

# On Special Description Logics for Processes and Plans

Alexander Borgida<sup>†</sup>, David Toman<sup>‡</sup> and Grant Weddell<sup>‡</sup>

<sup>†</sup>Department of Computer Science, Rutgers University, New Brunswick, USA  
borgida@cs.rutgers.edu

<sup>‡</sup>Cheriton School of Computer Science, University of Waterloo, Canada  
{david,gweddell}@uwaterloo.ca

**Abstract.** Representing and reasoning with processes and plans is a core problems in many areas, including AI Planning and Plan Recognition, Business Process Modeling, Web Services, and Human Behavior Recognition. Ontologies based on Description Logics have been repeatedly argued to help in all these areas, but they have almost never used DLs in a way that supports full reasoning. We start to remedy this problem by considering Process and Plan DLs, where concepts have as instances sequences of action instances, and are built with a variety of special constructors. Inspired by the CLASP system, we consider families of DLs based on combinations of regular-like expression constructors: sequence, disjunction, looping, conjunction, complement, and concurrency, providing a rich variety of Plan DLs.

We present and extend results from the wide formal languages literature with bearing on the complexity of standard DL reasoning tasks (concept inconsistency, subsumption, and recognition), as well as on the important issue of representation succinctness. This work hopefully opens up rich new areas of research, where traditional questions can be investigated in a new setting.

## 1 Introduction and Motivation

In modern applications one needs to represent not just static but also dynamic aspects of a domain. For example, one may want to say that making tea consists of putting hot water in a cup, repeatedly dunking a tea bag in it, and then optionally adding sugar and/or milk to it in either order. Each of these may in turn be described in further detail. When entering this information in ontologies, one is normally drawn to ontology languages, such as OWL, which are based on Description Logics (DLs). . In particular, in this paper we explore variations on DL-like formalisms for describing *concepts having as instances sequences of (property-less) action instances*<sup>1</sup>.

---

<sup>1</sup> We do not consider here the representation of atomic actions in DLs, since these have been more widely studied. There is nothing to prevent the actions from having complex structure, such as parameters, pre-/post-conditions,... but the plan concepts will not be able to access this structure.

The following is a list of some of the *areas motivating this research*, with pointers to relevant literature, including use of ontologies.

**Planning and Plan Recognition in Artificial Intelligence:** In her review paper [17], Gil elaborates on the following applications of DL reasoning about plans, especially plan taxonomies: i) organization of plan classes; ii) retrieval of plan types and instances with description-based queries; iii) validation of plans based on descriptions of valid classes of plans; and iv) recognition of plan executions/instances. Weida [63] also lists many advantages for using DL formalisms for the planning domain.

**(Business) Process Modeling (BPM) and Workflow Management:** BPM represents the processes of an enterprise, aiming to analyze, improve, and automate them [37]. The latter is often accomplished through the use of workflow management systems [46]. BPMN [45] is one widely known graphical notation, which specifies, among others, control flow<sup>2</sup> by connecting subprocesses with arrows (representing simple sequence) and control-flow “gateways” (XOR, AND, OR), which come in “split” and “join” variants. UML activity diagrams [55] are another notation for similar purposes, widely-used in object-oriented modeling and programming. There are numerous proposals of OWL DL ontologies for capturing BPMN flow objects [43, 54, 29].

**Web Service Description:** Web service languages are used to describe the functionality offered by a web service. Among important tasks are finding desired services [32], and composing services to achieve some goal [36]. Though these languages focus on describing the input/output behavior of operations, some allow for the description of complex processes. OWL-S [47], and its predecessor DAML-S, are the most ambitious, and include OWL DL ontologies for control constructs for combining complex processes.

**Human Behavior Recognition:** Many areas, including life logging, ambient intelligence, and recognizing activities of daily living, aim to detect what activities humans are engaged in based on reports from sensor networks [62, 64] or other digital evidence [30, 20]. The natural reasoning task here is plan concept instance recognition, once composite activities have been modeled as concepts. There have been many proposals to use ontologies to support these tasks [53], including an extensive examination of the utility of OWL 2 for helping describe human activities [52].

In using DLs for the above kinds of tasks, many applications describe atomic actions using ordinary DLs, and then use a separate kind of formalism, such as planners or Hidden Markov Models to “reason” with them.

Even when the DL ontologies contain composite processes [43, 54, 29, 47, 52, 53], they only model their syntactic structure, and are unable to *reason* about composite actions by deducing, for example, that (i) placing a call, followed by either talking or hanging up, is logically equivalent to (ii) placing a call followed by talking, or placing a call followed by hanging up. As another example, in [22] (where action recognition is performed using a probabilistic DL), a complex

---

<sup>2</sup> There are many other aspects to BPMN, including exceptions, data flow, and messages, which are not considered here.

process is described by giving its immediate components as the values of one role, such as *hasSimpleActivity*; but sequencing is specified using an ad-hoc technique, by conjoining to components  $\mathcal{EL}$  concepts indicating a number, such as  $\exists hasOrder.\{1\}$ ,  $\exists hasOrder.\{2\}$ , etc.

Our goal is to develop DLs for building ontologies and taxonomies which “understand” the meaning of composite processes. In addition to supporting standard DL reasoning tasks, we take the following to be a distinguishing feature of standard Description Logics: the language is term-like, with concept and role constructors that build composite concepts from simpler ones, bottoming out at identifiers. For example, the familiar  $\exists parent.(Doctor \sqcap Tall)$  is infix notation for the term **some**(*parent*, **and**(*Doctor*, *Tall*)).

