(I), int_concept(D, I)),
       alc_and(C1, C2):
         intersection(int_concept(C1, I), int_concept(C2, I)),
       alc_or(C1, C2): union(int_concept(C1, I), int_concept(C2, I)),
       alc_all(R, D):
         extend[U, (int_domain(I)), bool, FALSE]
             ({a: (int_domain(I)) |
                 FORALL (b: (int_domain(I))):
                   int_names_roles(I)(R)(a, b) IMPLIES
                    int_concept(D, I)(b)}),
       alc_some(R, D):
         extend[U, (int_domain(I)), bool, FALSE]
             ({a: (int_domain(I)) |
                 EXISTS (b: (int_domain(I))):
                   int_names_roles(I)(R)(a, b) AND int_concept(D, I)(b)})
    ENDCASES
   MEASURE C BY «

int_concept_alc_bottom_emptyset: LEMMA
  int_concept(alc_bottom, I) = emptyset

int_concept_alc_and: LEMMA
  int_concept(alc_and(C, D), I) =
   intersection(int_concept(C, I), int_concept(D, I))
```

```
  int_concept_alc_not: LEMMA
    int_concept(alc_not(C), I) =
     difference(int_domain(I), int_concept(C, I))

  int_concept_alc_subset_int_domain: LEMMA
    subset?(int_concept(C, I), int_domain(I))

  member_int_concept_member_int_domain: COROLLARY
    FORALL (u: U):
      member(u, int_concept(C, I)) IMPLIES member(u, int_domain(I))

  is_model_concept(I, C): bool = nonempty?(int_concept(C, I));

  |=(I, C): bool = is_model_concept(I, C)

  concept_satisfiable?(C): bool = EXISTS I: I |= C

  concept_unsatisfiable?(C): bool = FORALL I: empty?(int_concept(C, I))

  unsatisfiable_IFF_NOT_satisfiable: LEMMA
    concept_unsatisfiable?(C) IFF NOT concept_satisfiable?(C)

  concept_tautological?(C): bool =
      FORALL I: int_concept(C, I) = int_domain(I)

  top_is_tautological: LEMMA concept_tautological?(alc_top)

  exists_tautological: COROLLARY EXISTS C: concept_tautological?(C)

  bottom_is_unsatisfiable: LEMMA concept_unsatisfiable?(alc_bottom)

  exists_unsatisfiable: COROLLARY EXISTS C: concept_unsatisfiable?(C)

  concept_subsumed?(C, D): bool =
      FORALL I: subset?(int_concept(C, I), int_concept(D, I))

  concept_equivalent?(C, D): bool =
      concept_subsumed?(C, D) AND concept_subsumed?(D, C)
 END semantics_alc_concept
```

A.2.2. Semántica de las bases de conocimiento

```
kb_semantic[NC: TYPE+, NR: TYPE, NI: TYPE, U: TYPE+]: THEORY
 BEGIN
```

```
ASSUMING
 NC_TCC1: ASSUMPTION EXISTS (x: NC): TRUE;

 U_TCC1: ASSUMPTION EXISTS (x: U): TRUE;
ENDASSUMING

IMPORTING kb_syntax[NC, NR, NI]

IMPORTING semantics_alc_concept[NC, NR, NI, U]

C, D, C1, C2: VAR alc_concept

I: VAR interpretation

u: VAR U

x: VAR NI

Ax, Ax1, Ax2: VAR alc_gtax

T, T1, T2: VAR TBox

Aa, Aa1, Aa2: VAR assertional_ax

AB: VAR ABox

KB: VAR KnowledgeBase

satisfies_gtax(I, Ax): bool =
    CASES Ax
      OF gci(C, D): subset?(int_concept(C, I), int_concept(D, I)),
         concept_eq(C, D): int_concept(C, I) = int_concept(D, I)
      ENDCASES

is_model_TBox(I, T): bool =
    FORALL Ax: member(Ax, T) IMPLIES satisfies_gtax(I, Ax)

