A Formally Verified Prover for the $\mathcal{ALC}$ Description Logic

Francisco-Jesus Martín-Mateos, and José-Luis Ruiz-Reina

Departamento 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
{jalonso,jborrego,mjoseh,fjesus,jruiz}@us.es

Abstract. The Ontology Web Language (OWL) is a language used for the Semantic Web. OWL is based on Description Logics (DLs), a family of logical formalisms for representing and reasoning about conceptual and terminological knowledge. Among these, the logic $\mathcal{ALC}$ is a ground DL used in many practical cases. Moreover, the Semantic Web appears as a new field for the application of formal methods, that could be used to increase its reliability. A starting point could be the formal verification of satisfiability provers for DLs. In this paper, we present the PVS specification of a prover for $\mathcal{ALC}$, as well as the proofs of its termination, soundness and completeness. We also present the formalization of the well–foundedness of the multiset relation induced by a well–founded relation. This result has been used to prove the termination and the completeness of the $\mathcal{ALC}$ prover.

1 Introduction

The goal of the presented work is the formal verification of satisfiability algorithms for description logics (DLs), as a previous stage to the formal verification of DLs reasoners. In particular, we describe in this paper a formal proof of the well–known tableau algorithm for the $\mathcal{ALC}$ description logic in the PVS verification system [19].

Description Logics [5] are a family of logics which can be used to represent terminological and conceptual knowledge. Among these, the ground logic is the $\mathcal{ALC}$ logic, introduced by Schmidt–Schauß and Smolka [24], who also developed a tableau–like algorithm for testing satisfiability in it. $\mathcal{ALC}$ is the base for more expressive logics as $\mathcal{SHOIQ}$, obtained extending $\mathcal{ALC}$ in several expressive ways. The importance of $\mathcal{SHOIQ}$ stems from the fact that it and its fragments are used for reasoning in the semantic web [4]. Specifically, the fragment $\mathcal{SHOIN}$ corresponds to the ontology language OWL–DL [13,12], which was recommended by the W3C as the standard web ontology language [6]. The fragment $\mathcal{SHIQ}$ is the concept language supported by systems as FaCT++ and RACER [25,10].

⋆ This research was partially funded by Spanish Ministry of Education and Science under grant TIN2004–03884 and Feder funds.

Many techniques have been proposed and investigated to obtain decision procedures for DLs reasoning. Among these, tableaux reasoning are the most successful approach so far. In fact, DLs reasoners such as FaCT++ and RACER are based on tableau algorithms.

We believe that the verification of reasoning systems for the SW poses a new challenge for the application of formal methods. In this field, our research group have carried out several works related to the formal verification of reasoning systems. Some examples are the verification of a generic framework for propositional logic reasoning algorithms [14], Knuth–Bendix based reasoning [22], Buchberger algorithm for polynomial–based logical reasoning [15,16] and conceptual processing in Formal Concept Analysis [3].

In this paper, we show a proof in PVS of correctness of the tableau algorithm for $\mathcal{ALC}$ described in theoretical papers. The hardest part of this task is the termination proof, for which we have extended the multiset library of PVS in order to include well–foundedness of the multiset order relation. We use this property and a measure function as it is described in [9] to prove the termination of the algorithm.

The paper is structured as follows. Section 2 is devoted to describe the proof of the well–foundedness of the multiset induced relation of a well–founded relation. Section 3 shows the syntax and semantics of the $\mathcal{ALC}$ logic and how we have formalized it in the PVS language specification. Also, it contains the specification in PVS of a prover for the $\mathcal{ALC}$ logic, with some details about its termination, soundness and completeness proofs. Section 4 is devoted to explain how we have constructed in PVS a concrete measure function. Finally, in the last section we draw some conclusions and suggest some lines of future work.

Since the whole formalization consists of a large collection of PVS theorems and definitions, we give an overview presenting the main results and a sketch of how the pieces fit together. We necessarily omit many details that, we expect, can be inferred from the context. We urge the interested reader to consult the whole PVS theory developed, which is widely documented and available at `http://www.cs.us.es/~mjoseh/alc/`.

2 Well–Founded Ordering of Multisets

Multisets (bags) over a given set T are sets that admit multiple occurrences of elements taken from T. Dershowitz and Manna [9] prove that every well–founded relation on T induces a well–founded relation on the set of finite multisets over T. They also show how the multiset ordering is used to prove the termination of production systems, programs defined in term of rewriting rules. In our work, we use this method to prove the termination and completeness of the $\mathcal{ALC}$ reasoner.

The goal of this section is to show our extension of the PVS multiset library, to include the formalization of the Dershowitz and Manna theorem. The proof formalized in the PVS system is based on the proof described by T. Nipkow [17]. It uses a characterization of well–foundedness based on the notion of well–founded part of a set, and the transitive closure of a well–founded binary relation.

2.1 An Alternative Characterization of Well–Foundedness

The definition of well–foundedness included in the PVS prelude is the usual one: a relation $<$ on T is well–founded if every non-empty subset of T has a minimal element (w.r.t. $<$). Nevertheless, the proof of well-foundedness of the multiset orderings that we have formalized is based on an alternative definition given by P. Aczel [1]. Let us start describing the PVS proof of the equivalence between Aczel's definition and the usual one.

Given a binary relation $<$ defined on a set T, the well–founded part of T with respect to $<$, denoted as $W(T,<)$, is the smallest subset of T closed under the set of rules $(\forall a \in T)[(\forall y < a)[y \in W(T,<)] \rightarrow a \in W(T,<)]$. In our PVS definition of $W(T,<)$, both the type T and the relation $<$ are introduced as theory parameters. Also, we use the support that provides PVS for constructing inductive definitions of sets or predicates.

```
[T: TYPE+, <: pred[[T,T]]]: THEORY

well_founded_part(x): INDUCTIVE bool =
  FORALL y: y < x IMPLIES well_founded_part(y)
```

This way, the above inductive definition of the well–founded part generates, automatically, the following induction axioms, which allow us to prove properties by induction on the defined set:

- Weak induction axiom for the well–founded part:

$$\frac{(\forall x)[(\forall y)[y < x \rightarrow P(y)] \rightarrow P(x)]}{(\forall x)[x \in W(T,<) \rightarrow P(x)]}$$

- Induction axiom for the well–founded part

$$\frac{(\forall x)[(\forall y)[y < x \rightarrow y \in W(T,<) \wedge P(y)] \rightarrow P(x)]}{(\forall x)[x \in W(T,<) \rightarrow P(x)]}$$

The following theorem characterizes the well–foundedness of a relation by means of its well–founded part.

