A Logic For Decidable Reasoning About Services

Yilan Gu
Dept. of Computer Science
University of Toronto
10 King's College Road
Toronto, ON, M5S 3G4, Canada
Email: yilan@cs.toronto.edu

Mikhail Soutchanski
Department of Computer Science
Ryerson University
245 Church Street, ENG281
Toronto, ON, M5B 2K3, Canada
Email: mes@scs.ryerson.ca

Abstract

We consider a modified version of the situation calculus built using a two-variable fragment of the first-order logic extended with counting quantifiers. We mention several additional groups of axioms that need to be introduced to capture taxonomic reasoning. We show that the regression operator in this framework can be defined similarly to regression in the Reiter's version of the situation calculus. Using this new regression operator, we show that the projection problem (that is the main reasoning task in the situation calculus) is decidable in the modified version. We mention possible applications of this result to formalization of Web services and to reasoning about effects of composite Web services.

Introduction

The Semantic Web community makes significant efforts toward integration of Semantic Web technology with the ongoing work on web services. These efforts include use of semantics in the discovery, composition, and other aspects of web services. Web service *composition* is related to the task of designing a suitable combination of available component services into a composite service to satisfy a client request when there is no single service that can satisfy this request (Hull & Su 2005). This problem attracted significant attention of researchers both in academia and in industry. A major step in this direction is creation of ontologies for web services, in particular, OWL-S that models web services as atomic or complex actions with preconditions and effects. An emerging industry standard BPEL4WS (Business Process Execution Language for Web Services) provides the basis for manually specifying composite web services using a procedural language. However, in comparison to error-prone manual service compositions, (semi)automated service composition promises significant flexibility in dealing with available services and also accommodates naturally the dynamics and openness of service-oriented architectures. The problem of the automated composition of web services is often formulated in terms similar to a planning problem in AI: given a description of a client goal and a set of component services (that can be atomic or complex), find a composition of services that achieves the goal (McIlraith & Son 2002; Narayanan & McIlraith 2003; Sirin *et al.* 2004). Despite that several approaches to solving this problem have already been proposed, many issues remain to be resolved, e.g., how to give well-defined and general characterizations of service compositions, how to compute all effects and side-effects on the world of every action included in composite service, and other issues. Other reasoning problems, well-known in AI, that can be relevant to service composition and discovery are executability and projection problems. Executability problem requires determining whether preconditions of all actions included in a composite service can be satisfied given incomplete information about the world. Projection problem requires determining whether a certain goal condition is satisfied after the execution of all component services given an incomplete information about the current state. In this paper we would like to concentrate on the last problem because it is an important prerequisite for planning and execution monitoring tasks, and for simplicity we start with sequential compositions of the atomic actions (services) only (we mention complex actions in the last section). More specifically, following several previous approaches (McIlraith & Son 2002; Narayanan & McIlraith 2003; Berardi *et al.* 2003; Sirin *et al.* 2004; Hull & Su 2005), we choose the situation calculus as an expressive formal language for specification of actions. However, we acknowledge openness of the world and represent incomplete information about an initial state of the world by assuming that it is characterized by a predicate logic theory in the general syntactic form.

The situation calculus is a popular and well understood predicate logic language for reasoning about actions and their effects (Reiter 2001). It serves as a foundation for the Process Specification Language (PSL) that axiomatizes a set of primitives adequate for describing the fundamental concepts of manufacturing processes (PSL has been accepted as an international standard) (Grüninger & Menzel 2003; Grüninger 2004). It is used to provide a well-defined semantics for Web services and a foundation for a high-level programming language Golog (Berardi *et al.* 2003; McIlraith & Son 2002; Narayanan & McIlraith 2003; Pistore *et al.* 2005). However, because the situation calculus is formulated in a general predicate logic, reasoning about effects of sequences of actions is undecidable (unless some restrictions are imposed on the theory that axiomatizes the initial state of the world). The first motivation for our paper is intention to overcome this difficulty. We propose to use a two-variable fragment FO^2 of the first-order logic as a foundation for a modified situation calculus. Because the satisfiability problem in this fragment is known to be decidable (it is in NEXPTIME), we demonstrate that by reducing reasoning about effects of actions to reasoning in this fragment, one can guarantee decidability no matter what is the syntactic form of the theory representing the initial state of the world.

The second motivation for our paper comes from description logics. Description Logics (DLs) (Baader *et al.* 2003) are a well-known family of knowledge representation formalisms, which play an important role in providing the formal foundations of several widely used Web ontology languages including OWL (Horrocks, Patel-Schneider, & van Harmelen 2003) in the area of the Semantic Web (Baader, Horrocks, & Sattler 2005). DLs may be viewed as syntactic fragments of first-order logics (FOL) and offer considerable expressive power going far beyond propositional logic, while ensuring that reasoning is decidable (Borgida 1996). DLs have been mostly used to describe static knowledge-base systems. Moreover, several research groups consider formalization of actions using DLs or extensions of DLs. Following the key idea of (Giacomo 1995), that reasoning about complex actions can be carried in a fragment of the propositional situation calculus, De Giacomo et al. (Giacomo *et al.* 1999) give an epistemic extension of DLs to provide a framework for the representation of dynamic systems. However, the representation and reasoning about actions in this framework are strictly propositional, which reduces the representation power of this framework. In (Baader *et al.* 2005), Baader et al. provide another proposal for integrating description logics and action formalisms. They take as foundation the well known description logic $\mathcal{ALCQIO}$ (and its sub-languages) and show that the complexity of executability and projection problems coincides with the complexity of standard DL reasoning. However, actions (services) are represented in their paper meta-theoretically, not as first-order terms. This can potentially lead to some complications when specifications of other reasoning tasks (e.g., planning) will be considered because it is not possible to quantify over actions in their framework. In our paper, we take a different approach and represent actions as first-order terms, but achieve integration of taxonomic reasoning and reasoning about actions by restricting the syntax of the situation calculus. Our paper can be considered as a direct extension of the well-known result of Borgida (Borgida 1996) who proves that many expressive description logics can be translated to two-variable fragment FO^2 of FOL. However, to the best of our knowledge, nobody proposed this extension before.

The main contribution of our paper to the area of service composition and discovery is the following. We show that by using services that are composed from atomic services with no more than two parameters and by using only those properties of the world which have no more than two parameters (to express a goal condition), one can guarantee that the executability and projection problems for these services can always be solved even if information about the current state of the world is incomplete.

Motivations

Consider online web services provided by an university. Imagine that a system automates the department administrators by doing student management work online, for instance, admitting new students, accepting payments of tuition fees and doing course enrollments for students, etc. Unlike previously proposed e-services (e.g.,the e-services described in (Berardi *et al.* 2003) or in BPEL4WS)

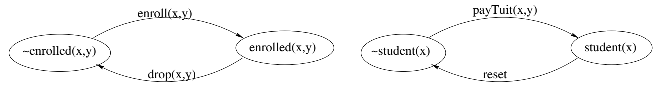

Figure 1: Examples of transition diagrams for services.

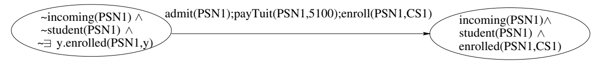

Figure 2: A transition diagram for a composite web service.

which allow only services without parameters, we use functional symbols to represent a class of services. For example, variables, say x and y, can be used to represent any objects; the service of enrolling any student x in any course y can be specified by using a functional symbol $enroll(x, y)$; and, the service of admitting any student x can be represented as a functional symbol $admit(x)$, etc. The composite web services can be considered as sequences of instantiated services. For example, a sequence $admit(PSN_1); payTuit(PSN_1, 5100); enroll(PSN_1, CS_1)$ represents the following composite web service for person PSN_1: admit her as a student, take the tuition fee \$5100 and enroll her in a course CS_1. The system properties are specified by using predicates with parameters. For example, the predicate $enrolled(x, y)$ represents that a student x is enrolled in a course y. This property becomes true when service $enroll(x, y)$ is performed and becomes false when service $drop(x, y)$ is performed for a student x and a course y (see Figure 1). A composite web service corresponds to the composition of these instantiated transition diagrams (see Figure 2). When one describes the preconditions of the services, the effects of the services on the world, i.e., when one characterizes which properties of the world are true before and after the execution of the services, given incomplete information about the current state of the world, the use of first-order language, such as the situation calculus (Reiter 2001), can provide more expressive power than propositional languages. For example, assume that a student is considered as a qualified full time student if the tuition fee she paid is more than 5000 dollars and she enrolls in at least four different courses in the school. Such property can be easily described using the first-order logic, and checking whether or not such property can be satisfied after execution of certain sequence of web services is equivalent to solving a projection problem. Because FOL is compact way of representing information about states and transitions between states, we want to take advantage of the expressive power of the first-order logic as much as possible to reason about web services.

