A Dynamic Description Logic for Semantic Web Service

Liang Chang\textsuperscript{1,2}, Fen Lin\textsuperscript{1,2}
\textsuperscript{1}Graduate University of
Chinese Academy of Sciences,
Beijing 100049, China
changl@ics.ict.ac.cn, linf@ics.ict.ac.cn

Zhongzhi Shi\textsuperscript{2}
\textsuperscript{2}Institute of Computing Technology,
Chinese Academy of Sciences,
P.O. Box 2704, Beijing 100080, China
shizz@ics.ict.ac.cn

Abstract

This paper presents a dynamic description logic and a sound and complete tableau-based satisfiability-checking algorithm for it. This logic is a combination of a typical action theory and the description logic $\text{ALCO}^\oplus$, with a feature that actions are treated as citizens. On the one hand, actions are represented over ontologies expressed in description logic; On the other hand, actions can be used as modal operators for the construction of concepts. Therefore, not only actions but also concepts with dynamic meanings can be described. Furthermore, actions can also be used as modal operators for the construction of formulas, so that many reasoning tasks on actions and concepts can be realized with the help of the satisfiability-checking algorithm for formulas. These properties make this logic more capable for the semantic web service.

1. Introduction

Description logics are logical formalisms tailored for expressing knowledge about concepts and concept hierarchies. They are playing important roles in the semantic web, acting as logical foundation of the ontology language OWL [4].

However, description logics can only represent knowledge about static application domains. For many dynamic-related knowledge, especially for these about semantic web services, description logics are not adequate.

A dynamic extension of description logics was proposed by Wolter [8], in which the propositional dynamic logic PDL was combined with the description logic $\text{ALC}$, so that concepts with dynamic meaning could be described. For example, a concept $\textit{Mortal}$ could be described as:

$$\textit{Mortal} = \exists <\textit{die}> \textit{LivingBeing}$$

which stated that a mortal is a living being that will not be alive after the action $\textit{die}$. However, actions such as the $\textit{die}$ used here were only treated as modal word so that could not be further specified and interpreted. Furthermore, efficient decision algorithm for this logic is still an open problem.

Shi [7] proposed to embrace actions into description logics, in such a way that actions were treated as citizens of the combined logic, so that not only concepts with dynamic meanings could be described, but also actions happened in dynamic application domains could be represented over ontologies. Therefore, formalisms constructed with this approach would be more capable for semantic web.

In our previous work [3], a dynamic description logic was constructed with Shi’s approach; a terminating, sound, and complete tableau-based satisfiability-checking algorithm for this logic was also proposed. However, many restrictions were put on this logic, so that both concepts of the form $\neg <\pi> C$ and formulas of the form $\neg <\pi> \varphi$ were not allowed.

In the present paper, we further enrich that logic to incorporate concepts like $\neg <\pi> C$ and formulas like $\neg <\pi> \varphi$. The incorporation of such concepts and formulas disables the regression operator used in the tableau algorithm developed in [3]. Therefore, a prefixed tableau calculus for this enriched dynamic description logic is firstly developed. Based on this calculus, a satisfiability-checking algorithm is proposed.

Next section we firstly define the syntax and semantics of this enriched dynamic description logic. The prefixed tableau calculus and satisfiability-checking algorithm for the logic is presented in section 3, soundness and completeness of the algorithm are proved in section 4. Some applications of this logic are discussed in section 5. Section 6 concludes the paper.

2. Syntax and Semantics of Dynamic Description Logic

Primitive symbols of this logic are a set $\mathcal{N}_C$ of concept names, a set $\mathcal{N}_R$ of role names, and a set $\mathcal{N}_I$ of individual
names. Starting with these symbols, the concept, formula, and action are inductively defined with the help of a set of constructors.

Concepts are formed with the following syntax rule:

\[ C, C' \rightarrow C_i \{ p \} \otimes p_C \wedge |C| \cup C \cup \otimes R.C | < \pi > C \]

where \( C_i \in N_C, p \in N_I, R \in N_R, \) and \( \pi \) is an action.

We introduce \( C \cap C', \forall R.C, [\pi]C, \) and \( \top \) as abbreviations of \( \neg(C \cup \neg C'), \neg \exists R.C, \neg< \pi > \neg C \) and \( C \cup \neg C \) respectively.