Theorem 1. *$(T,<)$ is well–founded if and only if $W(T,<) = T$.*

```
well_founded_part_nsc: THEOREM
  well_founded?[T](<) IFF (FORALL x: well_founded_part(x))
```

The PVS proof of the necessary condition is carried out by induction on $(T,<)$ with respect to the predicate $x \in W(T,<)$. On the other hand, the sufficient condition is proved using the weak induction axiom for the well–founded part of T with respect to $<$.

Additionally, the PVS theory about well–foundedness that we have developed also includes alternative sufficient conditions of well–foundedness, like the following embedding lemma.

Lemma 1. *If $f : (S,<') \rightarrow (T,<)$ is monotone and $(T,<)$ is well–founded, then $(S,<')$ is well–founded.*

We prove it using the definition of well–foundedness based in the notion of minimal element.

2.2 Well–Foundedness of the Transitive Closure

The *transitive closure* of a binary relation on T, $<$, is the smallest relation $<^+$ such that

$$(\forall x, y \in T)[(x < y \vee (\exists z)[x <^+ z \wedge z < y]) \rightarrow x <^+ y]$$

```
tr_cl(<)(x,y): INDUCTIVE bool =
 x < y OR EXISTS z: tr_cl(<)(x,z) AND z < y
```

The main result about well–foundedness and transitive closure is the following:

Theorem 2. *If $(T, <)$ is well–founded, then $(T, <^+)$ is well–founded.*

```
well_founded_cl_tr: THEOREM
  well_founded?[T](<) IMPLIES well_founded?[T](tr_cl(<))
```

To prove it, by theorem 1, it is sufficient to prove that $W(T, <) \subseteq W(T, <^+)$. We prove this in PVS using the weak induction axiom generated by the definition of $W(T, <)$.

In a similar way, the reflexive transitive closure of a relation, `rtr_cl(<)`, has been defined and their main properties have been proved in PVS.

2.3 Well–Founded Multiset Relations (in PVS)

In order to specify in PVS the multiset relations, we have used the PVS library about bags.[1] In this library, a multiset (bag) of elements in T is represented by means of a function with domain T and range $\mathbb{N}$. Let us start showing the specification of the `bag` and `finite_bag` types, and also the specifications of the basic operations `insert` and `plus`, included in it (in the following, we will denote both as $\uplus$)

```
bag: TYPE = [T -> nat]
insert(x,b): bag = (LAMBDA t: IF x = t THEN b(t) + 1 ELSE b(t) ENDIF)
plus(a,b)  : bag = (LAMBDA t: a(t) + b(t))
bag_to_set(b): set[T] = {t: T | b(t) > 0}
is_finite(b): bool = is_finite(bag_to_set(b))
finite_bag: TYPE = {b: bag | is_finite(b)}
```

Let $<$ be a relation in T and $\mathcal{M}(T)$ the set of finite multisets over T. The *multiset relation* induced by $<$ on $\mathcal{M}(T)$ is the relation $<_{mult}$ defined as: $N <_{mult} M$ if there exist multisets $M_0, K_1, K_2 \in \mathcal{M}(T)$ such that $K_1 \neq \emptyset$, $M = M_0 \uplus K_1$, $N = M_0 \uplus K_2$ and $(\forall a)[a \in K_2 \rightarrow (\exists b)[b \in K_1 \wedge a < b]]$. An alternative definition could be obtained if instead of replacing a multiset K_1 of elements in M, a

[1] Available at
`http://shemesh.larc.nasa.gov/fm/ftp/larc/PVS-library/pvslib.html`

single element $b \in M$ were replaced by a multiset of smaller elements (w.r.t. $<$). That is, we could define the following *multiset reduction relation* denoted as $<_1$: $N <_1 M$ if there exists multisets M_0, K_2 and $b \in M$ such that $M = M_0 \uplus \{b\}$, $N = M_0 \uplus K_2$ and $(\forall a)[a \in K_2 \rightarrow a < b]$. It can be proved that if $<$ is transitive, then $<_{mult}$ is the transitive closure of $<_1$. Therefore, by theorem 2, proving the well–foundedness of $<_{mult}$ amounts to proving that $<_1$ is well–founded, provided that $<$ is transitive.

We specify the relations $<_1$ and $<_{mult}$ in PVS by `less_1` and `less_mult`, respectively, as follows

```
less(K,a): bool = FORALL b: member(b,K) IMPLIES b < a

less_1(N,M): bool =
  EXISTS M_0,a,K: M = insert(a,M_0) AND N = plus(M_0,K) AND less(K,a)

less_mult(N,M): bool =
  EXISTS M_0,K1,K2: nonempty_bag?(K1) AND M = plus(M_0, K1) AND
                    N = plus(M_0, K2) AND
                    FORALL a: member(a,K2) IMPLIES
                              EXISTS b: member(b,K1) AND a < b
```

Then, we stablish in PVS the following result, ensuring that the relation $<_1$ is well–founded on $\mathcal{M}(T)$.