On the other hand, as we mentioned in the introduction, we want to avoid the undecidability of the entailment problem in the general FOL. Inspired by the decidability of reasoning in many DLs (which are sub-languages of a syntactic fragment of the FOL with the restriction on the number of variables), we restrict the number of variables to at most two in the specifications of the web services to ensure the decidability of the executability and projection problems techniques. At the same time, we can take the advantage of the expressive power of quantifiers to specify compactly realistic web services (such as mentioned above). Moreover, FOL with limited number of variables, in contrast to the propositional logic, still allows us to represent and reason about properties with infinite domains (such as $weight$ and $time$, etc) or with large finite domains (such as $money$, $person$, etc) in a very compact way. Two examples are given in the last section to illustrate the expressive power and reasoning about the web services.

The Situation Calculus

The situation calculus (SC) $\mathcal{L}_{sc}$ is a first-order (FO) language for axiomatizing dynamic systems. In recent years, it has been extended to include procedures, concurrency, time, stochastic actions, etc (Reiter 2001). Nevertheless, all dialects of the SC $\mathcal{L}_{sc}$ include three disjoint sorts (*actions*, *situations* and *objects*). **Actions** are first-order terms consisting of an action function symbol and its arguments. Actions change the world. **Situations** are first-order terms which denote possible world histories. A distinguished constant S_0 is used to denote the *initial situation*, and function $do(a, s)$ denotes the situation that results from performing action a in situation s. Every situation corresponds uniquely to a sequence of actions. Moreover, notation $s' \preceq s$ means that either situation s' is a subsequence of situation s or $s = s'$.[1] **Objects** are first-order terms other than actions and situations that depend on the domain of application. **Fluents** are relations or functions whose values may vary from one situation to the next. Normally, a fluent is denoted by a predicate or function symbol whose last argument has the sort situation. For example, $F(\vec{x}, do([\alpha_1, \cdots, \alpha_n], S_0)$ represents a relational fluent in the situation $do(\alpha_n, do(\cdots, do(\alpha_1, S_0)\cdots)$ resulting from execution of ground action terms $\alpha_1, \cdots, \alpha_n$ in S_0. We do not consider functional fluents in this paper.

The SC includes the distinguished predicate $Poss(a, s)$ to characterize actions a that are possible to execute in s. For any SC formula ϕ and a term s of sort situation, we say ϕ is a formula *uniform* in s iff it does not mention the predicates $Poss$ or $\prec$, it does not quantify over variables of sort situation, it does not mention equality on situations, and whenever it mentions a term of sort situation in the situation argument position of a fluent, then that term is s (see (Reiter 2001)). If $\phi(s)$ is a uniform formula and the situation argument is clear from the context, sometimes we suppress the situation argument and write this formula simply as ϕ. Moreover, for any predicate with the situation argument, such as a fluent F or $Poss$, we introduce an operation of restoring a situation argument s back to the corresponding atomic formula without situation argument, i.e., $F(\vec{x})[s] \stackrel{def}{=} F(\vec{x}, s)$ and $Poss(A)[s] \stackrel{def}{=} Poss(A, s)$ for any action term A and object vector $\vec{x}$. By the recursive definition, such notation can be easily extended to $\phi[s]$ for any first-order formula ϕ, in which the situation arguments of all fluents and $Poss$ predicates are left out, to represent the SC formula obtained by restoring situation s back to all the fluents and/or $Poss$ predicates (if any) in ϕ. It is obvious that $\phi[s]$ is uniform in s.

A *basic action theory* (BAT) $\mathcal{D}$ in the SC is a set of axioms written in $\mathcal{L}_{sc}$ with the following five classes of axioms to model actions and their effects (Reiter 2001). **Action precondition axioms $\mathcal{D}_{ap}$:** For each action function $A(\vec{x})$, there is one axiom of the form $Poss(A(\vec{x}), s) \equiv \Pi_A(\vec{x}, s)$. $\Pi_A(\vec{x}, s)$ is a formula uniform in s with free variables among $\vec{x}$ and s, which characterizes the preconditions of action A. **Successor state axioms $\mathcal{D}_{ss}$:** For each relational fluent $F(\vec{x}, s)$, there is one axiom of the form $F(\vec{x}, do(a, s)) \equiv \Phi_F(\vec{x}, a, s)$, where $\Phi_F(\vec{x}, a, s)$ is a formula uniform in s with free variables among $\vec{x}, a$ and s. The successor state axiom (SSA) for fluent F completely characterizes the value of fluent F in the next situation $do(a, s)$ in terms of the current situation s. **Initial theory $\mathcal{D}_{S_0}$:** It is a set of first-order formulas whose only situation term is S_0. It specifies the values of all fluents in the initial state. It also describes all the facts that are not changeable by any actions in the domain. **Unique name axioms for actions $\mathcal{D}_{una}$:** Includes axioms specifying that two actions are different if their action names are different, and identical actions have identical arguments. **Fundamental axioms for situations Σ:** The axioms for situations which characterize the basic properties of situations. These axioms are domain independent. They are included in the axiomatization of any dynamic systems in the SC (see (Reiter 2001) for details).

Suppose that $\mathcal{D} = \mathcal{D}_{una} \cup \mathcal{D}_{S_0} \cup \mathcal{D}_{ap} \cup \mathcal{D}_{ss} \cup \Sigma$ is a BAT, $\alpha_1, \cdots, \alpha_n$ is a sequence of ground action terms, and $G(s)$ is a uniform formula with one free variable s. One of the most important reasoning tasks in the SC is the projection problem, that is, to determine whether $\mathcal{D} \models G(do([\alpha_1, \cdots, \alpha_n], S_0))$. Another basic reasoning task is the executability problem. Let $executable(do([\alpha_1, \cdots, \alpha_n], S_0))$ be an abbreviation of the formula $Poss(\alpha_1, S_0) \wedge \bigvee_{i=2}^{n} Poss(\alpha_i, do([\alpha_1, \cdots, \alpha_{i-1}], S_0))$. Then, the executability problem is to determine whether $\mathcal{D} \models executable(do([\alpha_1, \cdots, \alpha_n], S_0))$. Planning and high-level program execution are two important settings where the executability and projection problems arise naturally. *Regression* is a central computational mechanism that forms the basis for automated solution to the executability and projection tasks in the SC (Reiter 2001). A recursive definition of the regression operator $\mathcal{R}$ on any *regressable formula* ϕ is given in (Reiter 2001); we use notation $\mathcal{R}[\phi]$ to denote the formula that results from eliminating $Poss$ atoms in favor of their definitions as given by action precondition axioms and replacing fluent atoms about $do(\alpha, s)$ by logically equivalent expressions about s as given by SSAs repeatedly until it cannot make such replacement any further. A formula W of $\mathcal{L}_{sc}$ is *regressable* iff (1) every term of sort situation in W is starting from S_0 and has the syntactic form $do([\alpha_1, \cdots, \alpha_n], S_0)$ where each α_i is of sort action; (2) for every atom of the form $Poss(\alpha, \sigma)$ in W, α has the syntactic form $A(t_1, \cdots, t_n)$ for some n-ary function symbol A of $\mathcal{L}_{sc}$; and (3) W

[1] Reiter (Reiter 2001) uses the notation $s' \sqsubseteq s$, but we use $s' \preceq s$ to avoid confusion with the inclusion relation $\sqsubset$ that is commonly used in description logic literature. In this paper, we use $\sqsubset$ to denote the inclusion relation between concepts or roles.

does not quantify over situations, and does not mention the relation symbols "$\prec$" or "=" between terms of situation sort. The formula $G(do([\alpha_1, \cdots, \alpha_n], S_0))$ is a particularly simple example of a regressable formula because it is uniform in $do([\alpha_1, \cdots, \alpha_n], S_0))$, but in the general case, regressable formulas can mention several different ground situation terms. Roughly speaking, the regression of a regressable formula ϕ through an action a is a formula ϕ' that holds prior to a being performed iff ϕ holds after a. Both precondition and SSAs support regression in a natural way and are no longer needed when regression terminates. The regression theorem (Reiter 2001) shows that one can reduce the evaluation of a regressable formula W to a first-order theorem proving task in the initial theory together with unique names axioms for actions:

$$\mathcal{D} \models W \text{ if } \mathcal{D}_{S_0} \cup \mathcal{D}_{una} \models \mathcal{R}[W].$$

This fact is the key result for our paper. It demonstrates that an executability or a projection task can be reduced to a theorem proving task that does not use precondition, successor state, and foundational axioms. This is one of the reasons why the SC provides a natural and easy way to representation and reasoning about dynamic systems. However, because $\mathcal{D}_{S_0}$ is an arbitrary first-order theory, this type of reasoning in the SC is undecidable. One of the common ways to overcome this difficulty is to introduce the closed world assumption that amounts to assuming that $\mathcal{D}_{S_0}$ is a relational theory and all statements that are not known to be true explicitly, are assumed to be false. However, in many application domains this assumption is unrealistic. For this reason, we would like to consider a version of the SC formulated in FO^2, a syntactic fragment of the first-order logic that is known to be decidable, or in C^2 an extension of FO^2 (see below), where the satisfiability problem is still decidable.