Formulas are formed with the following syntax rule:

\[ \varphi, \varphi' \rightarrow C(p) \mid R(p, q) \mid \neg \varphi \mid \varphi' \mid < \pi > \varphi \]

where \( C \) is a concept, \( p, q \in N_I, R \in N_R, \) and \( \pi \) is an action.

We introduce \( \varphi \wedge \varphi', [\pi] \varphi, \varphi \rightarrow \varphi', \) and \( \text{true} \) as abbreviations of \( \neg(\neg \varphi \wedge \neg \varphi'), \neg< \pi > \neg \varphi, \neg \varphi \rightarrow \varphi, \) and \( \varphi \vee \neg \varphi \) respectively.

An atomic action is a pair \((P, E)\), where,

- \( P \) is a finite set of formulas, used for describing the pre-condition of the action,
- \( E \) is a finite set of effects, with each effect be of form \( A(p), \neg A(p), R(p, q), \) or \( \neg R(p, q), \) where \( A \in N_C, R \in N_R, \) and \( p, q \in N_I, \)
- let \( P=\{ \varphi_1, \ldots, \varphi_n \} \) and \( E=\{ \phi_1, \ldots, \phi_m \} \), then \( P \) and \( E \) subject to the constraint that \( \varphi_1 \wedge \ldots \wedge \varphi_n \rightarrow \neg \phi_k \)
  for all \( k \) with \( 1 \leq k \leq m. \)

Actions are formed with the following syntax rule:

\[ \pi, \pi' \rightarrow (P, E) \mid \varphi' \mid \pi \pi' \mid \pi \]

where \( (P, E) \) is an atomic action, \( \varphi \) is a formula.

Actions of the form \( \varphi, \pi \cup \pi' \) and \( \pi; \pi' \) are respectively named as testing, choice, and sequential actions.

A model for dynamic description logic is a pair \( M=(W, I) \), where \( W \) is a set of states, \( I \) associates with each state \( w \in W \) an interpretation \( I(w) = (\Delta^I, C_0^I(w), \ldots, R_0^I(w), \ldots, p_0^I, \ldots) \), with \( C_i^I(w) \subseteq \Delta^I \) for each \( C_i \in N_C, R_i^I(w) \subseteq \Delta^I \times \Delta^I \) for each \( R_i \in N_R, \) and \( p_i^I \in \Delta^I \) for each \( p_i \in N_I. \) Each action \( \pi \) is interpreted as a binary relation \( \pi^I \subseteq W \times W. \)

Given a model \( M=(W, I) \) and a state \( w \in W \), the value \( C_i^I(w) \) of a concept \( C_i \), the truth-relation \( (M, w) \models \varphi \) (or simply \( w \models \varphi \)) for a formula \( \varphi \), and the relation \( \pi^I \) for an action \( \pi \) are defined inductively as follows:

\begin{enumerate}
\item \( \{ p \}^I(w) = \{ p^I \}; \)
\item \( \text{If } p^I \in C_i^I(w) \text{ then } (\otimes p_C)^I(w) = \Delta^I, \text{ else } (\otimes p_C)^I(w) = \emptyset; \)
\item \( (\neg C)^I(w) = \Delta^I - C_i^I(w); \)
\item \( (C \cup D)^I(w) = C_i^I(w) \cup D^I(w); \)
\item \( (R.C)^I(w) = \{ x \} (x, y) \in R^I(w) \wedge y \in C_i^I(w) \}; \)
\item \( (\pi > C)^I(w) = \{ p \mid \exists w' \in W.((w, w') \in \pi^I \wedge p \in C_i^I(w')) \}; \)
\item \( (M, w) \models C(p) \text{ iff } p^I \in C_i^I(w); \)
\item \( (M, w) \models R(p, q) \text{ iff } (p^I, q^I) \in R^I(w); \)
\item \( (M, w) \models \neg \varphi \text{ iff } (M, w) \models \varphi \text{ not holds; } \)
\item \( (M, w) \models \varphi \lor \psi \text{ iff } (M, w) \models \varphi \text{ or } (M, w) \models \psi; \)
\item \( (M, w) \models < \pi > \varphi \text{ iff } \exists w' \in W.((w, w') \in \pi^I \wedge (M, w') \models \varphi); \)
\item \( (M, w) \models \exists \varphi \text{ iff } \forall \varphi_i \in S; \)
\end{enumerate}

A formula \( \varphi \) (or a formula set \( S \)) is satisfiable if and only if there is a model \( M = (W, I) \) and a state \( w \in W \) such that \( (M, w) \models \varphi \) (or \( (M, w) \models S \)).