Lemma 2. *Let $<$ be a well–founded relation on T. Then $<_1$ is a well–founded relation on $\mathcal{M}(T)$.*

```
wf_less_1: THEOREM well_founded?[finite_bag[T]](less_1)
```

To prove this lemma, using theorem 1, it is sufficient to prove that $\mathcal{M}(T) \subseteq W(\mathcal{M}(T), <_1)$. The PVS proof is carried out by induction on finite multisets, according to the following scheme:

$$\frac{P(\emptyset) \wedge (\forall a)(\forall M)[P(M) \rightarrow P(M \uplus \{a\})]}{(\forall M)P(M)}$$

where the predicate $P(M)$ stands $M \in W(\mathcal{M}(T), <_1)$. Therefore, we have to prove:

1. $\emptyset \in W(\mathcal{M}(T), <_1)$ (which is true by its definition).
2. $(\forall a)[(\forall M)[M \in W(\mathcal{M}(T), <_1) \rightarrow M \uplus \{a\} \in W(\mathcal{M}(T), <_1)]]$.
   This result is proved by well–founded induction on $<$ (since it is well–founded), with the following predicate $P(a)$:

   $$(\forall M)[M \in W(\mathcal{M}(T), <_1) \rightarrow M \uplus \{a\} \in W(\mathcal{M}(T), <_1)]$$

   With this, the proof is reduced to prove $(\forall b)[b < a \rightarrow P(b)] \rightarrow P(a)$, or equivalently, to prove that if

   $$(\forall b)[b < a \rightarrow (\forall M)[M \in W(\mathcal{M}(T), <_1) \rightarrow M \uplus \{b\} \in W(\mathcal{M}(T), <_1)]]$$

then

$$(\forall M)[M \in W(\mathcal{M}(T), <_1) \rightarrow M \uplus \{a\} \in W(\mathcal{M}(T), <_1)] \tag{3}$$

We finish proving (3) using the weak induction axiom for the well–founded part $W(\mathcal{M}(T), <_1)$, with the predicate $Q(M)$:

$$M \uplus \{a\} \in W(\mathcal{M}(T), <_1) \vee M \notin W(\mathcal{M}(T), <_1).$$

Finally, as consequence of lemma 2, we obtain the main theorem of this section:

Theorem 3 (Dershowitz and Manna). *Let $<$ be a transitive and well–founded relation on T. Then the relation $<_{mult}$ is a well–founded relation on $\mathcal{M}(T)$.*

```
less_mult_is_wf: THEOREM
 transitive?[T](<) IMPLIES well_founded?[finite_bag[T]](less_mult)
```

A sketch of the proof is as follows. First, we show that the relation $<_{mult}$ is contained in $<_1^+$. Furthermore, if the relation $<$ is transitive, then $<_{mult}$ is transitive and contains the relation $<_1^+$. Then, if $<$ is transitive, $<_1^+ = <_{mult}$ and therefore, by lemma 2 and theorem 2, we conclude that $<_{mult}$ is well–founded.

3 Tableau Reasoning for $\mathcal{ALC}$ –Satisfiability

In this section, we first describe the basic components of $\mathcal{ALC}$ logic, and we show below how we have formalized in PVS the tableau based algorithm for this logic, as well as the proofs of its termination, soundness and completeness.

3.1 Syntax and Semantics of the $\mathcal{ALC}$ Logic

We start presenting a brief introduction to the $\mathcal{ALC}$ –logic, its syntax and its semantics, along with the corresponding description of its specification in PVS.

Let NC be a set of *concept names* and NR be a set of *role names*. The set of $\mathcal{ALC}$ –*concepts* is built inductively from these names as described by the following grammar, where $A \in \mathsf{NC}$ and $R \in \mathsf{NR}$

$$C ::= A \mid \neg C \mid C_1 \sqcap C_2 \mid C_1 \sqcup C_2 \mid \forall R.C \mid \exists R.C$$

The set of $\mathcal{ALC}$ –concepts can be represented in PVS as a recursive datatype, using the mechanism for defining abstract datatypes [20], and specifying the constructors, the accessors and the recognizers. When a datatype is typechecked in PVS, a new theory is created providing axioms and inductions principles for this datatype. In particular, this theory contains the relation `subterm` (specifying the notion of *subconcept*) and the well–founded relation `<<`, that is useful to make recursive definitions on concepts.

To introduce the assertional knowledge, let NI be a set of *individual names*. Given individual names $x, y \in \mathsf{NI}$, a concept C and a role name R, the expressions $x\colon C$ and $(x, y)\colon R$ are called *assertional axioms*. An *ABox* $\mathcal{A}$ is a finite set of assertional axioms. We specify in PVS the assertional axioms by a datatype and the ABox by a type.

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

ABox: TYPE = finite_set[assertional_ax]
```

The semantics of description logics is defined in terms of interpretations. An $\mathcal{ALC}$ *–interpretation* $\mathcal{I}$ is a pair $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$, where $\Delta^{\mathcal{I}}$ is a non–empty set called the *domain*, and $\cdot^{\mathcal{I}}$ is an *interpretation function* that maps every concept name A to a subset $A^{\mathcal{I}}$ of $\Delta^{\mathcal{I}}$, every role name R to a binary relation $R^{\mathcal{I}}$ over $\Delta^{\mathcal{I}}$ and every individual x to an element of $\Delta^{\mathcal{I}}$. We represent in PVS an interpretation $\mathcal{I}$ as a structure that contains the domain of $\mathcal{I}$ and the functions that define the interpretation of concept names, role names, and the individuals.