Description Logics and Two-variable FO Logics

In this section we review a few popular expressive description logics and related fragments of the FO logic. We start with logic $\mathcal{ALCHQI}$. Let $N_C = \{C_1, C_2, \ldots\}$ be a set of atomic *concept names* and $N_R = \{R_1, R_2, \ldots\}$ be a set of atomic *role names*. A $\mathcal{ALCHQI}$ role is either some $R \in N_R$ or an *inverse role* R^- for $R \in N_R$. A $\mathcal{ALCHQI}$ role hierarchy (RBox) $\mathcal{RH}$ is a finite set of role inclusion axioms $R_1 \sqsubseteq R_2$, where R_1, R_2 are $\mathcal{ALCHQI}$ roles. For $R \in N_R$, we define $\mathsf{Inv}(R) = R^-$ and $\mathsf{Inv}(R^-) = R$, and assume that $R_1 \sqsubseteq R_2 \in \mathcal{RH}$ implies $\mathsf{Inv}(R_1) \sqsubseteq \mathsf{Inv}(R_2) \in \mathcal{RH}$.

The set of $\mathcal{ALCHQI}$ concepts is the minimal set built inductively from N_C and $\mathcal{ALCHQI}$ roles using the following rules: all $A \in N_C$ are concepts, and, if C, C_1, and C_2 are $\mathcal{ALCHQI}$ concepts, R is a simple role and $n \in \mathbb{N}$, then also $\neg C$, $C_1 \sqcap C_2$, and $(\exists^{\geqslant n} R.C)$ are $\mathcal{ALCHQI}$ concepts. We use also some abbreviations for concepts:

$C_1 \sqcup C_2 \overset{def}{=} \neg(\neg C_1 \sqcap \neg C_2)$	$(\exists^{\leqslant n} R.C) \overset{def}{=} \neg(\exists^{\geqslant (n+1)} R.C)$
$C_1 \supset C_2 \overset{def}{=} \neg C_1 \sqcup C_2$	$(\exists^{n} R.C) \overset{def}{=} (\exists^{\leqslant n} R.C) \sqcap (\exists^{\geqslant n} R.C)$
$\exists R.C \overset{def}{=} (\exists^{\geqslant 1} R.C)$	$\top \overset{def}{=} A \sqcup \neg A$ for some $A \in N_C$
$\forall R.C \overset{def}{=} \exists^{<1} R.\neg C$	$\bot \overset{def}{=} \neg \top$

Concepts that are not concept names are called *complex*. A *literal* concept is a possibly negated concept name. A TBox $\mathcal{T}$ is a finite set of *equality* axioms $C_1 \equiv C_2$ (sometimes, *general inclusion axioms* of the form $C_1 \sqsubseteq C_2$ are also allowed, where C_1, C_2 are complex concepts). An equality with an atomic concept in the left-hand side is a concept *definition*. In the sequel, we always consider TBox axioms set $\mathcal{T}$ that is a *terminology*, a finite set of concept definition formulas with unique left-hand sides. We say that a *defined* concept name C_1 *directly uses* a concept name C_2 w.r.t. $\mathcal{T}$ if C_1 is defined by a concept definition axiom in $\mathcal{T}$ with C_2 occurring in the right-hand side of the axiom. Let *uses* be the transitive closure of directly uses, and a TBox axioms set $\mathcal{T}$ is *acyclic* if no concept name uses itself w.r.t. $\mathcal{T}$. An ABox $\mathcal{A}$ is a finite set of axioms $C(a)$, $R(a, b)$, and (in)equalities $a \approx b$ and $a \not\approx b$.

The logic $\mathcal{ALCQI}$ is obtained by disallowing RBox . A more expressive logic $\mathcal{ALCQI}(\sqcup, \sqcap, \neg, |, id)$ is obtained from $\mathcal{ALCQI}$ by introducing identity role id (relating each individual with itself) and allowing complex role expressions: if R_1, R_2 are $\mathcal{ALCQI}(\sqcup, \sqcap, \neg, |, id)$ roles and C is a concept, then $R_1 \sqcup R_2$, $R_1 \sqcap R_2$, $\neg R_1$, R_1^- and $R_1|_C$ are $\mathcal{ALCQI}(\sqcup, \sqcap, \neg, |, id)$ roles too.[2] These complex roles can be used in TBox (in the right-hand sides of definitions). Subsequently, we call a role R *primitive* if it is either $R \in N_R$ or it is an *inverse role* R^- for $R \in N_R$. *Two-variable FO logic*

$\tau_x(A) \overset{def}{=} A(x)$	for $A \in N_C$
$\tau_x(\top) \overset{def}{=} x = x$	$\tau_x(\bot) \overset{def}{=} \neg(x = x)$
$\tau_x(\neg C) \overset{def}{=} \neg \tau_x(C)$	$\tau_y(C) \overset{def}{=} \tau_x(C)[x/y, y/x]$
$\tau_x(C_1 \sqcap C_2) \overset{def}{=} \tau_x(C_1) \wedge \tau_x(C_2)$	
$\tau_x(\exists^{\bowtie n} R.C) \overset{def}{=} \exists^{\bowtie n} y.(\,\tau_{x,y}(R) \wedge \tau_y(C)\,)$	
$\tau_x(\forall R.C) \overset{def}{=} \forall y.(\,\tau_{x,y}(R) \supset \tau_y(C)\,)$	
$\tau_{x,y}(id) \overset{def}{=} x = y$	$\tau_{x,y}(\neg R) \overset{def}{=} \neg \tau_{x,y}(R)$
$\tau_{x,y}(R\|_C) \overset{def}{=} \tau_{x,y}(R) \wedge \tau_y(C)$	$\tau_{x,y}(R^-) \overset{def}{=} \tau_{y,x}(R)$
$\tau_{x,y}(R_1 \sqcap R_2) \overset{def}{=} \tau_{x,y}(R_1) \wedge \tau_{x,y}(R_2)$	
$\tau_{x,y}(R_1 \sqcup R_2) \overset{def}{=} \tau_{x,y}(R_1) \vee \tau_{x,y}(R_2)$	
$\tau_{x,y}(R) \overset{def}{=} R(x, y)$	for $R \in N_R$
$\tau_{y,x}(R) \overset{def}{=} R(y, x)$	for $R \in N_R$

FO^2 is the fragment of ordinary FO logic (with equality), whose formulas only use no more than two variable symbols x and y (free or bound). *Two-variable FO logic with counting* C^2 extends FO^2 by allowing FO counting quantifiers $\exists^{\geq m}$ and $\exists^{\leq m}$ for all $m \geq 1$. Borgida in (Borgida 1996) defines an expressive description logic $\mathcal{B}$ and shows that each sentence in the language $\mathcal{B}$ without transitive roles and role-composition operator can be translated to a sentence in C^2 with the same meaning, and vice versa, i.e., these two languages are *equally expressive*. A knowledge base KB is a triple $(\mathcal{R}, \mathcal{T}, \mathcal{A})$. The semantics of KB is given by translating it into FO logic with counting C^2 by the operator τ (see the table above, in which $\bowtie \in \{\geqslant, \leqslant\}$ and x/y means replace x with y). Borgida's logic $\mathcal{B}$ includes all

[2]These standard roles constructors and their semantics can be found in (Baader, Horrocks, & Sattler 2005).

concept and role constructors in $\mathcal{ALCQI}(\sqcup, \sqcap, \neg, |, id)$ and, in addition, it includes a special purpose constructor *product* that allows to build the role $C_1 \times C_2$ from two concepts C_1 and C_2. This construct has a simple semantics $\tau_{x,y}(C_1 \times C_2) \stackrel{def}{=} \tau_x(C_1) \wedge \tau_y(C_2)$, and makes the translation from C^2 into $\mathcal{B}$ rather straightforward. Although constructor *product* is not a standard role constructor, we can use restriction constructor $|$ in addition with $\sqcup, \sqcap, \neg$ and inverse role to represent it. That is, for any concepts C_1 and C_2, $\quad C_1 \times C_2 = (R \sqcup \neg R)|_{C_2} \sqcap ((R \sqcup \neg R)|_{C_1})^-$,
where R can be any role name. Consequently, product can be eliminated. Therefore, the following statement is a direct consequence of the theorems proved in (Borgida 1996).