tbox_consistent(T): bool = EXISTS I: is_model_TBox(I, T)

satisfiable_tbox(C, T): bool =
    EXISTS I: is_model_TBox(I, T) AND nonempty?(int_concept(C, I))

unsatisfiable_tbox(C, T): bool = NOT satisfiable_tbox(C, T)
```

```
satisfiable_tbox(T)(C): bool = satisfiable_tbox(C, T)

unsatisfiable_tbox(T)(C): bool = unsatisfiable_tbox(C, T)

satisfiable_tbox_empty: LEMMA
  empty?(T) IMPLIES (satisfiable_tbox(T)(C) IFF concept_satisfiable?(C))

subsumed_tbox(C, D, T): bool =
    FORALL I:
      is_model_TBox(I, T) IMPLIES
       subset?(int_concept(C, I), int_concept(D, I))

subsumed_tbox(T)(C, D): bool = subsumed_tbox(C, D, T)

subsumed_tbox_empty: LEMMA
  empty?(T) IMPLIES (subsumed_tbox(T)(C, D) IFF concept_subsumed?(C, D))

equivalent_tbox(C, D, T): bool =
    subsumed_tbox(C, D, T) AND subsumed_tbox(D, C, T)

equivalent_tbox(T)(C, D): bool = equivalent_tbox(C, D, T)

equivalent_tbox_rw: LEMMA
  equivalent_tbox(T)(C, D) IFF
   (subsumed_tbox(T)(C, D) AND subsumed_tbox(T)(D, C))

equivalent_tbox_empty: LEMMA
  empty?(T) IMPLIES
   (equivalent_tbox(T)(C, D) IFF concept_equivalent?(C, D))

satisfies_aax(I, Aa): bool =
    CASES Aa
      OF instanceof(ni, C):
           member(int_names_ind(I)(ni), int_concept(C, I)),
         related(ni1, R, ni2):
           int_names_roles(I)
               (R)(int_names_ind(I)(ni1), int_names_ind(I)(ni2)),
         different_ni(ni1, ni2):
           int_names_ind(I)(ni1) /= int_names_ind(I)(ni2)
      ENDCASES

is_model_ABox(I, AB): bool =
```

```
    FORALL Aa: member(Aa, AB) IMPLIES satisfies_aax(I, Aa)

abox_consistent(AB: ABox): bool = EXISTS I: is_model_ABox(I, AB)

is_model_kb(I, KB): bool =
    LET (T, AB) = KB IN is_model_TBox(I, T) AND is_model_ABox(I, AB)

kb_consistent(KB): bool = EXISTS I: is_model_kb(I, KB)

kb_inconsistent(KB): bool = NOT kb_consistent(KB)

subsumption_to_satisfiability: THEOREM
  unsatisfiable_tbox(T)(C) IFF subsumed_tbox(T)(C, alc_bottom)

satisfiability_to_subsumtion: THEOREM
  subsumed_tbox(T)(C, D) IFF unsatisfiable_tbox(T)(alc_and(C, alc_not(D)))

interpretation_all_ni_same(I, (u: (int_domain(I)))): interpretation =
    (# int_domain := int_domain(I),
       int_names_concept := int_names_concept(I),
       int_names_roles := int_names_roles(I),
       int_names_ind := (LAMBDA (x): u) #)

% Subtype TCC generated (at line 334, column 47) for  u
  % expected type  (int_domain(I))
int_concept_interpretation_all_ni_same_TCC1: OBLIGATION
  FORALL (I:
            [# int_domain: (nonempty?[U]),
               int_names_concept: [NC -> (powerset[U](int_domain))],
               int_names_ind: [NI -> (int_domain)],
               int_names_roles:
                 [NR -> PRED[[(int_domain), (int_domain)]]] #],
          u: U):
    member(u, int_domain(I)) IMPLIES int_domain(I)(u);

int_concept_interpretation_all_ni_same: LEMMA
  member(u, int_domain(I)) IMPLIES
   int_concept(C, interpretation_all_ni_same(I, u)) = int_concept(C, I)

interpretation_all_ni_same_preserve_satisfies_gtax: LEMMA
  member(u, int_domain(I)) IMPLIES
   (satisfies_gtax(I, Ax) IMPLIES
     satisfies_gtax(interpretation_all_ni_same(I, u), Ax))
```