The goal of the following two sections is to develop an algorithm for checking the satisfiability of formulas. Based on such an algorithm, many reasoning tasks on actions and concepts can be realized.

For simplicity, we will take the unique name assumption (UNA), i.e., for any model \( M = (W, I) \) and any names \( p_i, p_j \in N_i \) with \( p_i \neq p_j \), it is \( p_i^I \neq p_j^I \). Furthermore, we will not take into account TBoxes that composed of concept definitions \([1, 3]\).

3. Tableau algorithm

In this section we propose a satisfiability-checking algorithm for the dynamic description logic. The main idea is to introduce prefixes to indicate the relationships between action-related states, so that a prefixed tableau calculus for the logic can be developed. Then, a tableau algorithm based on the calculus can be easily presented.

A prefix \( \sigma \varepsilon \) is composed of a sequential action \( \sigma \) and a set of effects \( \varepsilon \), and are formed with the following syntax rule:

\[ \sigma \varepsilon \rightarrow (\emptyset, \emptyset) \sigma; (P, E), (\varepsilon - \{ \neg \varphi \mid \varphi \in E \}) \cup E \]

where \( (\emptyset, \emptyset) \) and \((P, E)\) are atomic actions, \( \sigma; (P, E) \) is a sequential action, \( (\varepsilon - \{ \neg \varphi \mid \varphi \in E \}) \cup E \) is a set of effects.
A prefixed formula is a pair $\sigma.\varepsilon : \varphi$, where $\sigma.\varepsilon$ is a prefix, $\varphi$ is a formula.

Prefixed tableau calculus for the dynamic description logic are presented in Figure 1, 2, 3, and 4. A rule from them could be applied in the case that the premise of this rule is hold.

Figure 1 shows the tableau rules for concepts. Rules $R_{\sim c}$, $R_{@ c}$, $R_{\sim C}$, $R_{C}$, and $R_{\sim C}$ are similar to traditional tableau rules for $\mathcal{ALCO}@\ominus$ concepts, except that prefixes are encoded here. With rules $R_{\sim c}$ and $R_{\sim C}$ actions occurred in a concept are shifted to a formula. Furthermore, it should be noted that rules $R_{a}$ and $R_{\sim C}$ are only applied to formulas prefixed by $(\emptyset, \emptyset, \emptyset)$.

Tableau rules for formulas are shown in Figure 2, where rules $R_{f}$, $R_{\sim f}$, $R_{\sim f}$, and $R_{\sim f}$ are similar to traditional tableau rules for $\mathcal{ALCO}@\ominus$ formulas. Rules $R_{\sim f}$, $R_{\sim f}$, $R_{\sim f}$, $R_{\sim f}$, and $R_{\sim f}$ respectively deal with the sequential, testing, and choice actions.