This syntactic issue eliminates at first glance diagrammatic notations such as Petri nets, various business process and workflow notations, and even finite state machines (but see Sections 3.3 and 4.1). It also rules out the vast majority of process description languages, such as the  $\pi$ -calculus [42], and even terminologic logics that use variables and quantifiers (e.g., [57]). Even languages that use DLs to describe bits and pieces of actions (e.g., using ABoxes to describe updates [21]) do not qualify.

The remaining DL formalisms that qualify under our criteria include: CLASP [12, 10, 4], which is based on regular expressions; DLs based on variants of Propositional Dynamic Logic [56, 7, 8]; and DLs based on temporal logics that use modal operators [38, 9]. This paper examines the use of extended regular expressions as the basis of Plan DLs, and briefly mention the others in Section 5.

**Contributions:** We mine the extensive literature on formal languages to obtain, and sometimes extend, results concerning the (i) expressive power, (ii) complexity of standard DL reasoning tasks (concept consistency, subsumption and membership), and especially (iii) descriptive complexity of Plan DLs based on extended regular expressions. This includes plan concept constructors for sequencing, alternation/disjunction, looping/Kleene star, intersection/conjunction and complement. Motivated by the desire to add concurrency to Plan DLs, we study the addition of interleaving as a concept constructor, and its connection to structured workflows [31].

One can also view plans/processes as forming a “concrete domain”, so that plan concepts can be used for role restrictions in ordinary DLs. This motivates the study of the *restricted* use of conjunction and complementation at the top-level, since Baader and Hanshke [2] show that this is sufficient for the domain to be “admissible”. This restriction is important since the complexity of reasoning with complement is non-elementary in the general case. We also consider briefly the effect on complexity of two concept constructors in the original CLASP: counted iteration and named subplans (acyclic definitional TBoxes).

In addition, we show, using language equations, that it is actually possible to view finite automata (FAs) as simple cyclic Plan DL TBoxes, using only sequencing and alternation as concept constructors. This means that such visually compelling, and sometimes more succinct notations, can also be captured by the family of Plan DL formalisms studied here.

## 2 The Framework of Regular Expression-based Plan DLs

The original CLASP system [12, 10] was developed to help reason about large telephonics software projects [11], and as such it had to handle information such as the fact that a phone call consists of picking up the phone, getting a dial tone, dialing, getting a ring tone, etc. For this purpose, CLASP provided a language for describing plan concepts whose instances are called *scenarios*, and algorithms for computing subsumption between these, as well as recognizing scenarios as concept instances. The representation was built on top of atomic actions, resembling STRIPS-like operators, such as *Ring*, with add- and delete-lists, and pre- and post-conditions, where specific states, such as *phone1-is-ringing*, were instances of atomic concepts, such as *PhoneRinging*. All of the information about states and actions was represented in the CLASSIC DL.

We are not interested here in the process of planning itself, and hence we will treat atomic action individuals as propertyless individuals. Following the “rational reconstruction” of CLASP in [4], we will use term-constructor **act** to identify atomic action concepts, and constructors **seq**, **or** and **loop** to represent sequencing, alternation and looping in composite plans. Using these, one might then describe the concept *MakingAPhoneCall* as

$$\mathbf{seq}(\mathbf{act}(\mathit{Dial}), \mathbf{loop}(\mathbf{act}(\mathit{Ring})), \mathbf{or}(\mathbf{act}(\mathit{Talk}), \mathbf{act}(\mathit{HangUp})))$$

A CLASP instance scenario of this plan might be

*[1234dials1212at6am, 1212ringsAt6am, 1212ringsAt6:01am, 1234hangsUpAt6:02am]* .

CLASP’s implementation of plan reasoning is based on the observation that **{seq, or, loop}** correspond to regular expression (RE) constructors  $\{\circ, \cup, *\}$ , when the set *Actions* of action concept names is viewed as the alphabet  $\Sigma$  used in REs. For example, in order to check the subsumption  $P1 \sqsubseteq P2$ , Devanbu and Litman [10] construct a product automaton from the deterministic automata for *P1* and the complement of *P2* (with a potential single exponential explosion when eliminating non-determinism), and then check it for emptiness.

### 2.1 Syntax and Semantics of RE-based Plan DLs

As in the above example, a scenario is a sequence/string of instances of action concepts from some finite set (the action terminology *Actions*). Since in this paper we are not interested in information about individual actions other than their type, we will not distinguish different instances of the same action concept, and assume that each action class *a* has a single instance, “**a**”. By abuse of notation, we will usually drop the quotes on strings, and use *a* to represent both the action class and its instance. We assume for now that action concepts are either disjoint or related by subsumption, that the set of action concepts in *Actions* covers the set of all possible individual actions, and even ignore action subsumption.