Theorem 1 *The description logic $\mathcal{ALCQI}(\sqcup, \sqcap, \neg, |, id)$ and C^2 are equally expressive (i.e., each sentence in language $\mathcal{ALCQI}(\sqcup, \sqcap, \neg, |, id)$ can be translated to a sentence in C^2, and vice versa). In addition, translation in both directions leads to no more than linear increase of the size of the translated formula.*

This statement has an important consequence. Gradel et. al.(Grädel, Otto, & Rosen 1997) and Pacholski et al (Pacholski, Szwast, & Tendera 1997) show that satisfiability problem for C^2 is decidable. Hence, the satisfiability and/or subsumption problems of concepts w.r.t. an acyclic or empty TBox in description logic $\mathcal{ALCQI}(\sqcup, \sqcap, \neg, |, id)$ is also decidable.[3] In Section , we take advantage of this and use C^2 as a foundation for a modified SC.

Modeling Dynamic Systems in a Modified Situation Calculus

In this section, we consider dynamic systems formulated in a minor modification of the language of the SC so that it can be considered as an extension to C^2 language (with situation argument for unary and binary fluents). The key idea is to consider a syntactic modification of the SC such that the executability and projection problems are guaranteed to be decidable as a consequence of the C^2 property of being decidable.[4] Moreover, since the modified SC has a very strong connections with description logics, which will be explained in detail below, we will denote this language as $\mathcal{L}_{sc}^{DL}$.

First of all, the three sorts in $\mathcal{L}_{sc}^{DL}$ (i.e., actions, situations and objects) are the same as those in $\mathcal{L}_{sc}$, except that they obey the following restrictions: (1) all terms of sort *object* are variables (x and y) or constants, i.e., functional symbols are not allowed; (2) all action functions include no more than two arguments. Each argument of any term of sort *action* is either a constant or an *object* variable (x or y); (3) variable symbol a of sort *action* and variable symbol s of sort *situation* are the only additional variable symbols being allowed in $\mathcal{L}_{sc}^{DL}$ in addition to variable symbols x and y.

Second, any fluent in $\mathcal{L}_{sc}^{DL}$ is a predicate either with two or with three arguments including the one of sort situation. We call fluents with two arguments, one is of sort object and the other is of sort situation, *(dynamic) concepts*, and call fluents with three arguments, first two of sort object and the last of sort situation, *(dynamic) roles*. Intuitively, each (dynamic) concept in $\mathcal{L}_{sc}^{DL}$, say $F(x, s)$ with variables x and s only, can be considered as a changeable concept F in a dynamic system specified in $\mathcal{L}_{sc}^{DL}$; the truth value of $F(x, s)$ could vary from one situation to another. Similarly, each (dynamic) role in $\mathcal{L}_{sc}^{DL}$, say $R(x, y, s)$ with variables x, y and s, can be considered as a changeable role R in a dynamic system specified in $\mathcal{L}_{sc}^{DL}$; the truth value of $R(x, y, s)$ could vary from one situation to another. In $\mathcal{L}_{sc}^{DL}$, *(static) concepts* (i.e., unary predicates with no situation argument) and *(static) roles* (i.e., binary predicates with no situation argument), if any, are considered as eternal facts and their truth values never change. They represent unchangeable taxonomic properties and unchangeable classes of an application domain. Moreover, each concept (static or dynamic) can be either *primitive* or *defined*. For each primitive dynamic concept, an SSA must be provided in the basic action theory formalized for the given system. Because defined dynamic concepts are expressed in terms of primitive concepts by axioms similar to TBox , SSAs for them are not provided. In addition, SSAs are provided for dynamic primitive roles.

Third, apart from standard first-order logical symbols $\wedge$, $\vee$ and $\exists$, with the usual definition of a full set of connectives and quantifiers, $\mathcal{L}_{sc}^{DL}$ also includes counting quantifiers $\exists^{\geq m}$ and $\exists^{\leq m}$ for all $m \geq 1$.

The dynamic systems we are dealing with here satisfy the *open world assumption* (OWA): what is not stated explicitly is currently unknown rather than false. In this paper, the dynamic systems we are interested in can be formalized as a *basic action theory* (BAT) $\mathcal{D}$ using the following seven groups of axioms in $\mathcal{L}_{sc}^{DL}$: $\mathcal{D} = \Sigma \cup \mathcal{D}_{ap} \cup \mathcal{D}_{ss} \cup \mathcal{D}_T \cup \mathcal{D}_R \cup \mathcal{D}_{una} \cup \mathcal{D}_{S_0}$. Five of them $(\Sigma, \mathcal{D}_{ap}, \mathcal{D}_{ss}, \mathcal{D}_{una}, \mathcal{D}_{S_0})$ are similar to those groups in a BAT in $\mathcal{L}_{sc}$, and the other two $(\mathcal{D}_T, \mathcal{D}_R)$ are introduced to axiomatize description logic related facts and properties (see below). However, because $\mathcal{L}_{sc}^{DL}$ allows only two object variables, all axioms must conform to the following additional requirements.

Action precondition axioms $\mathcal{D}_{ap}$: For each action A in $\mathcal{L}_{sc}^{DL}$, there is one axiom of the form
$Poss(A, s) \equiv \Pi_A[s]$ (or $Poss(A(x), s) \equiv \Pi_A(x)[s]$,
or $Poss(A(x, y), s) \equiv \Pi_A(x, y)[s]$, respectively),
if A is an action constant (or unary, or binary action term, respectively), where Π_A (or $\Pi_A(x)$, or $\Pi_A(x, y)$, respectively) is a C^2 formula with no free variables (or with at most x, or with at most x, y as the only free variables, respectively). These axioms characterize the preconditions of all actions.

Successor state axioms $\mathcal{D}_{ss}$: For each primitive dynamic concept $F(x, s)$ in $\mathcal{L}_{sc}^{DL}$, an SSA is specified for $F(x, do(a, s))$. According to the general syntactic form of the SSAs provided in (Reiter 2001), without loss of general-

[3] In (Baader, Horrocks, & Sattler 2005) it is shown that the satisfiability problems of concepts and subsumption problems of concepts can be reduced to each other; moreover, if a TBox $\mathcal{T}$ is acyclic, the reasoning problems w.r.t. $\mathcal{T}$ can always be reduced to problems w.r.t. the empty TBox.

[4] The reason that we call it a "modified" rather than a "restricted" is that we not only restrict the number of variables that can be mentioned in the theories, but we also extend the SC with other features, such as introducing counting quantifiers and adding acyclic TBox axioms to basic action theories.

ity, we assume the axiom is of the form

$$F(x, do(a, s)) \quad \equiv \quad \psi_F(x, a, s), \tag{1}$$

where the general structure of $\psi_F(x, a, s)$ is
$(\bigvee_{i=1}^{m_0} [\exists x][\exists y](a = A_i^+(\vec{x}_{(i,0,+)}) \wedge \phi_i^+(\vec{x}_{(i,1,+)})[s])) \vee F(x, s)$
$\wedge \neg((\bigvee_{j=1}^{m_1} [\exists x][\exists y](a = A_j^-(\vec{x}_{(j,0,-)}) \wedge \phi_j^-(\vec{x}_{(j,1,-)})[s]))),$
where each variable vector $\vec{x}_{(i,n,b)}$ (or $\vec{x}_{(j,n,b)}$ respectively) $(i = 1..m_0, j = 1..m_1, n \in \{0, 1\}, b \in \{+, -\})$ represents a list of object variables, which can be empty, x, y, $\langle x, y\rangle$ or $\langle y, x\rangle$. Moreover, $[\exists x]$ or $[\exists y]$ represents that the quantifier included in [ ] is optional; and each $\phi_i^+(\vec{x}_{(i,1,+)}), i = 1..m_0$ $(\phi_i^-(\vec{x}_{(j,1,-)}), j = 1..m_0$, respectively), is a C^2 formula with variables among x and y.

Similarly, an SSA for a dynamic primitive role $R(x, y, s)$ is provided as a formula of the form

$$R(x, y, do(a, s)) \equiv \psi_R(x, y, a, s). \tag{2}$$

Moreover, without loss of generality, the general structure of $\psi_R(x, y, a, s)$ is
$(\bigvee_{i=1}^{m_2} [\exists x][\exists y](a = A_i^+(\vec{x}_{(i,0,+)}) \wedge \phi_i^+(\vec{x}_{(i,1,+)})[s])) \vee R(x, y, s)$
$\wedge \neg((\bigvee_{j=1}^{m_3} [\exists x][\exists y](a = A_j^-(\vec{x}_{(j,0,-)}) \wedge \phi_j^-(\vec{x}_{(j,1,-)})[s]))),$
where each variable vector $\vec{x}_{(i,n,b)}$ (or $\vec{x}_{(j,n,b)}$ respectively) $(i = 1..m_2, j = 1..m_3, n \in \{0, 1\}, b \in \{+, -\})$ represents a vector of free variables, which can be either empty, x, y, $\langle x, y\rangle$ or $\langle y, x\rangle$. Moreover, $[\exists x]$ or $[\exists y]$ represents that the quantifier included in [ ] is optional; and each $\phi_i^+(\vec{x}_{(i,1,+)}), i = 1..m_2$ $(\phi_j^-(\vec{x}_{(j,1,-)}), j = 1..m_3$, respectively), is a C^2 formula with variables (both free and quantified) among x and y. Note that when m_0 (or m_1, m_2, m_3, respectively) is equal to 0, the corresponding disjunctive subformula is equivalent to $false$.