- \[ R_{\sim c} \text{ If } \sigma.\varepsilon : (\neg (C(x))) \in S \text{ and } \sigma.\varepsilon : (\neg (C)) \notin S, \text{ then } S_1 := (\sigma.\varepsilon : (\neg (C(x))) \cup S. \]
- \[ R_{@ c} \text{ If } \sigma.\varepsilon : (\theta \cap C) \in S \text{ and } \sigma.\varepsilon : (\theta) \notin S, \text{ then } S_1 := (\sigma.\varepsilon : (\theta) \cup S. \]
- \[ R_{\sim C} \text{ If } \sigma.\varepsilon : (\neg (\neg (C))) \in S \text{ and } \sigma.\varepsilon : (\neg (C)) \notin S, \text{ then } S_1 := (\sigma.\varepsilon : (\neg (C)) \cup S. \]
- \[ R_{C} \text{ If } \sigma.\varepsilon : (C(x)) \in S \text{ and } \sigma.\varepsilon : (C(x)) \notin S, \text{ then } S_1 := (\sigma.\varepsilon : (C(x)) \cup S. \]
- \[ R_{\sim f} \text{ If } \sigma.\varepsilon : (\pi \cap C) \in S \text{ and } \sigma.\varepsilon : (\pi) \notin S, \text{ then } S_1 := (\sigma.\varepsilon : (\pi) \cup S. \]
- \[ R_{\sim f} \text{ If } \sigma.\varepsilon : (\neg (\neg (C))) \in S \text{ and } \sigma.\varepsilon : (\neg (C)) \notin S, \text{ then } S_1 := (\sigma.\varepsilon : (\neg (C)) \cup S. \]
- \[ R_{\sim f} \text{ If } \sigma.\varepsilon : (\neg (\neg (C))) \in S \text{ and } \sigma.\varepsilon : (\neg (C)) \notin S, \text{ then } S_1 := (\sigma.\varepsilon : (\neg (C)) \cup S. \]
- \[ R_{\sim f} \text{ If } \sigma.\varepsilon : (\neg (\neg (C))) \in S \text{ and } \sigma.\varepsilon : (\neg (C)) \notin S, \text{ then } S_1 := (\sigma.\varepsilon : (\neg (C)) \cup S. \]

Figure 1: Rules for concepts

Rules presented in Figure 1 and Figure 2 only refer to a single prefix. For embodying the relationships between different prefixes, rules in Figure 3 and Figure 4 are designed.

Rules in Figure 3 embody the relationship with a forward approach: the rule $R_{\sim \text{atom} f}$ will generate new prefixes; Then, the rule $R_{\sim \text{atom} f}$ will add some constraints to these new generated prefixes.

\[ R_{f} \text{ If } \sigma.\varepsilon : (\neg (C(x))) \in S \text{ and } \sigma.\varepsilon : (\neg (C)) \notin S, \text{ then } S_1 := (\sigma.\varepsilon : (\neg (C(x))) \cup S. \]
\[ R_{\sim f} \text{ If } \sigma.\varepsilon : (\neg (\neg (C))) \in S \text{ and } \sigma.\varepsilon : (\neg (C)) \notin S, \text{ then } S_1 := (\sigma.\varepsilon : (\neg (C)) \cup S. \]
\[ R_{\sim f} \text{ If } \sigma.\varepsilon : (\neg (\neg (C))) \in S \text{ and } \sigma.\varepsilon : (\neg (C)) \notin S, \text{ then } S_1 := (\sigma.\varepsilon : (\neg (C)) \cup S. \]

Figure 2: Rules for formulas

Rules in Figure 4 embody the relationship with a back-
ward approach: if a primitive formula are generated and is not prefixed by \((\emptyset, \emptyset, \emptyset)\), then the rule \(R_{B1}\) will map the primitive formula backward to a formula prefixed by \((\emptyset, \emptyset, \emptyset)\); if a formula of the form \(\exists R.C\) or \(\neg\exists R.C\) are generated and is not prefixed by \((\emptyset, \emptyset, \emptyset)\), then the rule \(R_{B2}\) will map this formula backward to formulas prefixed by a preceding prefix.

\[
R_{B1} \quad \text{If } \sigma.e : \varphi \in S, \varphi \text{ is of form } R(x,y), \neg R(x,y), C(x), \text{ or } (\neg C)(x) \text{ with } C \in NCP, \text{ and } \varphi \notin \varepsilon, (\emptyset, \emptyset, \emptyset) : \varphi \notin S, \text{ then } S_1 := \{(\emptyset, \emptyset, \emptyset) : \varphi \} \cup S.
\]

\[
R_{B2} \quad \text{If } \sigma.e : D(x) \in S, D \text{ is of form } \exists R.C \text{ or } \neg \exists R.C \text{ with } C \text{ a concept, and } (\emptyset, \emptyset, \emptyset) : D^{\text{Regress}}(P,E)(x) \notin S, \text{ then } S_1 := \{(\emptyset, \emptyset, \emptyset) : D^{\text{Regress}}(P,E)(x)\} \cup S.
\]