```
  interpretation_all_ni_same_preserve_model_tbox: LEMMA
    member(u, int_domain(I)) IMPLIES
     (is_model_TBox(I, T) IMPLIES
       is_model_TBox(interpretation_all_ni_same(I, u), T))

  satisfiable_tbox_iff_kb_consistent: THEOREM
    satisfiable_tbox(T)(C) IFF kb_consistent(T, singleton(instanceof(x, C)))

  instance_of(KB)(x, C): bool =
      FORALL I:
        is_model_kb(I, KB) IMPLIES
         member(int_names_ind(I)(x), int_concept(C, I))

  instance_of_iff_kb_inconsistent: THEOREM
    instance_of(KB)(x, C) IFF
     kb_inconsistent(PROJ_1(KB), add(instanceof(x, alc_not(C)), PROJ_2(KB)))
END kb_semantic
```

Apéndice B

Guiones de pruebas de las teorías PVS de la formalización

B.1. Semántica de $\mathcal{ALC}$

B.1.1. Semántica de los conceptos de $\mathcal{ALC}$

```
Proof scripts for file semantics_alc_concept.pvs:


semantics_alc_concept.f_1_TCC1: unchecked [shostak](0.03 s)

("" (subtype-tcc))


semantics_alc_concept.f_3_TCC1: unchecked [shostak](0.01 s)

("" (subtype-tcc))


semantics_alc_concept.interpretation_TCC1: unchecked [shostak](0.01 s)

(""
 (inst +
  "(# int_domain:= S_1, int_names_concept:= f_1, int_names_roles:= f_2, int_names_ind:= f_


semantics_alc_concept.int_concept_TCC1: unchecked [shostak](0.05 s)

("" (termination-tcc))
```

```
semantics_alc_concept.int_concept_TCC2: unchecked [shostak](0.05 s)

("" (termination-tcc))


semantics_alc_concept.int_concept_TCC3: unchecked [shostak](0.04 s)

("" (termination-tcc))


semantics_alc_concept.int_concept_TCC4: unchecked [shostak](0.05 s)

("" (termination-tcc))


semantics_alc_concept.int_concept_TCC5: unchecked [shostak](0.04 s)

("" (termination-tcc))


semantics_alc_concept.int_concept_TCC6: unchecked [shostak](0.05 s)

("" (termination-tcc))


semantics_alc_concept.int_concept_TCC7: unchecked [shostak](0.06 s)

("" (termination-tcc))


semantics_alc_concept.int_concept_alc_bottom_emptyset: unchecked [shostak](0.62 s)

("" (skosimp) (apply-extensionality :hide? t) (grind))


semantics_alc_concept.int_concept_alc_and: unchecked [shostak](0.26 s)

("" (grind))


semantics_alc_concept.int_concept_alc_not: unchecked [shostak](0.19 s)

("" (grind))
```

```
semantics_alc_concept.int_concept_alc_subset_int_domain: unchecked [shostak](0.35 s)

(""
 (induct "C")
 (("1"
   (skosimp*)
   (expand "int_concept")
   (typepred "int_names_concept(I!1)(alc_a1_var!1)")
   (expand "powerset")
   (propax))
  ("2"
   (skosimp*)
   (inst?)
   (expand "int_concept" +)
   (expand "subset?")
   (skosimp)
   (inst?)
   (expand "member")
   (expand "difference")
   (expand "member")
   (prop))
  ("3"
   (skosimp*)
   (inst?)
   (inst?)
   (expand "int_concept" +)
   (use "intersection_subset1[U]")
   (lemma "subset_transitive[U]")
   (inst -
    "intersection(int_concept(alc_and1_var!1, I!1),
                                     int_concept(alc_and2_var!1, I!1))"
    "int_concept(alc_and1_var!1, I!1)" "int_domain(I!1)")
   (prop))
  ("4"
   (skosimp*)
   (inst?)
   (inst?)
   (expand "int_concept" +)
   (use "union_upper_bound[U]")
   (prop))
  ("5"
```