To describe classes of scenarios we therefore start with a finite set, *Actions*, of atomic concept names for actions, a disjoint set of identifiers *N* for plan concepts, and plan constructors **act**, for single action plans, and constructors **seq**,

**or** and **loop**. We add to this some useful constants, and plan constructors for intersection and complementation. The semantics of plan concepts is provided by an interpretation  $\mathcal{I} = (Actions_{\mathcal{I}}, \cdot^{\mathcal{I}})$ , where  $Actions_{\mathcal{I}}$  is, in our simplified case, just a finite set isomorphic to  $Actions$ , and  $\cdot^{\mathcal{I}}$  maps plan names to subsets of the set of strings/sequences over  $Actions_{\mathcal{I}}$ , written here, for clarity, as  $Sequences(Actions_{\mathcal{I}})$ .  $\mathcal{I}$  is then extended in the natural way to some constants and the constructors in the manner shown in Figure 1.

name of constructor	Syntax	Term notation	Semantics
atomic action $a$ $a \in Actions$	$a$	<b>act</b> ( $a$ )	$\{ "a" \}$
plan concept name $A$ $A \in N$	$A$	<b>subplan</b> ( $A$ )	$A^{\mathcal{I}} \subseteq Sequences(Actions_{\mathcal{I}})$
no-action plan	$Null$	<b>Null</b>	$\{ \lambda \}$
bottom concept	$\perp$	<b>Bottom</b>	$\emptyset$
top-plan concept	$\top_P$	<b>Top</b> <sub>Plan</sub>	$Sequences(Actions_{\mathcal{I}})$
any one action	$Actions$	<b>Actions</b>	$Actions_{\mathcal{I}}$
sequence	$P_1 \circ P_2$	<b>seq</b> ( $P_1, P_2$ )	$\{ uw \mid u \in P_1^{\mathcal{I}}, w \in P_2^{\mathcal{I}} \}$
alternation	$P_1 \sqcup P_2$	<b>or</b> ( $P_1, P_2$ )	$P_1^{\mathcal{I}} \cup P_2^{\mathcal{I}}$
repetition (base)	$P^0$	<b>repeat</b> ( $0, P$ )	$Null^{\mathcal{I}}$
repetition (ind'n)	$P^{k+1}$	<b>repeat</b> ( $k+1, P$ )	$P^{\mathcal{I}} \circ (P^k)^{\mathcal{I}}$
loop	$P^*$	<b>loop</b> ( $P$ )	$\bigcup_{i \geq 0} (P^i)^{\mathcal{I}}$
conjunction	$P_1 \sqcap P_2$	<b>and</b> ( $P_1, P_2$ )	$P_1^{\mathcal{I}} \cap P_2^{\mathcal{I}}$
complement	$\neg P$	<b>not</b> ( $P$ )	$Sequences(Actions_{\mathcal{I}}) - P^{\mathcal{I}}$

**Fig. 1.** Syntax and Semantics of a RE-based Plan DL

In the absence of Plan DL TBoxes, which introduce defined names for Plan DL concepts, we will drop the **act** constructor, since all names refer to action concepts.

## 2.2 Issues of Interest

Based on the application of Plan DLs, and the history of DL research, we will consider the following problems.

**Expressive power:** We can adapt Baader's notion of expressive power [1], intended for logics with Tarskian semantics, with the only difference being that interpretations now assign sets of strings to plan concepts. Interestingly, the results coincide with the notion of grammatical formalisms being more or equally expressive in terms of the formal languages they can describe.

**Computational complexity:** We will consider the complexity of the standard questions usually associated with DLs: *concept inconsistency* (*corresponding to language emptiness*), *subsumption* (*language containment*), and *membership*. Sometimes we will report the complexity of a problem in terms of its complement (e.g., notEmpty rather than inconsistent), because these are more easily checked non-deterministically, and one therefore avoids having to use "co-C" complexity classes; also, these are reported in this way in the literature.

Although the formal languages literature does address the question of (in)consistency, and membership, it usually does not address directly the question of subsumption. Instead, the problem considered is “weaker”: the inequality of two languages. Although complexity results for this provide lower bounds for the subsumption problem, one cannot automatically assume that subsumption is in the same complexity class. Deterministic Context Free Languages provide an extreme example: equality of these is decidable [58], but containment is undecidable [14].

**Succinctness/Descriptive Complexity:** Especially in cases of Plan DLs with equal expressive power, it is interesting to see when one allows for more succinct descriptions than another. For example, Non-deterministic Finite Automata (NFAs) are well-known to be sometimes exponentially more succinct than Deterministic Finite Automata (DFAs), because one can exhibit a family of languages  $L_n$  accepted by NFAs having  $O(n)$  states, but for which every DFA requires  $O(2^n)$  states.

### 3 Plan DLs based on Regular-Like Expressions

In this section we consider plan concept constructors for building ordinary REs, as well as conjunction/intersection (**and**/ $\sqcap$ ) and negation/complement (**not**/ $\neg$ ).

The utility of  $\sqcap$  arises in situations where one uses plan concepts as role restrictions in ordinary DLs. For example, if we want to relate a person to the sequences of steps they took for making phone calls, we could use for this purpose an ordinary role, **callsMade**, and include in the definition of  $\mathcal{AL}\mathcal{E}$ -concept **Persons** the role restriction

$$\forall \text{callsMade}.(\text{Dial} \circ \text{Ring}^* \circ (\text{Talk} \sqcup \text{Null}) \circ \text{HangUp})$$

If we now wanted to consider people who talked in at least one case before hanging up, we would conjoin to **Person** the restriction

$$\exists \text{callsMade}.(\text{Actions}^* \circ \text{Talk} \circ \text{Actions}^*)$$

This will require reasoning with

$$(\text{Actions}^* \circ \text{Talk} \circ \text{Actions}^*) \sqcap (\text{Dial} \circ \text{Ring}^* \circ (\text{Talk} \sqcup \text{Null}) \circ \text{HangUp})$$