Figure 4: Backward mapping rules

The concept \(D^{\text{Regress}}(P,E)\) appears in rule \(R_{B2}\) is calculated as follows:

**Algorithm 1** (\(D^{\text{Regress}}(P,E)\)). Let \(D\) be a concept, \((P,E)\) be an atomic action, and \(\text{Obj}(E)\) be all the individual names occurred in \(E\). The concept \(D^{\text{Regress}}(P,E)\) is calculated inductively as follows:

- For concept name \(C_i \in NC\), \(C_i^{\text{Regress}(P,E)} := C_i \cup \bigcup_{p \in I} \{p\} \cap \neg \bigcap_{C_i(p) \in E} \{p\}\):
- \(\{p\}^{\text{Regress}}(P,E) := \{p\}\):
- \((\emptyset)C^{\text{Regress}}(P,E) := \emptyset\):
- \((\neg C)^{\text{Regress}}(P,E) := \neg C^{\text{Regress}}(P,E)\):
- \((C \cup D)^{\text{Regress}}(P,E) := D^{\text{Regress}}(P,E) \cup D^{\text{Regress}}(P,E)\):
- \((\exists R.C)^{\text{Regress}}(P,E) := \bigcup_{p \in \text{Obj}(E)} \{p\} \cap \exists R.C^{\text{Regress}}(P,E)\):
- \((\exists R.C)^{\text{Regress}}(P,E) := \bigcup_{p \in \text{Obj}(E)} \{p\} \cap \exists R.C^{\text{Regress}}(P,E)\):
- \((\neg \exists R.C)^{\text{Regress}}(P,E) := \bigcup_{p \in \text{Obj}(E)} \{p\} \cap \neg \exists R.C^{\text{Regress}}(P,E)\):
- \((\neg \exists R.C)^{\text{Regress}}(P,E) := \bigcup_{p \in \text{Obj}(E)} \{p\} \cap \neg \exists R.C^{\text{Regress}}(P,E)\):
- \((< \pi) \cup \neg (\pi) C^{\text{Regress}}(P,E) := (\neg (\pi) C^{\text{Regress}}(P,E)\) (for the case \(\pi = \pi = C\)).

**Theorem 1.** For any model \(M = (W,I)\) and any states \(w, w' \in W\), if \((w, w') \in (P,E)^I\), then \(D^{\text{Regress}}(P,E)(w) = D^{\text{Regress}}(P,E)(w')\).

**Proof.** By structural induction on \(D\).

In the case that \(D\) is a concept name \(C_i\), since \(C_i^{\text{Regress}}(P,E)(w) = C_i^{\text{Regress}}(P,E)(w) \cup \{p | C_i(p) \in E\} - \{p | \neg C_i(p) \in E\}, C_i^{\text{Regress}}(P,E)(w) \cup \{p | C_i(p) \in E\} = \emptyset\), and \(\{p | \neg C_i(p) \in E\} \subseteq C_i^{\text{Regress}}(P,E)(w)\), we have \(C_i^{\text{Regress}}(P,E)(w) = C_i^{\text{Regress}}(P,E)(w) \cup \{p | C_i(p) \in E\} - \{p | \neg C_i(p) \in E\} \subseteq C_i^{\text{Regress}}(P,E)(w)\). Then, by substitution, it is an easy result that \(C_i^{\text{Regress}}(P,E)(w) = C_i^{\text{Regress}}(P,E)(w)\).

In these cases that \(D\) is \(\{p\}, \emptyset, \emptyset, -C, \emptyset \cup D\), the result is straightforward.

In the case that \(D\) is \(\exists R.C\), we can easily demonstrate that for any individuals \(x, y \in \Delta\) and any role name \(R \in N_R\), \((x, y) \in R^{\text{Regress}}(w')\) if any only if one of the following holds:

1. \(x \in \bigcap_{p \in \text{Obj}(E)} \{p\} \land (x, y) \in R^{\text{Regress}}(w')\).
2. \(\exists p \in \text{Obj}(E) \land (x = p \land y \in \bigcap_{q \in \text{Obj}(E)} \{q\} \land (x, y) \in R^{\text{Regress}}(w'))\).