```
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)] #]
```

It should be noted that in this specification we have used a universal type `U` to represent the elements of the domain. Also, we have taken advantage of the ability of PVS to deal with dependent types.

The interpretation function is extended to non-atomic concepts as follows

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

We have specified this notion in PVS in a natural way, by recursion on C, using the well–founded relation `<<`.

The interpretation $\mathcal{I}$ is a *model* of a concept C if $C^{\mathcal{I}} \neq \emptyset$. Thus, a concept C is called *satisfiable* if it has a model

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

The interpretation $\mathcal{I}$ *satisfies* the assertional axiom $x\colon C$ if $x^{\mathcal{I}} \in C^{\mathcal{I}}$ and satisfies $(x,y)\colon R$ if $(x^{\mathcal{I}}, y^{\mathcal{I}}) \in R^{\mathcal{I}}$. It *satisfies* the ABox $\mathcal{A}$ if it satisfies every axiom in $\mathcal{A}$. In that case, $\mathcal{A}$ is called *satisfiable* and $\mathcal{I}$ is called a *model* of $\mathcal{A}$.

We have made the PVS formalization of the above definitions using the PVS set theory and its capability of managing the existential and universal quantifiers.

The previous definitions naturally pose some standard inference problems for DLs systems, such as concept and ABox satisfiability. It can be proved, and we have done it in PVS, that concept satisfiability can be reduced to ABox satisfiability (i.e., C is satisfiable iff for all $x \in \mathsf{NI}$, $\{x{:}C\}$ is satisfiable).

3.2 Deciding Concept Satisfiability for $\mathcal{ALC}$

A tableau algorithm for $\mathcal{ALC}$ tries to prove the satisfiability of a concept C by attempting to explicitly construct a model of C. This is done considering an individual name x_0 and manipulating the initial ABox $\{x_0{:}C\}$, applying a set of completion rules. In this process, we consider concepts in negation normal form (NNF), a form in which negations appear only in front of concept names. This does not mean a restriction since it is easy to specify a PVS function such that, for each $\mathcal{ALC}$–concept computes another equivalent [2] in NNF form.

An ABox $\mathcal{A}$ contains a *clash* if, for some individual name $x \in \mathsf{NI}$ and concept name $A \in \mathsf{NC}$, $\{x{:}A, x{:}\neg A\} \subseteq \mathcal{A}$. Otherwise, $\mathcal{A}$ is called *clash–free*. To test the satisfiability of an $\mathcal{ALC}$–concept C in NNF, the $\mathcal{ALC}$–*algorithm* works starting from the initial ABox $\{x_0{:}C\}$ and iteratively applying the following *completion rules*

$\rightarrow_\sqcap$: if $x{:}C \sqcap D \in \mathcal{A}$ and $\{x{:}C, x{:}D\} \not\subseteq \mathcal{A}$
then $\mathcal{A} \rightarrow_\sqcap \mathcal{A} \cup \{x{:}C, x{:}D\}$

$\rightarrow_\sqcup$: if $x{:}C \sqcup D \in \mathcal{A}$ and $\{x{:}C, x{:}D\} \cap \mathcal{A} = \emptyset$
then $\mathcal{A} \rightarrow_\sqcup \mathcal{A} \cup \{x{:}E\}$ for some $E \in \{C, D\}$

$\rightarrow_\exists$: if $x{:}\exists R.D \in \mathcal{A}$ and there is no y with $\{(x, y){:}R, y{:}D\} \subseteq \mathcal{A}$
then $\mathcal{A} \rightarrow_\exists \mathcal{A} \cup \{(x, y){:}R, y{:}D\}$ for a fresh individual y

$\rightarrow_\forall$: if $x{:}\forall R.D \in \mathcal{A}$ and there is a y with $(x, y){:}R \in \mathcal{A}$ and $y{:}D \notin \mathcal{A}$
then $\mathcal{A} \rightarrow_\forall \mathcal{A} \cup \{y{:}D\}$

It stops when a clash has been generated or when no rule is applicable. In the last case, the ABox is *complete* and a model can be derived from it. The algorithm answers "C is satisfiable" if a complete and clash–free ABox has been generated.

These completion rules can be seen as a production system. Nevertheless, we have not formalized them in a functional way, but following a more declarative style, defining the completion rules in PVS as binary relations between ABoxes. For example, $\mathcal{A}_1 \rightarrow_\sqcap \mathcal{A}_2$ if there exists an assertional axiom $x{:}C \sqcap D$ in $\mathcal{A}_1$ such that $\{x{:}C, x{:}D\} \not\subseteq \mathcal{A}_1$ and $\mathcal{A}_2 = \mathcal{A}_1 \cup \{x{:}C, x{:}D\}$

```
and_step(AB1, AB2): bool =
 EXISTS Aa: member(Aa,AB1) AND
            instanceof?(Aa) AND
            alc_and?(right(Aa)) AND
            LET x = left(Aa), C = conc1(right(Aa)), D = conc2(right(Aa))
             IN (NOT member(instanceof(x, C), AB1) OR
                 NOT member(instanceof(x, D), AB1)) AND
                 AB2 = add(instanceof(x, C), add(instanceof(x, D), AB1))
```

Once the rules have been specified in this way, we define the successor relation on the ABoxes type: $\mathcal{A}_1 \rightarrow \mathcal{A}_2$ if $\mathcal{A}_1$ does not contain a clash and $\mathcal{A}_2$ is obtained from $\mathcal{A}_1$ by the application of a completion rule

[2] Two concepts are equivalent if they have the same models.

```
successor(AB2,AB1): bool =
  (NOT contains_clash(AB1)) AND
  (and_step(AB1,AB2)  OR or_step_1(AB1,AB2) OR or_step_2(AB1,AB2) OR
   some_step(AB1,AB2) OR all_step(AB1,AB2))
```

It should be noted that we have specified the non-deterministic rule $\rightarrow_{\sqcup}$ by two binary relations (`or_step_1` and `or_step_2`), one for each component.

Take into account that the completion process can be seen as a closure process, we say the ABox $\mathcal{A}_2$ is an *expansion* of the ABox $\mathcal{A}_1$ if $\mathcal{A}_1 \overset{*}{\rightarrow} \mathcal{A}_2$, where $\overset{*}{\rightarrow}$ is the reflexive and transitive closure of $\rightarrow$

```
is_expansion(AB1)(AB2): bool = rtr_cl(successor)(AB2, AB1)
```

To illustrate the completion process, the following example shows the application of some completion rules to an initial ABox $\{x_0{:}C\}$

Example 1. Let C be the concept $\forall R.D \sqcap (\exists R.(D \sqcup E) \sqcap \exists R.(D \sqcup F))$. Then,

$$\begin{aligned} & \mathcal{A}_0 := \{x_0{:}\forall R.D \sqcap (\exists R.(D \sqcup E) \sqcap \exists R.(D \sqcup F)) \\ \overset{*}{\rightarrow}\ & \mathcal{A}_1 := \mathcal{A}_0 \cup \{x_0{:}\forall R.D, x_0{:}\exists R.(D \sqcup E), x_0{:}\exists R.(D \sqcup F)\} \\ \rightarrow\ & \mathcal{A}_2 := \mathcal{A}_1 \cup \{(x_0, x_1){:}R, x_1{:}D \sqcup E\} \\ \rightarrow\ & \mathcal{A}_3 := \mathcal{A}_2 \cup \{x_1{:}D\} \end{aligned}$$

Once defined the expansion relation, we use it to specify the notion of consistency: An ABox $\mathcal{A}$ is *consistent* if it has a complete and clash–free expansion. Similarly, a concept C is *consistent* if the initial ABox $\{x_0{:}C\}$ is consistent