In a different direction, suppose one used “sensing actions”, such as *IsOnHook?* to partially simulate conditional execution, as in *IsOnHook?*  $\circ$  *LiftReceiver*. In such situations one would want to avoid “illegitimate” sequences, such as *IsOnHook?* immediately followed by its opposite *IsNotOnHook?*. The absence of such plans could be detected by intersection with the concept

$$\neg ( (\text{Actions}^* \circ (\text{IsOnHook?} \circ \text{IsNotOnHook?}) \circ (\text{Actions}^*) ).$$

In discussing the variants of regular expressions, we will use the notation  $\text{RegExp}(\{\mathcal{S}\})$  to refer to the set of all regular-like expressions (over an implicit alphabet  $\Sigma$ ) built using constructors in  $\mathcal{S}$ . Thus  $\text{RegExp}(\{\circ, \sqcup, *\})$  refers to ordinary standard REs, while  $\text{RegExp}(\{\circ, \sqcup, *, \sqcap\})$  adds **and**.

A complete discussion of the individual complexity results for all the variants would take too much space, but is available in [5]. Instead, we summarize the results obtained by others and us at the top of Table 1 . The lines without

references at the end indicate that we provided the proofs. However, in our opinion these are not sufficiently deep to merit inclusion as separate theorems.

An interesting observation is that using unrestricted negation leads to very high complexity, even in the absence of looping. Define the (non-elementary) tower function  $tow$  recursively as  $tow(0, j) = j$ ,  $tow(k + 1, j) = 2^{tow(k, j)}$ ; it was proved in [59] that every problem in  $NSPACE(tow(\lceil \log_b(n) \rceil, 0))$  can be polynomially reduced to one in  $notTopPlan(\{\sqcup, \circ, \neg\})$ , while  $notTopPlan(\{\sqcup, \circ, \neg\})$  is itself not in  $NSPACE(tow(\lceil \log_b(n) \rceil, 0))$ .

Because regular languages are closed under intersection and complementation, these constructors do not increase expressive power. The situation with succinctness is very different, since *eliminating  $\sqcap$  or  $\neg$  can lead to double exponential blow up* [16]. More precisely, (a) For every integer  $n$ , there is  $r_n$  in  $RegExp(\{\circ, \sqcup, *, \sqcap, \neg\})$  of size  $O(n)$  such that any ordinary RE defining  $\neg r_n$  is of size at least  $2^{2^n}$ ; (b) for every integer  $n$ , there are REs  $r_1, \dots, r_m$ , with  $m = 2n + 1$ , of size  $O(n)$  such that any RE defining  $\bigcap_{i \leq m} r_i$  is of size at least  $2^{2^n}$ .

Problem	Reduction	Complexity [Ref.]
<b>(ordinary) RE-based Plan DL: <math>\{\sqcup, \circ, *\}</math></b>		
notEmpty	log-lin complete	NLOGSPACE [27] (citing [28])
containment	log complete	PSPACE [60]
containment	log-lin complete	NLINSPACE
member	log complete	NLOGSPACE [28]
<b>RE Plan DL + Conjunction: <math>\{\sqcup, \circ, *, \sqcap\}</math></b>		
notEmpty	complete	PSPACE [50] (citing [33])
nonEqual	complete	EXSPACE [15]
containment	hard	for EXSPACE [15]
member	log-lin complete	LOGCFL [50]
<b>RE Plan DL + Complement: <math>\{\sqcup, \circ, *, \neg\}</math></b>		
notEmpty		not bded by Elementary Fn [59]
nonEqual		in $NSPACE(tow(n, 0))$ [59]
nonEqual	poly-lin hard	$NSPACE(tow(\log_b(n), 0))$ [59]
member	log complete	P [24] (citing [49])
<b>RE Plan DL + top-level conjunction: <math>\{\sqcup, \circ, *, \text{top-level } \sqcap\}</math></b>		
notEmpty	log complete	PSPACE [33]
containment	complete	EXSPACE
member	log-lin	in LOGCFL
<b>RE Plan DL + complement nesting depth 1</b>		
notTopPlan	poly-lin complete	$NSPACE(\bigcup_d 2^{d \times n})$ [24]
<b>RE Plan DL + conjunction of possibly negated REs</b>		
containment	log-complete	EXSPACE

**Table 1.** Complexity Results for Extended RE-based Plan DLs

### 3.1 Plan Concepts as Concrete Domains

If we are interested in using RE-based plan concepts as universal/existential role restrictions in ordinary DLs with tableaux reasoners, then, following the work of Baader and Hanschke [2], this means that we can consider such plan concepts as unary predicates for a “concrete domain”. They have shown that for tableaux reasoning, in such cases it is sufficient that there be support for reasoning with the conjunction of these (possibly negated) predicates (so called “admissible domains”). This means that we are interested in the restricted use of  $\sqcap$  and  $\neg$ : a conjunction of possibly negated REs. By analyzing and extending some of the proofs for the cases of arbitrary intersection and complement, we obtain additional complexity results at the bottom of Table 1.

### 3.2 Counted Iteration, Subplans and Action Hierarchies

The original paper on CLASP [10] proposed a constructor **repeat**, which can be used as in the plan concept **repeat**(5, *DialOneDigit*). There is a widely studied formal language construct called “squaring”, where the extended regular expression  $(R)^2$  is simply a short form for  $R \circ R$ ; it can be simulated by the proportional size **repeat**(2, R). Hence the complexity results in Table 2 concerning  $(\cdot)^2$  provide hardness results for **repeat**.