```
   (skosimp*)
   (inst?)
   (expand "int_concept" +)
   (expand "subset?")
   (skosimp)
   (inst?)
   (expand "member")
   (expand "extend")
   (prop))
  ("6"
   (skosimp*)
   (inst?)
   (expand "int_concept" +)
   (expand "subset?")
   (skosimp)
   (inst?)
   (expand "member")
   (expand "extend")
   (prop))))


semantics_alc_concept.member_int_concept_member_int_domain: proved - incomplete [shostak](

(""
 (skosimp)
 (use "int_concept_alc_subset_int_domain")
 (expand "subset?")
 (inst?)
 (prop))


semantics_alc_concept.unsatisfiable_IFF_NOT_satisfiable: unchecked [shostak](0.09 s)

("" (default-strategy))


semantics_alc_concept.top_is_tautological: unchecked [shostak](0.13 s)

(""
 (apply (then (expand "alc_top") (expand "concept_tautological?")
        (skosimp)))
 (apply-extensionality :hide? t)
 (grind)
```

```
 (typepred "int_names_concept(I!1)(A1)")
 (grind))


semantics_alc_concept.exists_tautological: unchecked [shostak](0.00 s)

("" (inst + "alc_top") (rewrite "top_is_tautological"))


semantics_alc_concept.bottom_is_unsatisfiable: unchecked [shostak](0.08 s)

("" (default-strategy))


semantics_alc_concept.exists_unsatisfiable: unchecked [shostak](0.01 s)

("" (inst + "alc_bottom") (rewrite "bottom_is_unsatisfiable"))
```

B.1.2. Semántica de las bases de conocimiento

```
Proof scripts for file kb_semantic.pvs:


kb_semantic.satisfiable_tbox_empty: unchecked [shostak](0.11 s)

(""
 (grind :defs nil :rewrites
  ("satisfiable_tbox"
   "concept_satisfiable?"
   "is_model_TBox"
   "|="
   "is_model_concept"
   "empty?")))


kb_semantic.subsumed_tbox_empty: unchecked [shostak](0.18 s)

(""
 (grind :defs nil :rewrites
  ("subsumed_tbox" "concept_subsumed?" "is_model_TBox" "empty?")))


kb_semantic.equivalent_tbox_rw: unchecked [shostak](0.03 s)
```

```
("" 
 (grind :defs nil :rewrites
  ("equivalent_tbox" "subsumed_tbox" "equivalent_tbox")))


kb_semantic.equivalent_tbox_empty: unchecked [shostak](0.05 s)

("" 
 (grind :defs nil :rewrites
  ("equivalent_tbox_rw" "concept_equivalent?" "subsumed_tbox_empty")))


kb_semantic.subsumption_to_satisfiability: unchecked [shostak](0.14 s)

("" 
 (grind :defs nil :rewrites
  ("unsatisfiable_tbox"
   "subsumed_tbox"
   "satisfiable_tbox"
   "int_concept_alc_bottom_emptyset"
   "nonempty?"
   "empty?"
   "subset?"
   "member"
   "emptyset")))


kb_semantic.satisfiability_to_subsumtion: unchecked [shostak](0.25 s)

("" 
 (grind-with-lemmas :defs nil :rewrites
  ("subsumed_tbox"
   "unsatisfiable_tbox"
   "satisfiable_tbox"
   "int_concept_alc_and"
   "int_concept_alc_not"
   "nonempty?"
   "empty?"
   "subset?"
   "intersection"
   "difference"
   "member")
  :lemmas "int_concept_alc_subset_int_domain"))
```

```
kb_semantic.int_concept_interpretation_all_ni_same_TCC1: unchecked [shostak](0.02 s)