Acyclic TBox axioms $\mathcal{D}_T$: Similar to the TBox axioms in DL, we may define new concepts using TBox axioms. Any group of TBox axioms $\mathcal{D}_T$ may include two sub-classes: static TBox $\mathcal{D}_{T,st}$ and dynamic TBox $\mathcal{D}_{T,dyn}$. Every formula in static TBox is a *concept definition* formula of the form $G(x) \equiv \phi_G(x)$, where G is a unary predicate symbol and $\phi_G(x)$ is a C^2 formula in the domain with free variable x, and there is no fluent in it. Every formula in dynamic TBox is a *concept definition* formula of the form $G(x, s) \equiv \phi_G(x)[s]$, where $\phi_G(x)$ is a C^2 formula with free variable x, and there is at least one dynamic concept or dynamic role in it. All the concepts appeared in the left-hand side of TBox axioms are called *defined* concepts. We also require that the set of TBox axioms must be acyclic.

RBox axioms $\mathcal{D}_R$: Similar to the idea of RBox in DL, we may also specify a group of axioms, called RBox axioms below, to support a role taxonomy. Each role inclusion axiom is represented as $R_1(x, y)[s] \supset R_2(x, y)[s]$ where R_1 and R_2 are primitive roles (either static or dynamic). If these axioms are included in the BAT $\mathcal{D}$, then it is assumed that $\mathcal{D}$ is specified correctly in the sense that the meaning of any RBox axiom included in the theory is correctly compiled into SSAs. That is, one can prove by induction that $\mathcal{D} \models \forall s.R_1(x, y)[s] \supset R_2(x, y)[s]$. Although RBox axioms are not used by the regression operator, they are used for taxonomic reasoning in the initial theory.

Initial theory $\mathcal{D}_{S_0}$: It is a finite set of C^2 sentences (assuming that we suppress the only situation term S_0 in all fluents). It specifies the incomplete information about the initial problem state and also describes all the facts that are not changeable over time in the domain of an application. In particular, it includes static TBox axioms $\mathcal{D}_{T,st}$ as well as RBox axioms in the initial situation S_0 (if any).

The remaining two classes (foundational axioms for situations Σ and unique name axioms for actions $\mathcal{D}_{una}$) are the same as those in the BATs of the usual SC.

Extending Regression with Lazy Unfolding

After giving the definition of the BAT in $\mathcal{L}_{sc}^{DL}$, we turn our attention to the reasoning tasks. There are various kinds of reasoning problems we could think of. For example, if we are considering a planning problem, we are looking for a ground situation starting from the initial situation such that it is *executable* and a given goal (formalized as a logic formula w.r.t. this situation) can be entailed by $\mathcal{D}$. However, below we focus on two sub-problems of the planning problem (executability and projection), because they are the most essential for solving the planning (composition) problem.

Consider a BAT $\mathcal{D}$ of $\mathcal{L}_{sc}^{DL}$ specified as in the previous section for some dynamic system with OWA. Given a formula W of $\mathcal{L}_{sc}^{DL}$ in the domain $\mathcal{D}$, the definition of W being regressable (called $\mathcal{L}_{sc}^{DL}$ *regressable* below) is slightly different from the definition of W being regressable in $\mathcal{L}_{sc}$ (see Section) by adding the following additional conditions: (4) any variable (free or bounded) in W is either x or y; (5) every term of sort situation in W is ground. Moreover, in $\mathcal{L}_{sc}^{DL}$ we have to be more careful with the definition of the regression operator $\mathcal{R}$ for two main reasons. First, to deal with TBox we have to extend regression. For a $\mathcal{L}_{sc}^{DL}$ regressable formula W, we *extend* below the regression operator defined in (Reiter 2001) with the *lazy unfolding technique* (see (Baader *et al.* 2003)) and still denote such operator as $\mathcal{R}$. Second, $\mathcal{L}_{sc}^{DL}$ uses only two object variables and we have to make sure that after regressing a fluent atom we still get a $\mathcal{L}_{sc}^{DL}$ formula, i.e., that we never need to introduce new (free or bound) object variables. To deal with the two-variable restriction, we modify our regression operator $\mathcal{R}$ in comparison to the conventional operator defined in (Reiter 2001) as follows, where σ denotes the term of sort situation, and α denotes the term of sort action.

• If W is not atomic, i.e. W is of the form $W_1 \vee W_2$, $W_1 \wedge W_2$, $\neg W'$, $Qv.W'$ where Q represents a quantifier (including counting quantifiers) and v represents a variable symbol, then

$\mathcal{R}[W_1 \vee W_2] = \mathcal{R}[W_1] \vee \mathcal{R}[W_2], \qquad \mathcal{R}[\neg W'] = \neg\mathcal{R}[W'],$
$\mathcal{R}[W_1 \wedge W_2] = \mathcal{R}[W_1] \wedge \mathcal{R}[W_2], \mathcal{R}[Qv.W'] = Qv.\mathcal{R}[W'].$

• Otherwise, W is atom. There are several cases.

(a) If W is situation independent (including equality), or W is a concept or role uniform in S_0, then $\mathcal{R}[W] = W$.

(b) If W is a regressable $Poss$ atom, so it has the form $Poss(A(\vec{t}), \sigma)$, for terms of sort action and situation respectively in $\mathcal{L}_{sc}^{DL}$. Then there must be an action precondition axiom for A of the form $Poss(A(\vec{x}), s) \equiv \Pi_A(\vec{x}, s)$, where the argument $\vec{x}$ of sort object can either be empty (i.e., A

is an action constant), a single variable x or two-variable vector $\langle x, y\rangle$. Because of the syntactic restrictions of $\mathcal{L}_{sc}^{DL}$, each term in $\vec{t}$ can only be a variable x, y or a constant C. Then,

$$\mathcal{R}[W] = \begin{cases} \mathcal{R}[(\exists y)(x = y \wedge \Pi_A(x, y, \sigma))] & \text{if } \vec{t} = \langle x, x\rangle, \\ \mathcal{R}[(\exists x)(y = x \wedge \Pi_A(x, y, \sigma))] & \text{else if } \vec{t} = \langle y, y\rangle, \\ \mathcal{R}[\Pi_A(\vec{t}, \sigma)] & \text{else if } \vec{t} = x \text{ or} \\ & \vec{t} = \langle x, y\rangle \text{ or} \\ & \vec{t} = \langle x, C\rangle, \\ \mathcal{R}[\widetilde{\Pi_A}(\vec{t}, \sigma)] & \text{otherwise,} \end{cases}$$

where C represents a constant and $\widetilde{\phi}$ denotes a *dual formula* for formula ϕ obtained by replacing every variable symbol x (free or quantified) with variable symbol y and replacing every variable symbol y (free or quantified) with variable symbol x in ϕ, i.e., $\widetilde{\phi} = \phi[x/y, y/x]$.

(c) If W is a defined dynamic concept, so it has the form $G(\vec{t}, \sigma)$ for some object term $\vec{t}$ and situation term σ, and there must be a TBox axiom for G of the form $G(x, s) \equiv \phi_G(x, s)$. Because of the restrictions of the language $\mathcal{L}_{sc}^{DL}$, term $\vec{t}$ can only be a variable x, y or a constant. Then, we use lazy unfolding technique as follows:

$$\mathcal{R}[W] = \begin{cases} \mathcal{R}[\phi_G(t, \sigma)] & \text{if } t \text{ is not variable } y, \\ \mathcal{R}[\widetilde{\phi_G}(y, \sigma)] & \text{otherwise.} \end{cases}$$

(d) If W is a primitive concept (a primitive role, respectively), so it has the form $F(t_1, do(\alpha, \sigma))$ (the form $R(t_1, t_2, do(\alpha, \sigma))$, respectively) for some terms t_1 (and t_2) of sort object, term α of sort action and term σ of sort situation. There must be an SSA for F (for R, respectively) such as Eq. 1 (such as Eq. 2, respectively). Because of the restriction of the language $\mathcal{L}_{sc}^{DL}$, the term t_1 and t_2 can only be a variable x, y or a constant C and α can only an action function with no more than two arguments of sort object. Then, when W is a concept,

$$\mathcal{R}[W] = \begin{cases} \mathcal{R}[\psi_F(t_1, \alpha, \sigma)] & \text{if } t_1 \text{ is not variable } y, \\ \mathcal{R}[\widetilde{\psi_F}(y, \alpha, \sigma)] & \text{otherwise;} \end{cases}$$

and, when W is a role,

$$\mathcal{R}[W] = \begin{cases} \mathcal{R}[(\exists y)(x = y \wedge \psi_R(x, y, \alpha, \sigma))] & \text{if } t_1 = x, t_2 = x; \\ \mathcal{R}[\widetilde{(\exists x)}(y = x \wedge \psi_R(x, y, \alpha, \sigma))] & \text{if } t_1 = y, t_2 = y; \\ \mathcal{R}[\widetilde{\psi_R}(y, x, \alpha, \sigma)] & \text{if } t_1 = y, t_2 = x; \\ \mathcal{R}[\psi_R(t_1, t_2, \alpha, \sigma)] & \text{otherwise.} \end{cases}$$

Based on the above definition, we are able to prove the following theorems.