It is also very convenient to break down the description of complex plans and processes into smaller, component plans. For example, *Dial* itself, in *MakeA-PhoneCall*, could have been defined as

$$Dial \doteq \mathbf{seq}(PickUpReceiver, ListenForTone, \mathbf{repeat}(10, DialOneNumber))$$

This can be viewed as allowing acyclic definitional TBoxes. The squaring operator for regular expressions can be immediately obtained by writing axioms like  $DoubleS \doteq S \circ S$ , and the complexity results for squaring then provide lower bounds for the complexity of reasoning with acyclic TBoxes in RE-based Plan DLs. Finally, in most applications, ontologies of plans or processes bottom out in taxonomies of primitive actions. To accommodate this, all that is needed is to replace action name  $b$  in a RE-based Plan DL expression by  $(b \sqcup c_1 \sqcup c_2 \dots)$ , where  $c_1, \dots$  are all the subclasses of  $b$ .

### 3.3 Adding Concurrency

Modeling concurrent execution of subprocesses or plans is key in certain situation (e.g., making a phone call while cooking). With sequences, we model conveniently the interleaving of primitive actions into traces (trace semantics/equivalence), rather than more refined notions of process equivalence/true parallelism (see [61, 26], say). The formal foundation is the notion of interleaving/shuffle:

**Definition 1** Given alphabet  $\Sigma$ , symbols  $a, b \in \Sigma$ , and sequences  $x, y \in \Sigma^*$ , the shuffle/interleaving  $x$  and  $y$ , written as  $x \# y$ , is defined recursively as follows:

$$\begin{aligned} a \# \lambda &= \lambda \# a = a \\ (a \circ s) \# (b \circ t) &= a \circ (s \# b \circ t) \sqcup b \circ (a \circ s \#) \text{ for } s, t \in \Sigma^* \end{aligned}$$

Shuffle is extended to languages as  $\mathcal{L}_1 \# \mathcal{L}_2 = \{u \# w \mid u \in \mathcal{L}_1, w \in \mathcal{L}_2\}$ .

We should point out that  $RegExp(\{\circ, \sqcup, *, \#\})$  corresponds to *structured workflows/Petri nets* [51, 31] and OWL-S[47], which have single entry/single exist components. In fact, the *process trees* in [35] are generalized to have arbitrary block-structured operators, thus fully resembling extended REs. Although less expressive than their full counterparts [31], such models are frequent in practice, and are advocated as having “better style” [6].

Let us consider the properties of this extension to RE-based Plan DLs. **Expressiveness:** Regular languages are easily shown to be closed under shuffle, so it does not increase expressive power. **Complexity of Reasoning:** Some known results for various reasoning tasks are summarized in the bottom half of Table 2. **Succinctness:** Gruber and Holzer [19] show that any ordinary regular expression defining the language  $(a_1 \circ b_1)^* \# (a_2 \circ b_2)^* \# \dots \# (a_n \circ b_n)^*$  must be of size at least *double exponential* in  $n$ .

## 4 Formalisms based on Finite State Machines

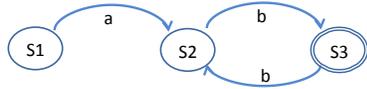
Finite state formalism have the advantage of being visually more perspicacious because of their graphical representation, which abounds in the process specification literature as illustrated by notations such as BPMN, Petri nets, UML activity diagrams, and Harel state charts. However, FAs are normally presented using transition matrixes, and the definition of “language accepted” is not in the usual Tarskian/compositional style. How can we present syntactically FAs as DLs with natural plan concept constructors? We could convert the FA to an RE, and then use the approach above in Sections 2 and 3. However, although there is a simple polynomial conversion from REs to FAs, the converse is not true. Ehrenfeucht and Zeiger [13] were the first to exhibit a simple family of FAs  $A_n$  whose corresponding REs grow exponentially in size: take a complete graph over states  $\{1, \dots, n\}$ , including self-loops, with distinct symbols labeling every edge (hence an alphabet of size  $n^2$ ); pick 1 and  $n$  as start and final states. They then show that  $A_n$  requires a regular expression of size at least  $2^{n-1}$  to describe the same language, though the size of  $A_n$  is  $O(n^2)$ .

Problem	Constructors	Reduction	Complexity
<b>Effect of adding squaring</b>			
nonEqual	$\{\sqcup, \circ, ^2\}$	log-lin complete	NEXPTIME [41]
nonEqual	$\{\sqcup, \circ, *, ^2\}$	log-lin complete	EXPSPACE [41]
member	$\{\sqcup, \circ, \sqcap, ^2\}$	log-lin complete	LOGCFL [50]
member	$\{\sqcup, \circ, \neg, ^2\}$	log complete	P [24] (citing [49])
<b>Effect of adding concurrency</b>			
notEmpty	$\{\sqcup, \circ, *, \#\}$		in P, based on [3]
nonEqual	$\{\sqcup, \circ, \#\}$	complete	$\Sigma_2^P$ [39]
containment	$\{\sqcup, \sqcup, *, \#\}$	complete	EXPSPACE [40]
member	$\{\sqcup, \#\} \{*, \#\}$	complete	NP [39]

**Table 2.** Complexity Results for RE-based Plan DLs with  $^2$  and  $\#$

#### 4.1 Representing FAs directly as DLs

We show by example how to convert an FA into a cyclic TBox, with only **seq** and **or** as plan concept constructors<sup>3</sup>. One starts by converting NFAs to Type 3 grammars, in the usual way [25]. For example, the FA



is translated to grammar (a) in Figure 2.