("" (subtype-tcc))


kb_semantic.int_concept_interpretation_all_ni_same: unfinished [shostak](0.16 s)

(""
 (induct-and-simplify "C")
 (("1" (expand "extend") (grind)) ("2" (expand "extend") (grind))))


kb_semantic.interpretation_all_ni_same_preserve_satisfies_gtax: unchecked [shostak](0.64 s

(""
 (skosimp)
 (expand "satisfies_gtax")
 (lift-if)
 (grind :defs nil :rewrites "int_concept_interpretation_all_ni_same"))


kb_semantic.interpretation_all_ni_same_preserve_model_tbox: unchecked [shostak](0.19 s)

(""
 (grind :defs nil :rewrites
  ("is_model_TBox"
   "interpretation_all_ni_same_preserve_satisfies_gtax")))


kb_semantic.satisfiable_tbox_iff_kb_consistent: unchecked [shostak](0.16 s)

(""
 (skosimp)
 (prop)
 (("1"
   (apply (then (expand "satisfiable_tbox") (expand "satisfiable_tbox")
           (skosimp)))
   (apply (then (expand "nonempty?") (expand "empty?") (skosimp)))
   (expand "kb_consistent")
   (inst + "interpretation_all_ni_same(I!1,x!2)")
   (("1"
```

```
     (expand "is_model_kb")
     (prop)
     (("1"
       (rewrite "interpretation_all_ni_same_preserve_model_tbox")
       (forward-chain "member_int_concept_member_int_domain"))
      ("2"
       (expand "is_model_ABox")
       (skosimp)
       (expand "member" -1)
       (expand "singleton")
       (replace*)
       (expand "satisfies_aax")
       (expand "interpretation_all_ni_same" 1 1)
       (rewrite "int_concept_interpretation_all_ni_same")
       (forward-chain "member_int_concept_member_int_domain"))))
    ("2"
     (forward-chain "member_int_concept_member_int_domain")
     (expand "member")
     (propax))))
  ("2"
   (expand "kb_consistent")
   (skosimp)
   (expand "is_model_kb")
   (expand "satisfiable_tbox")
   (expand "satisfiable_tbox")
   (inst?)
   (prop)
   (expand "is_model_ABox")
   (inst?)
   (expand "member")
   (expand "singleton")
   (expand "satisfies_aax")
   (apply (then (expand "nonempty?") (expand "empty?") (inst?))))))


kb_semantic.instance_of_iff_kb_inconsistent: unchecked [shostak](0.23 s)

(""
 (grind :defs nil :rewrites
  ("instance_of" "kb_inconsistent" "kb_consistent" "is_model_kb"))
 (("1"
   (grind :defs nil :rewrites
    ("is_model_ABox"
```

```
    "member"
    "add"
    "satisfies_aax"
    "int_concept_alc_not"
    "difference")))
 ("2"
  (grind :defs nil :rewrites ("is_model_ABox" "member" "add")
   :polarity? t))
 ("3"
  (expand "is_model_ABox")
  (skosimp)
  (expand "member" -3)
  (expand "add")
  (prop)
  (("1"
    (replace -1 :dir rl)
    (grind :defs nil :rewrites
     ("satisfies_aax" "int_concept_alc_not" "difference" "member")))
   ("2" (inst?) (prop))))))
```

Bibliografía

[BMNPS03] F. Baader, D. McGuiness, D. Nardi, and P. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation and Applications*. Cambridge University Press, 2003.

[BN03] Franz Baader and Werner Nutt. *Basic Description Logics*, pages 43–95. 2003.

[Lut02] Carsten Lutz. *The Complexity of Description Logics with Concrete Domains*. PhD thesis, Rheinisch–Westfalischen Technischen Hochschule, 2002.

[Raca] *RacerPro Reference Manual (Version 1.8)*.

[Racb] *RacerPro User's Guide (Version 1.8)*.

[Tob01] S. Tobies. *Complexity results and practical algorithms for logics in knowledge representation*. PhD thesis, Univ. Aachen, 2001.