Theorem 2 *Suppose W is a $\mathcal{L}_{sc}^{DL}$ regressable formula, then the regression $\mathcal{R}[W]$ defined above terminates in a finite number of steps.*

Proof: Immediately follows from acyclicity of the TBox axioms, and from the assumption that $RBox$ axioms are compiled into the SSAs and consequently do not participate in regression. □

Moreover, it is easy to see that any $\mathcal{L}_{sc}^{DL}$ regressable formula has no more than two variables (x and y), and the following theorem holds.

Theorem 3 *Suppose W is a $\mathcal{L}_{sc}^{DL}$ regressable formula with the background basic action theory $\mathcal{D}$. Then, $\mathcal{R}[W]$ is a $\mathcal{L}_{sc}^{DL}$ formula uniform in S_0 with no more than two variables (x and y). Moreover, $\mathcal{D} \models W \equiv \mathcal{R}[W]$, and*

$$\mathcal{D} \models W \quad \textit{iff} \quad \mathcal{D}_{S_0} \cup \mathcal{D}_{una} \models \mathcal{R}[W].$$

Moreover, we can also obtain the following corollary about decidability of the projection problem for $\mathcal{L}_{sc}^{DL}$ regressable formula W (particularly, when W is of form $executable(S)$ for some ground situation S, it becomes the executability problem).

Corollary 1 *Suppose W is a $\mathcal{L}_{sc}^{DL}$ regressable formula with the background basic action theory $\mathcal{D}$. Then, the problem whether $\mathcal{D} \models W$ is decidable.*

Proof: Let $\mathcal{D}_0$ (W_0, respectively) be the theory (formula, respectively) obtained by suppressing situation term S_0 in $\mathcal{D}_{S_0}$ ($\mathcal{R}[W]$, respectively). Therefore, $\mathcal{D}_0$ and W_0 are in C^2. According to Theorem 3, $\mathcal{D} \models W$ iff $\mathcal{D}_{S_0} \cup \mathcal{D}_{una} \models \mathcal{R}[W]$, iff $\mathcal{D}_0 \cup \mathcal{D}_{una} \models W_0$, where W_0 is a C^2 formula. Therefore, the problem whether $\mathcal{D} \models W$ is equivalent to whether $\mathcal{D}_0 \wedge \mathcal{D}_{una} \wedge \neg W_0$ is unsatisfiable or not. According to the fact that the satisfiability problem in C^2 is decidable, the theorem is proved. □

Examples

In this section, we give some examples to illustrate the basic ideas described in the previous sections. First, we give the formal specification for the web services of an imaginary university described informally in the second section.

Example 1 Consider a university that provides on the Web student administration and management services, such as admitting students, paying tuition fees, enrolling or dropping courses and entering grades.

Notice that although the number of object arguments in the predicates can be at most two, sometimes, we are still able to handle those features that require more than two arguments. For example, the grade z of a student x in a course y may be represented as a predicate $grade(x, y, z)$ in the general FOL. Because the number of distinct grades is finite and they can be easily enumerated as "A", "B", "C" or "D", we can handle $grade(x, y, z)$ by replacing it with a finite number of extra predicates, say $gradeA(x, y)$, $gradeB(x, y)$, $gradeC(x, y)$ and $gradeD(x, y)$ such that they all have two variables only. However, the restriction on the number of variables limits the expressive power of the language if more than two arguments vary over infinite domains. Despite that, we conjecture that lots of the web services still can be represented with two variables either by introducing extra predicates (just like we did for the predicate $grade$) or by grounding some of the arguments if their domains are finite and relatively small. Intuitively, it seems that most of the dynamic systems can be specified by using properties and actions with small arities, hence the techniques for arity reductions mentioned above require no more than polynomial increase in the number of axioms.

The high-level features of our example are specified as the following concepts and roles:

- Static primitive concepts: $person(x)$ (x is a person); $course(x)$ (x is a course provided by the university).
- Dynamic primitive concepts: $incoming(x, s)$ (x is an

incoming student in the situation s true when x was admitted); $student(x,s)$ (x is an eligible student in the situation s when an incoming student x pays the tuition fee).

- Dynamic defined concepts: $eligFull(x,s)$ (x is eligible to be a full-time student by paying more than 5000 dollars tuition fee); $eligPart(x,s)$ (x is eligible to be a part-time student by paying no more than 5000 dollars tuition); $qualFull(x,s)$ (x is a qualified full-time student if he or she pays full time tuition fee and takes at least 4 courses); $qualPart(x,s)$ (x is a part-time student if he or she pays part-time tuition and takes 2 or 3 courses).
- Static role: $preReq(x,y)$ (course x is a prerequisite of course y).
- Dynamic roles: $tuitPaid(x,y,s)$ (x pays tuition fee y in the situation s); $enrolled(x,y,s)$ (x is enrolled in course y in the situation s); $completed(x,y,s)$ (x completes course y in the situation s); $hadGrade(x,y,s)$ (x had a grade for course y in the situation s); $gradeA(x,y,s)$; $gradeB(x,y,s)$; $gradeC(x,y,s)$; $gradeD(x,y,s)$.

Web services are specified as actions: $reset$ (at the beginning of each academic year, the system is being reset so that students need to pay tuition fee again to become eligible); $admit(x)$ (the university admits student x); $payTuit(x,y)$ (x pays tuition fee with the amount of y); $enroll(x,y)$ (x enrolls in course y); $drop(x,y)$ (x drops course y); $enterA(x,y)$ (enter grade "A" for student x in course y); $enterB(x,y)$; $enterC(x,y)$; $enterD(x,y)$.

The basic action theory is as follows (most of the axioms are self-explanatory).

Precondition Axioms: $Poss(reset,s) \equiv true$,

$Poss(admit(x),s) \equiv person(x) \wedge \neg incoming(x,s)$,
$Poss(payTuit(x,y),s) \equiv incoming(x,s) \wedge \neg student(x,s)$,
$Poss(drop(x,y),s) \equiv enrolled(x,y,s) \wedge \neg completed(x,y,s)$,
$Poss(enterA(x,y),s) \equiv enrolled(x,y,s)$,

and similar to $enterA(x,y)$, the precondition for $enterB(x,y)$ ($enterC(x,y)$ and $enterB(x,y)$ respectively) at any situation s is also $enrolled(x,y,s)$. Moreover, in the traditional SC, the precondition for action $enroll(x,y)$ would be equivalent to

$$student(x) \wedge (\forall z)(preReq(z,y) \wedge completed(x,z,s) \wedge \neg gradeD(x,z,s)) \wedge course(y).$$

However, in the modified SC, we only allow at most two variables (including free or quantified) other than the situation variable s and action variable a. Fortunately, the number of the courses offered in a university is limited (finite and relatively small) and relatively stable over years (if we manage the students in a college-wise range or department-wise range, the number of courses may be even smaller). Therefore, we can specify the precondition for the action $enroll(x,y)$ for each instance of y. That is, assume that the set of courses is $\{CS_1,\cdots,CS_n\}$, the precondition axiom for each CS_i $(i=1..n)$ is

$$Poss(enroll(x,CS_i),s) \equiv student(x) \wedge (\forall y)(preReq(y,CS_i) \wedge completed(x,y,s) \wedge \neg gradeD(x,y,s)).$$

On the other hand, when we do this transformation, we can omit the statements $course(x)$ for each course available at the university in the initial theory.