```
is_consistent_abox(AB): bool =
  EXISTS AB1: is_expansion(AB)(AB1) AND complete_clash_free(AB1)

is_consistent_concept(C): bool =
  is_consistent_abox(singleton(instanceof(x_0,C)))
```

where `complete_clash_free`($\mathcal{A}$) holds if the ABox $\mathcal{A}$ is both complete and clash–free.

This definition is the PVS specification of a generic $\mathcal{ALC}$ –algorithm for deciding satisfiability of $\mathcal{ALC}$ –concepts. It should be pointed out the two kinds of non–determinism in it: the way in which the rule $\rightarrow_{\sqcup}$ is applied ("don't know" non-determinism); and the choice of which rule to apply in each step and to which axiom ("don't care" non-determinism). To prove that the algorithm is correct, we have to establish its termination, soundness and completeness. The following subsections describe the corresponding PVS proofs.

3.3 Soundness

The $\mathcal{ALC}$ –algorithm is sound, that is:

```
alc_soundness: THEOREM
  is_consistent_concept(C) IMPLIES concept_satisfiable?(C)
```

The PVS proof is based on the following steps:

1. If $\mathcal{A}$ is a complete and clash–free expansion of an initial ABox $\mathcal{A}_0$, then $\mathcal{A}$ is satisfiable. We have proved this by specifying in PVS the canonical interpretation $\mathcal{I}_\mathcal{A}$ associated with $\mathcal{A}$, and proving that $\mathcal{I}_\mathcal{A}$ is a model of $\mathcal{A}$.
2. If $\mathcal{A}_1 \rightarrow \mathcal{A}_2$ and $\mathcal{A}_2$ is satisfiable, then $\mathcal{A}_1$ is satisfiable too.
3. Finally, we have proved the soundness theorem, using the induction scheme suggested by the above definition of the expansion relation (which in turn is based on the inductive definition of closure).

3.4 Termination

In order to verify the termination of the $\mathcal{ALC}$ –algorithm, it suffices to prove that the successor relation, defined on the set $\mathcal{E}(C)$ of the expansions of the initial ABox $\{x_0{:}C\}$, is well founded

```
well_founded_successor: THEOREM
  well_founded?[expansion_abox_concept(C)](successor)
```

where by the type `expansion_abox_concept`(C), we specify the set $\mathcal{E}(C)$

```
expansion_abox_concept(C:(is_nnf?)): TYPE =
  (is_expansion(singleton(instanceof(x_0, C))))
```

The proof of the well–foundedness of the successor relation is based on the embedding lemma (Lemma 1). So, it suffices to show the existence of a type T, a well–founded relation $<$ on T, and a function

$$\mu_C : \mathcal{E}(C) \rightarrow T \tag{1}$$

such that

$$(\forall \mathcal{A}_1, \mathcal{A}_2)[\mathcal{A}_1 \rightarrow \mathcal{A}_2 \Rightarrow \mu_C(\mathcal{A}_2) < \mu_C(\mathcal{A}_1)] \tag{2}$$

Those functions with these properties are called *measure functions*.

The formalization of this proof in PVS has been carried out in two phases. In the first one, we assume the existence of a measure function, including T and $<$ in the parameters, and (1) and (2) in the body of the PVS theory,

```
[..., T: TYPE+, <: (well_founded?[T])]: THEORY

measure_concept(C): [expansion_abox_concept(C) -> T]

measure_concept_decrease_successor: AXIOM
  FORALL (AB1,AB2: expansion_abox_concept(C)):
    successor(AB2,AB1)
    IMPLIES measure_concept(C)(AB2) < measure_concept(C)(AB1)
```

and we prove that successor is well–founded on $\mathcal{E}(C)$.

In the second one, we prove the existence of measure functions. For the sake of completeness, we outline Nutt's definition of a measure function. In [18], W. Nutt constructs a measure function taking T as the type of the finite multisets of pairs of natural numbers $\mathcal{M}(\mathbb{N} \times \mathbb{N})$, and the well–founded order $<$ as the extension to $\mathcal{M}(\mathbb{N} \times \mathbb{N})$ of the lexicographic order on $\mathbb{N} \times \mathbb{N}$, that we denote by $<_{mult}$. In order to formalize in PVS that construction, it is necessary carry out two tasks:

1. To prove that the extension to $\mathcal{M}(\mathbb{N}\times\mathbb{N})$ of the lexicographic order on $\mathbb{N}\times\mathbb{N}$ is a well–founded ordering. For this, it suffices to instantiate, in theorem 3, T by $\mathbb{N}\times\mathbb{N}$ and $<$ by the lexicographic ordering on $\mathbb{N}\times\mathbb{N}$.
2. To define a function μ_C mapping each expansion $\mathcal{A}\in\mathcal{E}(C)$ to a multiset of pairs, such that $\mu_C(\mathcal{A}_2) <_{mult} \mu_C(\mathcal{A}_1)$ if $\mathcal{A}_1 \to \mathcal{A}_2$. For the sake of clarity, we devote section 4 to explain the details of its formalization.

3.5 Completeness

The last verification task is the proof of completeness of $\mathcal{ALC}$–algorithm, that is:

```
alc_completeness: THEOREM
  concept_satisfiable?(C) IMPLIES is_consistent_concept(C)
```

The PVS proof is achieved by means three subtasks in turn:

1. If $\mathcal{A}$ is a satisfiable ABox, then $\mathcal{A}$ is clash–free.
2. If $\mathcal{A}_1$ is a satisfiable and not complete ABox, then there exists a satisfiable ABox $\mathcal{A}_2$, which is successor of $\mathcal{A}_1$.
3. If $\mathcal{A}\in\mathcal{E}(C)$ is satisfiable, then there exists a complete and clash–free expansion of $\mathcal{A}$ in $\mathcal{E}(C)$ itself.

In order to carry out the second task, we have proved that for every satisfiable not complete ABox $\mathcal{A}$, for every rule r and for every axiom of $\mathcal{A}$ to which r is applied, there exists an axiom such that by adding it we obtain a satisfiable ABox. On the other hand, the last one is the key lemma for the main proof. We have proved it by well–founded induction on the successor relation.

4 Measure on $\mathcal{ALC}$–Expansion of C

In this section we show a measure function on $\mathcal{E}(C)$ verifying the monotonicity condition of section 3.4. The idea for defining the measure function μ_C is to map each expansion $\mathcal{A}\in\mathcal{E}(C)$ to a multiset of pairs, in such way that those pairs represent all possibles rules that can be applied to $\mathcal{A}$.

The first step is to define the notions of *level* and *colevel*. For this, we have used the library of graphs of PVS [7]. We define the *associated graph* to an ABox $\mathcal{A}$, $\mathcal{G}(\mathcal{A})$, as the graph whose vertices are the individuals that occur in $\mathcal{A}$, and whose edges are the subsets $\{x,y\}$, such that $(x,y):R\in\mathcal{A}$ for some role R.

```
graph_assoc_abox(AB: ABox): graph[NI] =
  (# vert:= occur_ni(AB), edges:= dbl_assoc_abox(AB) #)
```

From this definition, we prove that if $\mathcal{A}\in\mathcal{E}(C)$, then $\mathcal{G}(\mathcal{A})$ is a tree with root x_0. This fact allows us to define the level of x in $\mathcal{A}$ as the length of the path from x_0 to x (minus 1), and the colevel of x in $\mathcal{A}$, $|x|_{\mathcal{A}}$, as the difference between the size of the concept C (denoted by $|C|$) and the level of x in $\mathcal{A}$.

```
level(AB)(x:(occur_ni(AB))): nat = l(path_from_root(AB)(x)) - 1
colevel(AB)(x:(occur_ni(AB))): nat = size(C) - level(AB)(x)
```

Also, we prove that the colevel of an individual in $\mathcal{A}$ remains invariant under the completion rules and that if y is a *successor* of x in $\mathcal{A}$ (i.e., (x, y): $R \in \mathcal{A}$), then $|y|_{\mathcal{A}} = |x|_{\mathcal{A}} - 1$. Both properties are essential to prove the monotonicity of μ_C.

```
successor_preserve_colevel: LEMMA
 occur_ni(AB1)(y) AND successor(AB2,AB1)
 IMPLIES colevel(AB2)(y) = colevel(AB1)(y)

colevel_successor_related: LEMMA
 successor_related(AB)(y,x) IMPLIES colevel(AB)(y) = colevel(AB)(x) - 1
```

As we have already said, the elements of the multiset associated to an expansion $\mathcal{A}$ should represent all possibles applicable rules to $\mathcal{A}$. In some cases, the applicability of a rule is completely determined by an instance axiom of $\mathcal{A}$, but that is not the case for the $\rightarrow_\forall$ rule. Thus, in order to capture the notion of applicability of a rule, we introduce the type *activation* (`activ`), whose elements are structures consisting of an instance axiom and an individual, that made it applicable. Then, we specify when an activation is applicable in $\mathcal{A}$ and we define the *agenda* of $\mathcal{A}$, $\mathsf{agenda}(\mathcal{A})$, as the set of applicable activations in $\mathcal{A}$.

```
activ: TYPE = [# ax: (instanceof?), witness: NI #]

applicable_activ(Ac,AB): bool =
 LET Aa = ax(Ac),y = witness(Ac),x = left(Aa),D = right(Aa) IN
 member(Aa,AB) AND
 CASES D OF
  alc_a(A)       : false,
  alc_not(D1)    : false,
  alc_and(C1,C2) : x = y AND (NOT member(instanceof(x,C1),AB) OR
                              NOT member(instanceof(x,C2),AB)),
  alc_or(C1,C2)  : x = y AND NOT member(instanceof(x,C1),AB) AND
                             NOT member(instanceof(x,C2),AB),
  alc_all(R,D1)  : x /= y AND member(related(x,R,y),AB) AND
                           NOT member(instanceof(y,D1),AB),
  alc_some(R,D1) : x = y AND NOT (EXISTS y: member(related(x,R,y),AB) AND
                                            member(instanceof(y,D1),AB))
 ENDCASES

agenda(AB): finite_set[activ] = {Ac | applicable_activ(Ac, AB)}
```

To specify the function μ_C we found the following difficulty: we can not define a multiset in PVS in a declarative way, as with sets. Thus, the measure of the expansion $\mathcal{A}$, $\mu_C(\mathcal{A})$, is constructed by recursion in the agenda of $\mathcal{A}$, adding the pair $(|y|_{\mathcal{A}}, |D|)$ for each applicable activation $[x{:}\, D, y]$.

```
bag_assoc_activ(Ac, AB): finite_bag[[nat, nat]] =
  IF NOT applicable_activ(Ac, AB) THEN emptybag
```