After conversion to EBNF notation, this can be viewed as a set of *language equations* [18, 34] (Figure 2(b)), where nonterminals are viewed as set-valued variables. By definition, the solutions of this set of equations are 3-tuples of languages  $(L1, L2, L3)$ , which when substituted for  $(S1, S2, S3)$ , make the equations be true. In turn these can be viewed as a Plan DL TBox using (Figure 2(c)), where, as usual, interpretations assign sets of strings to concept names (which are non-terminals here), and S1 is the concept denoting the language of the FA. Note that we only need plan constructors **seq** and **or**, but not **loop**.

$  \begin{aligned}  S1 &::= a S2 \\  S2 &::= b S3 \\  S3 &::= b S2 \\  S3 &::= \lambda  \end{aligned}  $	$  \begin{aligned}  S1 &= a \cdot S2 \\  S2 &= b \cdot S3 \\  S3 &= b \cdot S2 \cup \lambda  \end{aligned}  $	$  \begin{aligned}  S1 &\doteq \mathbf{seq}(a, S2) \\  S2 &\doteq \mathbf{seq}(b, S3) \\  S3 &\doteq \mathbf{or}(\mathbf{seq}(b, S2), \mathbf{Null})  \end{aligned}  $
(a)	(b)	(c)

**Fig. 2.** From automaton to TBox: an example

However, our TBox is cyclic so we have to consider the issue of possible alternative solutions, which relates to fixed point semantics. Grammar (and hence FA) languages have been shown to correspond to least fixed points [18]. Therefore we will adopt the same semantics. Incidentally, results by Okhotin [44] show that for the above kinds of equations the least, greatest and descriptive fixpoint semantics are identical.

#### 4.2 Reasoning with Finite Automata

We briefly consider the standard decision problems for Plan DLs, assuming they are specified as TBoxes derived from right linear grammars/finite automata. Note that we can reconstruct the FA immediately from the TBox. One reason to re-consider these issues is because of the NFA vs DFA distinction, which does not arise naturally for REs. The following results are from [23].

**Emptiness:** The non-emptiness problem for DFAs and NFAs is log-lin complete in NLOGSPACE.

**Subsumption:** The language containment problem for DFAs is NLOGSPACE-complete, and for NFAs it is log-lin complete for PSPACE.

<sup>3</sup> We emphasize from the beginning that we propose to recover the standard FA in order to implement algorithms. Our point is only that this seems like an elegant presentation of FAs as DLs with TBoxes.

**Membership:** The (general) membership problem for NFAs is complete for NLOGSPACE, but only LOGSPACE-complete for DFAs with respect to constant depth reducibilities.

## 5 Summary, Related and Future Work

The paper motivated the utility of Plan DLs for the use of plan/process ontologies in planning and plan recognition, the description of business processes and workflows, and the recognition of plan/process instances in many situations, including sensor-equipped environments and digital life-logging. We viewed plan concepts as denoting scenarios: sequences of property-less atomic activities. Starting from the work on the CLASP system, this led us to an obvious connection to regular-like expressions, which denote sets of strings, and a large family of Plan DLs, based on subsets of the considerable variety of RE-like constructors studied in the formal language literature. The corresponding collection of formal results about them, can be translated or improved into complexity results concerning RE-based Plan DL concept reasoning, and descriptive complexity results about their relative succinctness. We view part of the contribution of this paper the exposure of the rich literature which the DL community can tap into to obtain interesting results in the search for complexity-expressiveness trade-offs, for example.

We also studied the use of Plan DLs as "admissible concrete domains", and showed a way to view Finite Automata as DLs described by cyclic TBoxes, via language equations.

Some obvious problems left for future work are filling in holes in the complexity tables, considering data complexity for recognition, and using ABoxes to describe partially completed scenarios.

As mentioned earlier, two other strands of work in Description Logics fall under the category of having term constructors for process concepts. *Propositional Dynamic Logics* (PDL) allow programs to be described in a manner very similar to RE-based Plan DLs, with program constructors  $\{';', \cup, *\}$  corresponding to  $\{\circ, \sqcup, *\}$  for example. The work in [56, 7, 8] uses these as *role constructors*. PDLs are clearly more expressive since they combine action descriptions with state descriptions into formulas. However, this can lead to problems: while reasoning with conjunctions of RE-based Plan DLs was shown to be decidable in this paper, adding role conjunction to the set of role constructors leads to undecidability of the corresponding PDL-based DL. Other relevant recent work concerns Linear Dynamic Logic [9], and Declarative Business Process Models (e.g., [48]), also based on linear temporal logic (LTL).

LTL also plays a role in recent proposals for temporal DLs with modal operators [38] which avoid using variables. For example,  $\diamond(EU_{candidate} \mathcal{U} EU_{member})$  has a clear term-like expression **eventually**(**until**( $EU_{candidate}$ ,  $EU_{member}$ )).

The exact relationships of these to RE-based Plan DLs is the subject of ongoing research.