Then, by substitution, it can be easy demonstrate that for any individual \(x \in (\exists R.C)^{\text{Regress}}(P,E)(w')\) if \(x \in (\exists R.C)^{\text{Regress}}(P,E)(w')\). So, \((\exists R.C)^{\text{Regress}}(P,E)(w) = (\exists R.C)^{\text{Regress}}(P,E)(w')\).

In the case that \(D\) is \(< \pi > C\), for any individual \(x\) we have:

\(x \in ((< \pi > C)^{\text{Regress}}(P,E)(w) \text{ if } x \in ((< \pi > C)^{\text{Regress}}(P,E)(w))\)

\(if \exists w'' \in W.( (w', w'') \in ((P,E) ; p | x \in C^{\text{Regress}}(w') ) \)
\n\(\text{iff } x \in ((< \pi > C)^{\text{Regress}}(P,E)(w') \). Therefore we have \((< \pi > C)^{\text{Regress}}(P,E)(w) = (\neg (\pi) C)^{\text{Regress}}(P,E)(w)\).

Based on the prefixed tableau calculus, we can utilize the following algorithm to decide the satisfiability of any formulas.

**Algorithm 2** (Deciding the satisfiability of a formula). For a formula \(\varphi\), decide its satisfiability with the following steps:

1. Construct a set \(S' := \{(\emptyset, \emptyset, \emptyset) : \varphi\}\). If \(S'\) contains clash, exit the algorithm with the result “\(\varphi\) is unsatisfiable”.

2. Construct a set \(SS := \{S'\}\).

3. Take out an element \(S\) from \(SS\), find a rule to apply to \(S\), with the order that rules in Figure 1 are firstly examined for application, then rules in Figure 2, then rules in Figure 4; Rules in Figure 3 will not be examined for application until no rules from Figure 1, 2 and 4 can be applied. For every new generated set, if it contains no clash, then add it into \(SS\).

4. Repeat step 3, until \(SS\) is empty or no rules can be applied to a set \(S\) that just taken out from \(SS\), in the former case return the result “\(\varphi\) is unsatisfiable”, in the latter case return the result “\(\varphi\) is satisfiable”.

77
Where, a clash in a set $S \in SS$ is one of the following cases: (i) $\sigma.\varepsilon : \varphi \in S$, $\sigma'.\varepsilon' : \neg \varphi \in S$ and $\varepsilon = \varepsilon'$ for a formula $\varphi$ and two prefixes $\sigma.\varepsilon$ and $\sigma'.\varepsilon'$; (ii) $\sigma.\varepsilon : C(p) \in S$, $\sigma'.\varepsilon : (\neg C)(p) \in S$ and $\varepsilon = \varepsilon'$ for a concept $C$, an individual name $p$, and two prefixes $\sigma.\varepsilon$ and $\sigma'.\varepsilon'$; (iii) $\sigma.\varepsilon : \{q\}(p) \in S$ for two different individual names $p, q$ and a prefix $\sigma.\varepsilon$; (iv) $\sigma.\varepsilon : (\neg \{q\})(p) \in S$ for an individual name $p$ and a prefix $\sigma.\varepsilon$.

4. Soundness and Completeness

In this section we prove the soundness and completeness of the tableau algorithm.

Firstly, we introduce a mapping function to map prefixes into states of a model:

**Definition 1.** Let $S$ be a set of prefixed formulas, $M = (W, I)$ be a model. Then, a mapping $\iota$ with respect to $S$ and $M$ is a function $\iota : S \rightarrow W$ from prefixes to states, such that for all prefixes $\sigma.\varepsilon$ and $\sigma'\varepsilon' : (\neg \varphi)(p) \in S$ and $\varphi \in \varepsilon'$ and $\varepsilon$.