```
  ELSE LET Aa = ax(Ac), y = witness(Ac), D = right(Aa) IN
           singleton_bag((colevel(AB)(y), size(D)))
  ENDIF

expansion_measure_aux(AB,(AB1: finite_set[activ])):
                      RECURSIVE finite_bag[[nat, nat]] =
  IF empty?(AB1) THEN emptybag
  ELSE plus(bag_assoc_activ(choose(AB1), AB),
            expansion_measure_aux(AB, rest(AB1)))
  ENDIF
MEASURE card(AB1)

expansion_measure(AB): finite_bag[[nat, nat]] =
  expansion_measure_aux(AB, agenda(AB))
```

We illustrate the evolution of measures through the effect of completion rules to example 1.

Example 2. The agendas and measures of the ABoxes of Example 1 are:

	agenda	measure
$\mathcal{A}_0$	$\{(x_0{:}\forall R.D \sqcap (\exists R.(D \sqcup E) \sqcap \exists R.(D \sqcup F)), x_0)\}$	$\dot{\{}(15,15)\dot{\}}$
$\mathcal{A}_1$	$\{(x_0{:}\exists R.(D \sqcup E), x_0), (x_0{:}\exists R.(D \sqcup F), x_0)\}$	$\dot{\{}(15,5),(15,5)\dot{\}}$
$\mathcal{A}_2$	$\{(x_0{:}\exists R.(D \sqcup F), x_0), (x_0{:}\forall R.D, x_1), (x_1{:}D \sqcup E, x_1)\}$	$\dot{\{}(14,3),(14,3),(15,5)\dot{\}}$
$\mathcal{A}_3$	$\{(x_0{:}\exists R.(D \sqcup F), x_0)\}$	$\dot{\{}(15,5)\dot{\}}$

Finally, we prove the theorem that assures the monotony of μ_C.

Theorem 4. *Let $\mathcal{A}_1, \mathcal{A}_2 \in \mathcal{E}(C)$. If $\mathcal{A}_1 \rightarrow \mathcal{A}_2$ then $\mu_C(\mathcal{A}_2) <_{mult} \mu_C(\mathcal{A}_1)$.*