## References

1. Franz Baader. A formal definition for the expressive power of terminological knowledge representation languages. *J. Logic and Computation*, 6(1):33–54, 1996.
2. Franz Baader and Philipp Hanschke. A scheme for integrating concrete domains into concept languages. In *Proc. IJCAI'91*, pages 452–457, 1991.
3. M. Berglund, H. Björklund, and J. Björklund. Shuffled languages—representation and recognition. *Theoretical Computer Science*, 489:1–20, 2013.
4. Alexander Borgida. Towards the systematic development of description logic reasoners: CLASP reconstructed. In *Proc. KR'92*, pages 259–269, 1992.
5. Alexander Borgida. Initial steps towards a family of regular-like plan description logics. volume 11560 of *Lecture Notes in Computer Science*. Springer, 2019. To appear.
6. Flavio Corradini, Alessio Ferrari, Fabrizio Fornari, Stefania Gnesi, Andrea Polini, Barbara Re, and Giorgio Oronzo Spagnolo. A guidelines framework for understandable BPMN models. *Data Knowl. Eng.*, 113:129–154, 2018.
7. Giuseppe De Giacomo and Maurizio Lenzerini. Boosting the correspondence between description logics and propositional dynamic logics. In *Proc. AAAI'94*, pages 205–212. AAAI Press, 1994.
8. Giuseppe De Giacomo and Maurizio Lenzerini. Tbox and abox reasoning in expressive description logics. In *Proc. AAAI'96*, pages 37–48. AAAI Press, 1996.
9. Giuseppe De Giacomo and Moshe Y. Vardi. Linear temporal logic and linear dynamic logic on finite traces. In *Proc. IJCAI*, pages 854–860. IJCAI/AAAI, 2013.
10. P. T. Devanbu and D. J. Litman. Taxonomic plan reasoning. *Artificial Intelligence*, 84(1-2):1–35, 1996.
11. Prem Devanbu, Ronald J Brachman, Peter G Selfridge, and Bruce W Ballard. Lassie: A knowledge-based software information system. In *Proc. 12th International Conference on Software Engineering*, pages 249–261. IEEE, 1990.
12. Premkumar T. Devanbu and Diane J. Litman. Plan-based terminological reasoning. In *Proc. KR'91*, pages 128–138. Morgan Kaufmann, 1991.
13. A. Ehrenfeucht and P. Zeiger. Complexity measures for regular expressions. *Journal of computer and system sciences*, 12(2):134–146, 1976.
14. Emily P. Friedman. The inclusion problem for simple languages. *Theor. Comput. Sci.*, 1(4):297–316, 1976.
15. Martin Fürer. The complexity of the inequivalence problem for regular expressions with intersection. In *Automata, Languages and Programming, 7th Colloquium*, pages 234–245, 1980.
16. W. Gelade and F. Neven. Succinctness of the complement and intersection of regular expressions. *ACM Transactions on Computational Logic*, 4(1):1–19, 2012.
17. Yolanda Gil. Description logics and planning. *AI Magazine*, 26(2):73–84, 2005.
18. S. Ginsburg and H. G. Rice. Two families of languages related to algol. *Journal of the ACM (JACM)*, 9(3):350–371, 1962.
19. Hermann Gruber and Markus Holzer. Tight bounds on the descriptiveness of regular expressions. In *Developments in Language Theory*, pages 276–287, 2009.
20. Cathal Gurrin, Alan F Smeaton, and Aiden R Doherty. Lifelogging: Personal big data. *Foundations and Trends® in Information Retrieval*, 8(1):1–125, 2014.
21. Babak Bagheri Hariri, Diego Calvanese, Marco Montali, Giuseppe De Giacomo, Riccardo De Masellis, and Paolo Felli. Description logic knowledge and action bases. *J. Artif. Intell. Res.*, 46:651–686, 2013.

22. Rim Helaoui, Daniele Riboni, and Heiner Stuckenschmidt. A probabilistic ontological framework for the recognition of multilevel human activities. In *Proc. UbiComp*, pages 345–354. ACM, 2013.
23. M. Holzer and M. Kutrib. Descriptive and computational complexity of finite automata—a survey. *Information and Computation*, 209(3):456–470, 2011.
24. Markus Holzer and Martin Kutrib. The complexity of regular(-like) expressions. In *Developments in Language Theory*, pages 16–30, 2010.
25. John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to automata theory, languages, and computation*. Addison-Wesley, 2003.
26. Lalita Jategaonkar and Albert R Meyer. Deciding true concurrency equivalences on safe, finite nets. *Theoretical Computer Science*, 154:107–143, 1996.
27. Tao Jiang and Bala Ravikumar. A note on the space complexity of some decision problems for finite automata. *Inf. Process. Lett.*, 40(1):25–31, 1991.
28. Neil D. Jones. Space-bounded reducibility among combinatorial problems. *J. Comput. Syst. Sci.*, 11(1):68–85, 1975.
29. E.-M. Kalogeraki, D. Apostolou, T. Panayiotopoulos, G. Tsihrintzis, and S. Theocharis. A semantic approach for representing and querying business processes. In *Intelligent Computing Systems*, pages 87–114. Springer, 2016.
30. Varvara Kalokyri, Alexander Borgida, and Amélie Marian. Yourdigitalself: A personal digital trace integration tool. In *CIKM*, pages 1963–1966. ACM, 2018.
31. Bartek Kiepuszewski, Arthur Harry Maria Ter Hofstede, and Christoph J Bussler. On structured workflow modelling. In *Proc. CAiSE*, pages 431–445. Springer, 2000.
32. M. Klusch, P. Kapahnke, S. Schulte, F. Lecue, and A. Bernstein. Semantic web service search: a brief survey. *KI-Künstliche Intelligenz*, 30(2):139–147, 2016.
33. Dexter Kozen. Lower bounds for natural proof systems. In *18th Annual Symposium on Foundations of Computer Science*, pages 254–266, 1977.
34. M. Kunc. What do we know about language equations? In *Int. Conf. on Developments in Language Theory*, pages 23–27, Berlin, 2007. Springer.
35. Sander J. J. Leemans, Dirk Fahland, and Wil M. P. van der Aalst. Discovering block-structured process models from incomplete event logs. In *Petri Nets*, volume 8489 of *Lecture Notes in Computer Science*, pages 91–110. Springer, 2014.
36. A. L. Lemos, F. Daniel, and B. Benatallah. Web service composition: a survey of techniques and tools. *ACM Computing Surveys*, 48(3):1–41, 2016.
37. Ruopeng Lu and Shazia Wasim Sadiq. A survey of comparative business process modeling approaches. In *Proc. BIS*, volume 4439 of *Lecture Notes in Computer Science*, pages 82–94. Springer, 2007.
38. Carsten Lutz, Frank Wolter, and Michael Zakharyashev. Temporal description logics: A survey. In *Proc. Temporal Representation and Reasoning*, pages 3–14. IEEE, 2008.
39. A. J. Mayer and L. J. Stockmeyer. The complexity of word problems—this time with interleaving. *Information and Computation*, 115(2):293–311, 1994.
40. A. J. Mayer and L. J. Stockmeyer. The complexity of pdl with interleaving. *Theoretical Computer Science*, 161(1-2):109–122, 1996.
41. Albert R. Meyer and Larry J. Stockmeyer. The equivalence problem for regular expressions with squaring requires exponential space. In *13th Annual Symposium on Switching and Automata Theory*, pages 125–129, 1972.
42. Robin Milner, Joachim Parrow, and David Walker. A calculus of mobile processes, i. *Information and computation*, 100(1):1–40, 1992.
43. Christine Natschläger. Towards a bpmn 2.0 ontology. In *International Workshop on Business Process Modeling Notation*, pages 1–15. Springer, 2011.