Successor State Axioms:

$incoming(x,do(a,s)) \equiv a = admit(x) \vee incoming(x,s)$,
$student(x,do(a,s)) \equiv (\exists y)(a = payTuit(x,y)) \vee student(x) \wedge a \neq reset$,
$tuitPaid(x,y,do(a,s)) \equiv a = payTuit(x,y) \vee tuitPaid(x,y,s) \wedge a \neq reset$,
$enrolled(x,y,do(a,s)) \equiv a = enroll(x,y) \vee enrolled(x,y,s) \wedge \neg(a = drop(x,y) \vee a = enterA(x,y) \vee a = enterB(x,y) \vee a = enterC(x,y) \vee a = enterD(x,y))$,
$completed(x,y,do(a,s)) \equiv a = enterA(x,y) \vee a = enterB(x,y) \vee a = enterC(x,y) \vee a = enterD(x,y) \vee completed(x,y,s) \wedge a \neq enroll(x,y)$,
$gradeA(x,y,do(a,s)) \equiv a = enterA(x,y) \vee gradeA(x,y,s) \wedge \neg(a = enterB(x,y) \vee a = enterC(x,y) \vee a = enterD(x,y))$,

and the SSAs for fluent $gradeB(x,y,s)$, $gradeC(x,y,s)$ and $gradeD(x,y,s)$ are very similar to the one for fluent $gradeA(x,y,s)$, which ensures that for each student and each course no more than one grade is assigned.

Acyclic TBox Axioms:

$eligFull(x,s) \equiv (\exists y)(tuitPaid(x,y,s) \wedge y > 5000)$,
$eligPart(x,s) \equiv (\exists y)(tuitPaid(x,y,s) \wedge y \leq 5000)$,
$qualFull(x,s) \equiv eligFull(x,s) \wedge (\exists^{\geq 4} y)enrolled(x,y,s)$,
$qualPart(x,s) \equiv eligPart(x,s) \wedge (\exists^{\geq 2} y)enrolled(x,y,s) \wedge (\exists^{\leq 3} enrolled(x,y,s))$.

An initial theory $\mathcal{D}_{S_0}$ may be the conjunctions of the following sentences: $(\forall x)\neg student(x,S_0)$;
$person(PSN_1), person(PSN_2), \cdots, person(PSN_m)$,
$(\forall x)incoming(x,S_0) \supset x = PSN_2 \vee x = PSN_3$,
$preReq(CS_1,CS_4) \vee preReq(CS_3,CS_4)$,
$(\forall x)x \neq CS_4 \supset \neg(\exists y).prePeq(y,x)$.

Suppose we denote above basic action theory as $\mathcal{D}$. Given goal G, for example $\exists x.eligPart(x)$, and a composite web service starting from the initial situation, for example $do([admit(PSN_1), payTuit(PSN_1, 3000)], S_0)$ (we denote the corresponding resulting situation as S_r), we can check if the goal is satisfied after the execution of this composite web service by solving the projection problem whether $\mathcal{D} \models G[S_r]$. In our example, this corresponds to solving whether $\mathcal{D} \models \exists x.eligPart(x,S_r)$. We may also check if a given (ground) composite web service $A_1; A_2; \cdots; A_n$ is possible to execute starting from the initial state by solving the executability problem whether $\mathcal{D} \models executable(do([A_1,A_2,\cdots,A_n],S_0))$. For example, we can check if composite web service $admit(PSN_1); payTuit(PSN_1,3000)$ is possible to be executed from the starting state by solving whether $\mathcal{D} \models executable(S_r)$.

Example 2 Consider a web service dynamic system in which clients are able to buy CDs and books online with credit cards. The system high-level features of this example are specified as concepts and roles.

- Static primitive concept(s): $person(x)$ (x is a person); $cd(x)$ (x is a CD); $book(x)$ (x is a book); $creditCard(x)$ (x is a credit card).
- Static defined concept(s): $client(x)$ (x is a client).
- Dynamic primitive concept(s): $instore(x,s)$ (x is in store in situation s).

• Dynamic defined concept(s): $valCust(x, s)$ (x is valuable customer in s).
• Static role(s): $has(x, y)$ (x has y).
• Dynamic role(s): $boughtCD(x, y, s)$ (x bought CD y in situation s); $boughtBook(x, y, s)$ (x bought book y in situation s); $bought(x, y, s)$ (x bought y in situation s).

Web services are specified as actions: $buyCD(x, y)$ (x buys CD y); $buyBook(x, y)$ (x buys book y); $returnCD(x, y)$ (x returns CD y); $returnBook(x, y)$ (x returns book y); $order(x)$ (the web service agent orders x from the publisher).

The basic action theory is as follows (most of the axioms are self-explanatory).

Precondition Axioms:
$Poss(buyCD(x, y), s) \equiv client(x) \wedge cd(y) \wedge instore(y, s)$,
$Poss(buyBook(x, y), s) \equiv client(x) \wedge book(y) \wedge instore(y, s)$,
$Poss(returnCD(x, y), s) \equiv boughtCD(x, y, s)$,
$Poss(returnBook(x, y), s) \equiv boughtBook(x, y, s)$,
$Poss(order(x), s) \equiv book(x) \vee cd(x)$.

Successor State Axioms:
$instore(x, do(a, s)) \equiv (\exists y)(a = returnCD(y, x)) \vee$
$(\exists y)(a = returnBook(y, x)) \vee a = order(x) \vee instore(x, s)$
$\wedge \neg((\exists y)(a = buyCD(y, x)) \vee (\exists y)(a = buyBook(y, x)))$,
$boughtCD(x, y, s) \equiv a = buyCD(x, y) \vee$
$boughtCD(x, y, s) \wedge a \neq returnCD(x, y)$,
$boughtBook(x, y, s) \equiv a = buyBook(x, y) \vee$
$boughtBook(x, y, s) \wedge a \neq returnBook(x, y)$,
$bought(x, y, s) \equiv a = buyCD(x, y) \vee a = buyBook(x, y) \vee$
$bought(x, y, s) \wedge \neg(cd(y) \wedge a = returnCD(x, y)$
$\vee book(y) \wedge a = returnBook(x, y))$.

Acyclic TBox Axioms: (both dynamic and static)
$valCust(x, s) \equiv person(x) \wedge \exists^{\geq 3} y.(bought(x, y, s))$.
$client(x) \equiv person(x) \wedge (\exists y)(has(x, y) \wedge CreditCard(y))$.

RBox Axioms: $boughtCD(x, y, s) \supset bought(x, y, s)$,
$boughtBook(x, y, s) \supset bought(x, y, s)$.

We also provide below some examples of $\mathcal{L}_{sc}^{DL}$ regressable formulas and the regression of some of these formulas.

$executable(S_1)$, $(\exists x) valCust(x, S_1)$, where
$S_1 = do([buyCD(Tom, BackStreetBoys),$
$buyBook(Tom, HarryPotter), buyBook(Tom, TheFirm)], S_0)$

Here is an example of the regression.
$\mathcal{R}[(\exists x) valCust(x, S_1)]$
$= (\exists x)(person(x) \wedge \exists^{\geq 3} y.\mathcal{R}[bought(x, y, S_1)]) = \cdots$
$= (\exists x)(person(x) \wedge \exists^{\geq 3} y.(x = Tom \wedge y = TheFirm \vee$
$x = Tom \wedge y = HarryPotter \vee$
$x = Tom \wedge y = BackStreetBoys \vee bought(x, y, S_0)))$,
which is true given that $\mathcal{D}_{S_0}$ is the conjunction of the following sentences.
$person(Tom), cd(SpiceGirls), person(Sam)$,
$creditCard(Visa), creditCard(MasterCard)$,
$book(TheFirm), book(Java), book(HarryPotter)$,
$has(Tom, Visa) \vee has(Tom, MasterCard)$,
$has(Sam, Visa) \vee has(Same, MasterCard)$,
$\forall x(instore(x, S_0) \vee x = Java), cd(BackStreetBoys)$.

Discussion and Future Work

The major consequence of the results proved above for the problem of service composition is the following. If both atomic services and properties of the world that can be affected by these services have no more than two parameters, then we are guaranteed that even in the state of incomplete information about the world, one can always determine whether a sequentially composed service is executable and whether this composite service will achieve a desired effect. The previously proposed approaches made different assumptions: (McIlraith & Son 2002) assumes that the complete information is available about the world when effects of a composite service are computed, and (Berardi *et al.* 2003) considers the propositional fragment of the SC.

As we mentioned in Introduction, (McIlraith & Son 2002; Narayanan & McIlraith 2003) propose to use Golog for composition of Semantic Web services. Because our primitive actions correspond to elementary services, it is desirable to define Golog in our modified SC too. It is surprisingly straightforward to define almost all Golog operators starting from our C^2 based SC. The only restriction in comparison with the original Golog (Reiter 2001) is that we cannot define the operator $(\pi x)\delta(x)$, non-deterministic choice of an action argument, because $\mathcal{L}_{sc}^{DL}$ regressable formulas cannot have occurrences of non-ground action terms in situation terms. In the original Golog this is allowed, because the regression operator is defined for a larger class of regressable formulas. However, everything else from the original Golog specifications remain in force, no modifications are required. In addition to providing a well-defined semantics for Web services, our approach also guarantees that evaluation of tests in Golog programs is decidable (w.r.t. arbitrary theory $\mathcal{D}_{S_0}$) that is missing in other approaches (unless one can make the closed world assumption or impose another restriction to regain decidability).