```
expansion_measure_decrease_successor: THEOREM
  successor(AB2, AB1)
  IMPLIES less_mult(expansion_measure(AB2), expansion_measure(AB1))
```

The formalization of the proof of this theorem in PVS has turned out to be more difficult than the hand proof presented in [18]. Firstly, we can observe (in Example 2) that if $\mathcal{A}_1, \mathcal{A}_2 \in \mathcal{E}(C)$ are such that $\mathcal{A}_1 \rightarrow \mathcal{A}_2$, then there exists an activation $Ac_1 = [x : D, y] \in \mathsf{agenda}(\mathcal{A}_1)$, that matches with the applied rule. In addition, $Ac_1 \notin \mathsf{agenda}(\mathcal{A}_2)$ and, for each activation Ac_2 introduced in the $\mathsf{agenda}(\mathcal{A}_2)$ as result of the rule application, its associated pair is smaller (lexicographically) than $(|y|_{\mathcal{A}}, |D|)$. Indeed, one of the following cases may occur:

1. $Ac_2 = [x{:}E, z]$, being z a successor of y in $\mathcal{A}_2$. Then, $|z|_{\mathcal{A}_2} < |y|_{\mathcal{A}_2} = |y|_{\mathcal{A}_1}$. So, $(|z|_{\mathcal{A}_2}, |E|) < (|y|_{\mathcal{A}_1}, |D|)$.
2. $Ac_2 = [x{:}E, y]$, being E a subconcept of D. In this case, $|y|_{\mathcal{A}_2} = |y|_{\mathcal{A}_1}$ and $|E| < |D|$. So, $(|y|_{\mathcal{A}_2}, |E|) < (|y|_{\mathcal{A}_1}, |D|)$.

Secondly, we should note that the application of a rule can disable some activations of the agenda and, so, it can eliminate its associated pairs of the multiset.

Thus, we define in PVS the multiset $K_1 = \mu_{aux}(\mathcal{A}_1, \mathsf{agenda}(\mathcal{A}_1) \setminus \mathsf{agenda}(\mathcal{A}_2))$, whose elements are the pairs associated to disabled activations. Also, the multiset $K_2 = \mu_{aux}(\mathcal{A}_2, \mathsf{agenda}(\mathcal{A}_2) \setminus \mathsf{agenda}(\mathcal{A}_1))$ that contains the pairs associated to new enabled activations. Finally, $M_0 = \mu_{aux}(\mathcal{A}_1, \mathsf{agenda}(\mathcal{A}_1) \cap \mathsf{agenda}(\mathcal{A}_2))$ is the multiset whose elements are the pairs associated to the activations that remains enabled after the application of the rule. Regarding these multisets, we prove the following properties: (1) $K_1 \neq \emptyset$, (2) $\mu_C(\mathcal{A}_2) = M_0 \uplus K_2$, (3) $\mu_C(\mathcal{A}_1) = M_0 \uplus K_1$ y (4) $(\forall a \in K_2)(\exists b \in K_1)[a < b]$. Thus, we conclude that $\mu_C(\mathcal{A}_2) <_{mult} \mu_C(\mathcal{A}_1)$.

Once the measure function has been constructed, the parameters T and $<$, and the signature `measure-concept(C)` of subsection 3.4 are interpreted by the appropriated mechanism of PVS.

5 Conclusions and Future Work

We have presented a formalization of the $\mathcal{ALC}$ logic in PVS, and a formalization of a tableau–based algorithm for checking satisfiability of $\mathcal{ALC}$ –concepts, proving its soundness, completeness and termination. Both for the termination proof and the completeness proof we have used the well–foundedness of the multiset relation induced by a well–founded relation, a property that we have also formally proved. This last result has been proved in a general setting. So, the parameters of the PVS theory can be instantiated in order to establish the well–foundedness of concrete relations.

It should be pointed out that the choice of PVS as our verification system has turned out to be beneficial for our formalization, since PVS provide definition of abstract datatypes, inductive sets and dependent types, as well as parameterized theories of sets, multisets and graphs.

To the best of our knowledge, it is the first work on formalizing DL reasoning, although related works (see, for example [14,11,21]) have been done for other logics. And, also, the first work in PVS about the well–foundedness of multiset relations. Others formal proofs of the well–foundedness of the multiset ordering have been carried out, in a similar way, in Coq [8] and in Isabelle. In [23], an ACL2 formalization of the same result is also presented.

Finally, we point out some possible lines for future work. We plan to continue the work following two research lines. First, we will apply type and operator refinement techniques presented in [3], in order to construct $\mathcal{ALC}$ –reasoners as PVSio specifications, executable and formally verified, refining the specification of the $\mathcal{ALC}$ –algorithm that we have presented here. Due to "don't care" nondeterminism of the algorithm specified in PVS, its correctness will be translated to specific implementations, whose efficiency will depend of the applied optimization strategy.

Also, we are interested in extending the formalization of the $\mathcal{ALC}$ logic to other descriptive logics, incrementally approaching us to the description logic $\mathcal{SHOIN}$, which is the description logic corresponding to OWL–DL [12], by adding new constructors. The modular characteristic of the extensions of our

formalization may provide the automatic synthesis of ad–hoc reasoning systems for specifics ontologies [2].

References

1. Aczel, P.: An introduction to inductive definitions. In: Barwise, J. (ed.) Handbook of Mathematical Logic, pp. 739–782. North–Holland Publishing Company, Amsterdam (1977)
2. Alonso, J.A., Borrego, J., Chávez, A.M., Martín, F.J.: Foundational challenges in automated semantic web data and ontology cleaning. IEEE Intelligent Systems 21(1), 45–52 (2006)
3. Alonso, J.A., Borrego, J., Hidalgo, M.J., Martín, F.J., Ruiz, J.L.: Verification of the Formal Concept Analysis. RACSAM (Revista de la Real Academia de Ciencias), Serie A: Matemáticas 98, 3–16 (2004)
4. Baader, F., Horrocks, I., Sattler, U.: Description logics as ontology languages for the semantic web. In: Hutter, D., Stephan, W. (eds.) Mechanizing Mathematical Reasoning. LNCS (LNAI), vol. 2605, pp. 228–248. Springer, Heidelberg (2005)
5. Baader, F., McGuiness, D., Nardi, D., Patel-Schneider, P. (eds.): The Description Logic Handbook: Theory, Implementation and Applications. Cambridge University Press, Cambridge (2003)
6. Bechhofer, S., Harmelen, F.v., Hendler, J., McGuinness, I.H.D.L., Patel-Schneider, P.F., Stein, L.A.: OWL Web Ontology Language Reference (2004), available on the Web at `http://www.w3.org/TR/owl-ref`
7. Butler, R.W., Sjogren, J.: A PVS graph theory library. Technical report, NASA Langley (1998)
8. Coupet-Grimal, S., Delobel, W.: An effective proof of the well–foundedness of the multiset path ordering. Applicable Algebra in Engineering, Communication and Computing 17(6), 453–469 (2006)
9. Dershowitz, N., Manna, Z.: Proving termination with multiset orderings. Communications of the ACM 22(8), 465–476 (1979)
10. Haarslev, V., Möller, R.: RACER system description. In: Goré, R.P., Leitsch, A., Nipkow, T. (eds.) IJCAR 2001. LNCS (LNAI), vol. 2083, pp. 701–705. Springer, Heidelberg (2001)
11. Harrison, J.: Formalizing basic first order model theory. In: Grundy, J., Newey, M. (eds.) Theorem Proving in Higher Order Logics. LNCS, vol. 1479, Springer, Heidelberg (1998)
12. Horrocks, I., Patel-Schneider, P.: Reducing OWL entailment to description logic satisfiability. J. of Web Semantics 1(4), 345–357 (2004)
13. Horrocks, I., Patel-Schneider, P.F., van Harmelen, F.: From $\mathcal{SHIQ}$ and RDF to OWL: The making of a Web Ontology Language. J. of Web Semantics 1(1), 7–26 (2003)
14. Martín, F.J., Alonso, J.A., Hidalgo, M.J., Ruiz, J.L.: Formal verification of a generic framework to synthesize SAT–provers. Journal of Automated Reasoning 32(4), 287–313 (2004)
15. Medina, I., Palomo, F., Alonso, J.A.: A certified polynomial-based decision procedure for propositional logic. In: Boulton, R.J., Jackson, P.B. (eds.) TPHOLs 2001. LNCS, vol. 2152, pp. 297–312. Springer, Heidelberg (2001)
16. Medina, I., Palomo, F., Alonso, J.A., Ruiz, J.L.: Verified computer algebra in ACL2. In: Buchberger, B., Campbell, J.A. (eds.) AISC 2004. LNCS (LNAI), vol. 3249, pp. 171–184. Springer, Heidelberg (2004)

17. Nipkow, T.: An inductive proof of the wellfoundedness of the multiset order. A proof due to W. Buchholz (1998), available on the Web at `http://www4.informatik.tu-muenchen.de/~nipkow/misc/multiset.ps`
18. Nutt, W.: Algorithms for constraint in deduction and knowledge representation. PhD thesis, Universität des Saarlandes (1993)
19. Owre, S., Rushby, J.M., Shankar, N.: PVS: A Prototype Verification System. In: Kapur, D. (ed.) Automated Deduction - CADE-11. LNCS, vol. 607, pp. 748–752. Springer, Heidelberg (1992)
20. Owre, S., Shankar, N.: Abstract datatype in PVS. Technical report, Computer Science Laboratory, SRI International (1997)
21. Ridge, T., Margetson, J.: A mechanically verified, sound and complete theorem prover for first order logic. In: Hurd, J., Melham, T. (eds.) TPHOLs 2005. LNCS, vol. 3603, pp. 294–309. Springer, Heidelberg (2005)
22. Ruiz, J.L., Alonso, J.A., Hidalgo, M.J., Martín, F.J.: Formal proofs about rewriting using ACL2. Ann. Math. Artif. Intell. 36(3), 239–262 (2002)
23. Ruiz, J.L., Alonso, J.A., Hidalgo, M.J., Martín, F.J.: Termination in ACL2 using multiset relation. In: Kamareddine, F.D. (ed.) Thirty Five Years of Automating Mathematics, pp. 217–245. Kluwer Academic Publishers, Dordrecht (2003)
24. Schmidt–Schauß, M., Smolka, G.: Attributive concept descriptions with complements. Artificial Intelligence 48(1), 1–26 (1991)
25. Tsarkov, D., Horrocks, I.: FaCT++ description logic reasoner: System description. In: Furbach, U., Shankar, N. (eds.) IJCAR 2006. LNCS (LNAI), vol. 4130, pp. 292–297. Springer, Heidelberg (2006)