Such a mapping function holds the following property:

**Theorem 2.** Let $S$ be a set of prefixed formulas, $M = (W, I)$ be a model, and $\iota$ be a mapping with respect to $S$ and $M$. Then, we have:

- $C^i(\sigma.\varepsilon) = C^i((\emptyset, \emptyset, \emptyset, \emptyset) \cup \{p^I \mid C(p) \in \varepsilon\} - \{p^I \mid \neg C(p) \in \varepsilon\}$ for each concept name $C \in \mathcal{C}$, and
- $R^i(\sigma.\varepsilon) = R^i((\emptyset, \emptyset, \emptyset, \emptyset) \cup \{(p^I, q^I) \mid R(p, q) \in \varepsilon\} - \{(p^I, q^I) \mid \neg R(p, q) \in \varepsilon\}$ for each role name $R \in \mathcal{R}$.

**Proof.** By induction on the construction of prefix $\sigma.\varepsilon$.

Base case: the result is obvious for the prefix $((\emptyset, \emptyset, \emptyset, \emptyset))$. Therefore, set $\Delta := \{p^I \mid p \in \mathcal{S}^I, p \in S\}$.

Induction step: assume the result holds for a prefix $\sigma.\varepsilon$. Then, for the prefix $\sigma'.\varepsilon'$ with $\sigma' = \sigma; (P, E)$ and $\varepsilon' = (\varepsilon - \{\neg \varphi\} \varphi \in \varepsilon) \cup \varepsilon$, we have:

(i) for each concept name $C \in \mathcal{C}$ it is:

$$C^i((\sigma'.\varepsilon')) = C^i((\emptyset, \emptyset, \emptyset, \emptyset)) \cup \{p^I \mid C(p) \in \varepsilon\} - \{p^I \mid \neg C(p) \in \varepsilon\}$$

(ii) for each role name $R \in \mathcal{R}$ it is:

$$R^i((\sigma'.\varepsilon')) = R^i((\emptyset, \emptyset, \emptyset, \emptyset)) \cup \{(p^I, q^I) \mid R(p, q) \in \varepsilon\} - \{(p^I, q^I) \mid \neg R(p, q) \in \varepsilon\}.$$

The following is an easy consequence:

**Corollary 1.** Let $\iota$ be a mapping with respect to a set $S$ and a model $M = (W, I)$, then:

- for any prefix $\sigma.\varepsilon$ and every formula $\varphi \in \varepsilon$, we have $\iota(\sigma.\varepsilon) = \varphi$;
- for any prefixes $\sigma.\varepsilon$ and $\sigma'.\varepsilon'$, $\iota(\sigma.\varepsilon) = \iota(\sigma'.\varepsilon')$ if and only if $\varepsilon = \varepsilon'$.

Secondly, we introduce the satisfiability of a set composed of prefixed formulas:

**Definition 2.** Let $S$ be a set of prefixed formulas. Then, $S$ is satisfiable if and only if there is a model $M = (W, I)$ and a mapping $\iota$ such that for every $\sigma.\varepsilon : \varphi \in S$ it is $(M, \iota(\sigma.\varepsilon)) = \varphi$.

Then, we can demonstrate that our tableau algorithm holds the following properties:

**Theorem 3.** For the set $SS$ constructed in Algorithm 2, let $SS'$ be the result of applying a tableau rule. Then, there is a satisfiable set $S$ in $SS$ if and only if there is a satisfiable set $S'$ in $SS'$.

**Proof.** The “if” direction is obvious because every new generated set subsumes the original set. The “only-if” direction can be proved by induction on rules applied. Due to space limitation, we omit the details here.

**Theorem 4.** Let $S$ be a set of prefixed formulas. Then, $S$ is satisfiable if no rules can be applied to it and no clashes are contained in it.

**Proof.** Let $\Sigma$ be all the prefixes present in $S$. Construct a model $M = (W, I)$ and a function $\iota() : \Sigma \rightarrow W$ as follows:

- For each individual name $p$, present in $S$, set $p^I := p$.
- Set the domain $\Delta := \{p^I \mid p \in \mathcal{S}^I, p \in S\}$.
- Construct a state $w_0$ in $W$ such that for each role name $R_i \in \mathcal{R}$ it is $R_i^{(w_0)} = \{(p^I, q^I) \mid (\emptyset, \emptyset) : R_i(p, q) \in S\}$, for each concept name $C_i \in \mathcal{C}$ it is $C_i^{(w_0)} := \{(p^I \mid (\emptyset, \emptyset) : C_i(p) \in S)\}$. Then, set $\iota((\emptyset, \emptyset, \emptyset, \emptyset)) := w_0$.
- For each prefix $\sigma_i, \varepsilon_i$ in $\Sigma$ construct a state $w_i$ in $W$, such that for each role name $R_i \in \mathcal{R}$ it is $R_i^{(w_i)} = \{(p^I, q^I) \mid (\emptyset, \emptyset) : R_i(p, q) \in \varepsilon_i\}$, for each concept name $C_i \in \mathcal{C}$ it is $C_i^{(w_i)} := \{(p^I \mid C_i(p) \in \varepsilon_i)\}$. Then, set $\iota(\sigma_i, \varepsilon_i) := w_i$.

It is obvious that $\iota$ is a mapping with respect to $S$ and $M$. Furthermore, by induction on the structure of $\varphi$, it is not difficult to demonstrate that for every $\sigma.\varepsilon : \varphi \in S$ it is $(M, \iota(\sigma.\varepsilon)) = \varphi$. Therefore, $S$ is satisfiable.

Finally, the following is an easy consequence of the definition of clash:

**Theorem 5.** Let $S$ be a set of prefixed formulas. Then, $S$ is unsatisfiable if it contains clash.
5. Application Perspective

A direct application of dynamic description logics is to describe concepts with dynamic meanings, just as the example stated in section 1. For reasoning tasks on concepts, the basic task is to decide whether a concept is satisfiable, i.e., for a concept $C$, whether there is a model $M = (W, I)$ and a state $w \in W$ such that $C^I(w)$ is nonempty. For this reasoning task, we only need to introduce an individual name $i$ and check whether the formula $C(i)$ is satisfiable.

Another practical application of dynamic description logics is to describe and reason about semantic web services [6]. More precisely, world-altering services [6] can be described as actions and be reasoned with this logic. Here we investigate two typical reasoning tasks.

One task is to check whether a composed service is realizable. With our formalism, this task can be described as: given a service $\pi$, decide whether there exist a model $M = (W, I)$ and two states $w, w' \in W$ such that $(w, w') \in \pi^I$. So, this task can be transformed into deciding whether the formula $<\pi\\true>$ is satisfiable.

The second task is to check whether an assertion really holds after executing a service under certain situation, i.e., the so-called projection problem. With our formalism, let the situation be described by a formula set $A$, $\pi$ be the service and $\varphi$ the formula, then, we need to check whether $A$ entails the formula $[\pi]\varphi$, which can be realized by deciding whether $A \cup \{<\pi\\true> \land \neg\varphi\}$ is unsatisfiable.

Active document framework (ADF), proposed by Zhuge [9] is a self-representable, self-explainable, and self-executable document mechanism, with a feature that the representation of the document content incorporates not only static knowledge, such as the ontology and semantic links, but also machine interpretable dynamic services. Therefore, the logic studied here could also be used for it.

An action formalism based on description logics was studied by Baader and used to describe and reason about web services [2]. Our approach is similar to Baader’s approach in the description and interpretation of atomic actions. The difference is that our formalism is a logical system with actions embraced in, so that reasoning tasks on actions can be described with the language provided by the logic and be realized with the help of the satisfiability-checking algorithm. Furthermore, testing actions and choice actions are modeled in our logical system.

6. Conclusion

In this paper we proposed a more powerful dynamic description logic and a sound and complete tableau-based satisfiability-checking algorithm for it. With this logic, atomic actions are described by their preconditions and effects, which are represented over ontologies expressed in description logic; Complex actions can be modeled with the help of the test, sequence, and choice operators. Both types of actions can be used as modal operators to construct concepts, so that concepts with dynamic meanings can be described. Furthermore, actions can also be used as modal operators for the construction of formulas, so that many reasoning tasks can be realized with the help of the satisfiability-checking algorithm for formulas, such as the satisfiability of concepts, the realizability of actions, and the projection problems on actions.

One future work is to further enrich this logic to embrace the iteration of actions. Another work is to evaluate and optimize the Tableau algorithm. Furthermore, it is also an valuable and interesting work to combine the reasoning mechanism provided by our logic with the analogical reasoning mechanism for the Knowledge Grid [10].

References