44. A. S. Okhotin. Conjunctive grammars and systems of language equations. *Programming and Computer Software*, 28(5):243–249, 2002.
45. OMG. Business Process Model and Notation (BPMN), Version 2.0, 2011.
46. Chun Ouyang, Michael Adams, Moe Thandar Wynn, and Arthur HM ter Hofstede. Workflow management. In *Handbook on Business Process Management 1*, pages 475–506. Springer, 2015.
47. OWL-S Coalition. OWL-S 1.1 Release, 2004.
48. M. Pesic, H. Schonenberg, and W. Van der Aalst. Declare: Full support for loosely-structured processes. In *EDOC'07*, pages 287–287. IEEE, 2007.
49. Holger Petersen. Decision problems for generalized regular expressions. In *Proc. Descr. Complexity of Automata, Grammars and Related Structures*, pages 22–29, 2000.
50. Holger Petersen. The membership problem for regular expressions with intersection is complete in LOGCFL. In *Proc. STACS'02*, pages 513–522, 2002.
51. Manfred Reichert and Peter Dadam. Adept flexsupporting dynamic changes of workflows without losing control. *Journal of Intelligent Information Systems*, 10(2):93–129, 1998.
52. Daniele Riboni and Claudio Bettini. Owl 2 modeling and reasoning with complex human activities. *Pervasive and Mobile Computing*, 7(3):379–395, 2011.
53. Natalia Díaz Rodríguez, M. P. Cuéllar, Johan Lilius, and Miguel Delgado Calvo-Flores. A survey on ontologies for human behavior recognition. *ACM Comput. Surv.*, 46(4):1–33, 2014.
54. Marco Rospoche, Chiara Ghidini, and Luciano Serafini. An ontology for the business process modelling notation. In *Proc. FOIS'14*, pages 133–146, 2014.
55. James Rumbaugh, Ivar Jacobson, and Grady Booch. *Unified Modeling Language Reference Manual, The (2nd Edition)*. Pearson Higher Education, 2004.
56. Klaus Schild. A correspondence theory for terminological logics: Preliminary report. In *Proc. IJCAI'91*, pages 466–471. Morgan Kaufmann, 1991.
57. Albrecht Schmiedel. Temporal terminological logic. In *Proc. AAAI'90*, pages 640–645, 1990.
58. Géraud Sénizergues. The equivalence problem for deterministic pushdown automata is decidable. In *Proc. ICALP'97*, pages 671–681. Springer, 1997.
59. L. J. Stockmeyer. *The Complexity of Decision Problems in Automata Theory and Logic*. PhD thesis, Massachusetts Institute of Technology, Cambridge, Massachusetts, 1974.
60. L. J. Stockmeyer and A. R. Meyer. Word problems requiring exponential time. In *Proc. Symposium on Theory of Computing (STOC 1973)*, pages 1–9, 1973.
61. Rob J. van Glabbeek. The linear time-branching time spectrum (extended abstract). In *CONCUR*, volume 458 of *Lecture Notes in Computer Science*, pages 278–297. Springer, 1990.
62. Tim LM van Kasteren, Gwenn Englebienne, and Ben JA Kröse. Human activity recognition from wireless sensor network data: Benchmark and software. In *Activity recognition in pervasive intelligent environments*, pages 165–186. Springer, 2011.
63. Robert Weida. Knowledge representation for plan recognition. In *IJCAI'95 Workshop on the Next Generation of Plan Recognition Systems*, 1995.
64. J. Ye, S. Dasiopoulou, G. Stevenson, G. Meditskos, E. Kontopoulos, I. Kompatsiaris, and S. Dobson. Semantic web technologies in pervasive computing: A survey and research roadmap. *Pervasive and Mobile Computing*, 23:1–25, 2015.