The most important direction for future research is an efficient implementation of a decision procedure for solving the executability and projection problems. This procedure should handle the modified $\mathcal{L}_{sc}^{DL}$ regression and do efficient reasoning in $\mathcal{D}_{S_0}$. It should be straightforward to modify existing implementations of the regression operator for our purposes, but it is less obvious which reasoner will work efficiently on practical problems. There are several different directions that we are going to explore. First, according to (Borgida 1996) and Theorem 2, there exists an efficient algorithm for translating C^2 formulas to $\mathcal{ALCQI}(\sqcup, \sqcap, \neg, |, id)$ formulas. Consequently, we can use any resolution-based description logic reasoners that can handle $\mathcal{ALCQI}(\sqcup, \sqcap, \neg, |, id)$ (e.g., MSPASS). Alternatively, we can try to use appropriately adapted tableaux-based description logic reasoners, such as FaCT++, for (un)satisfiability checking in $\mathcal{ALCQI}(\sqcup, \sqcap, \neg, |, id)$. Second, we can try to avoid any translation from C^2 to $\mathcal{ALCQI}(\sqcup, \sqcap, \neg, |, id)$ and adapt resolution based automated theorem provers for our purposes.

The recent paper by (Baader *et al.* 2005) proposes integration of description logics $\mathcal{ALCQIO}$ (and its sub-languages) with an action formalism for reasoning about Web services. This paper starts with a description logic and then defines services (actions) meta-theoretically: an atomic service is defined as the triple of sets of description logic formulas. To solve the executability and projection problems this paper introduces an approach similar to regression, and reduces this problem to description logic reasoning. The main aim is to show how executability of sequences of actions and solution of the executability and projection problems can be computed, and how complexity of these problems depend on the chosen description logic. In the full version of (Baader *et al.* 2005), there is a detailed embedding of the proposed framework into the syntactic fragment of the Reiter's SC. It is shown that solutions of their executability and projection problems correspond to solutions of these problems w.r.t. the Reiter's basic action theories in this fragment for appropriately translated formulas. To achieve this correspondence, one needs to eliminate TBox by unfolding (this operation can result potentially in exponential blow-up of the theory). Despite that our paper and (Baader *et al.* 2005) have common goals, our developments start differently and proceed in the different directions. We start from the syntactically restricted first-order language (that is significantly more expressive than $\mathcal{ALCQIO}$), use it to construct the modified SC (where actions are terms), define basic action theories in this language and show that by augmenting (appropriately modified) regression with lazy unfolding one can reduce the executability and projection problems to the satisfiability problem in C^2 that is decidable. Furthermore, C^2 formulas can be translated to $\mathcal{ALCQI}(\sqcup, \sqcap, \neg, |, id)$, if desired. Because our regression operator unfolds fluents "on demand" and uses only relevant part of the (potentially huge) TBox , we avoid potential computational problems that may occur if the TBox were eliminated in advance. The advantage of (Baader *et al.* 2005) is that all reasoning is reduced to reasoning in description logics (and, consequently, can be efficiently implemented especially for less expressive fragments of $\mathcal{ALCQIO}$). Our advantages are two-fold: the convenience of representing actions as terms, and the expressive power of $\mathcal{L}_{sc}^{DL}$. Because C^2 and $\mathcal{ALCQI}(\sqcup, \sqcap, \neg, |, id)$ are equally expressive, there are some (situation suppressed) formulas in our SC that cannot be expressed in $\mathcal{ALCQIO}$ (that does not allow complex roles).

There are several other proposals to capture the dynamics of the world in the framework of description logics and/or its slight extensions. Instead of dealing with actions and the changes caused by actions, some of the approaches turned to extensions of description logic with temporal logics to capture the changes of the world over time (Artale & Franconi 2001; Baader *et al.* 2003), and some others combined planning techniques with description logics to reason about tasks, plans and goals and exploit descriptions of actions, plans, and goals during plan generation, plan recognition, or plan evaluation (Gil 2005). Both (Artale & Franconi 2001) and (Gil 2005) review several other related papers. In (Berardi *et al.* 2003), Berardi et al. specify all the actions of e-services as constants, all the fluents have only situation argument, and translate the basic action theory under such assumption into description logic framework. It has a limited expressive power without using arguments of objects for actions and/or fluents: this may cause a blow-up of the knowledge base.

References

Artale, A., and Franconi, E. 2001. A survey of temporal extensions of description logics. *Annals of Mathematics and Artificial Intelligence* 30(1-4).

Baader, F.; Calvanese, D.; McGuinness, D.; Nardi, D.; and Patel-Schneider, P. F., eds. 2003. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge Un. Press.

Baader, F.; Lutz, C.; Miliĉić, M.; Sattler, U.; and Wolter, F. 2005. Integrating description logics and action formalisms: First results. In *Proceedings of the Twentieth National Conference on Artificial Intelligence (AAAI-05)*, 572–577. extended version is available as LTCS-Report-05-02 from http://lat.inf.tu-dresden.de/research/reports.html.

Baader, F.; Horrocks, I.; and Sattler, U. 2005. Description logics as ontology languages for the semantic web. In Hutter, D., and Stephan, W., eds., *Mechanizing Mathematical Reasoning, Essays in Honor of Jörg H. Siekmann on the Occasion of His 60th Birthday, Lecture Notes in Computer Science, vol. 2605*, 228–248. Springer.

Berardi, D.; Calvanese, D.; Giacomo, G. D.; Lenzerini, M.; and Mecella, M. 2003. e-service composition by description logics based reasoning. In Calvanese, D.; de Giacomo, G.; and Franconi, E., eds., *Proceedings of the 2003 International Workshop in Description Logics (DL-2003)*.

Borgida, A. 1996. On the relative expressiveness of description logics and predicate logics. *Artif. Intell.* 82(1-2):353–367.

Giacomo, G. D.; Iocchi, L.; Nardi, D.; and Rosati, R. 1999. A theory and implementation of cognitive mobile robots. *Journal of Logic and Computation* 9(5):759–785.

Giacomo, G. D. 1995. *Decidability of Class-Based Knowledge Representation Formalisms*. Roma, Italy: Dipartimento di Informatica e Sistemistica Universita di Roma "La Sapienza".

Gil, Y. 2005. Description logics and planning. *AI Magazine* 26(2):73–84.

Grädel, E.; Otto, M.; and Rosen, E. 1997. Two-variable logic with counting is decidable. In *Proceedings of the 12th Annual IEEE Symposium on Logic in Computer Science (LICS'97)*, 306–317.

Grüninger, M., and Menzel, C. 2003. The process specification language (PSL): Theory and applications. *AI Magazine* 24(3):63–74.

Grüninger, M. 2004. Ontology of the process specification language. In Staab, S., and Studer, R., eds., *Handbook on Ontologies*, 575–592. Springer.

Horrocks, I.; Patel-Schneider, P.; and van Harmelen, F. 2003. From SHIQ and RDF to OWL: The making of a web ontology language. *Journal of Web Semantics* 1(1):7–26.

Hull, R., and Su, J. 2005. Tools for composite web services: a short overview. *SIGMOD Record* 34(2):86–95.

McIlraith, S., and Son, T. 2002. Adapting Golog for composition of semantic web services. In Fensel, D.; Giunchiglia, F.; McGuinness, D.; and Williams, M.-A., eds., *Proceedings of the Eighth International Conference on Knowledge Representation and Reasoning (KR2002)*, 482–493. Toulouse, France.

Narayanan, S., and McIlraith, S. 2003. Analysis and simulation of web services. *Computer Networks* 42:675–693.

Pacholski, L.; Szwast, W.; and Tendera, L. 1997. Complexity of two-variable logic with counting. In *Proceedings of the 12th Annual IEEE Symposium on Logic in Computer Science (LICS-97)*, 318–327. Warsaw, Poland: A journal version: SIAM Journal on Computing, v 29(4), 1999, p. 1083–1117.

Pistore, M.; Marconi, A.; Bertoli, P.; and Traverso, P. 2005. Automated composition of web services by planning at the knowledge level. In *Proc. of the Nineteenth International Joint Conference on Artificial Intelligence (IJCAI05)*, 1252–1259. Edinburgh, Scotland, UK: http://ijcai.org/papers/1428.pdf.

Reiter, R. 2001. *Knowledge in Action: Logical Foundations for Describing and Implementing Dynamical Systems*. The MIT Press.

Sirin, E.; Parsia, B.; Wu, D.; Hendler, J.; and Nau, D. 2004. HTN planning for web service composition using SHOP2. *Journal of Web Semantics* 1(4):377–396.