THEORY AND PRACTICE
OF LOGIC PROGRAMMING

CAMBRIDGE
UNIVERSITY PRESS

Knowledge Authoring for Rules and Actions

Journal:	*Theory and Practice of Logic Programming*
Manuscript ID	Draft
Manuscript Type:	Original Article
Date Submitted by the Author:	n/a
Complete List of Authors:	Wang, Yuheng; Stony Brook University, Fodor, Paul; Stony Brook University, Computer Science Kifer, Michael; Stony Brook University, Computer Science
Keywords:	Knowledge Representation and Nonmonotonic Reasoning
Computing Classification System (CCS):	

Abstract:

Knowledge representation and reasoning (KRR) systems describe and reason with complex concepts and relations in the form of facts and rules. Unfortunately, wide deployment of KRR systems runs into the problem that domain experts have great difficulty constructing correct logical representations of their domain knowledge. Knowledge engineers can help with this construction process, but there is a deficit of such specialists. The earlier Knowledge Authoring Logic Machine (KALM) based on Controlled Natural Language
(CNL) was shown to have very high accuracy for authoring facts and questions. More recently, KALMFL,
a successor of KALM, replaced CNL with factual English, which is much less restrictive and requires very
little training from users. However, KALMFL has limitations in representing certain types of knowledge,
such as authoring rules for multi-step reasoning or understanding actions with timestamps. To address these
limitations, we propose KALMRA to enable authoring of rules and actions.

	Our evaluation using the UTI guidelines benchmark shows that KALMRA achieves a high level of correctness (100%) on rule authoring. When used for authoring and reasoning with actions, KALMRA achieves more than 99.3% correctness on the bAbI benchmark, demonstrating its effectiveness in more sophisticated KRR jobs.

Under consideration for publication in Theory and Practice of Logic Programming 1

Knowledge Authoring for Rules and Actions∗

YUHENG WANG, PAUL FODOR, and MICHAEL KIFER
Stony Brook University, Stony Brook, NY, USA
(*e-mail:* {yuhewang,pfodor,kifer}@cs.stonybrook.edu)

submitted 1 January 2003; revised 1 January 2003; accepted 1 January 2003

Abstract

Knowledge representation and reasoning (KRR) systems describe and reason with complex concepts and relations in the form of facts and rules. Unfortunately, wide deployment of KRR systems runs into the problem that domain experts have great difficulty constructing correct logical representations of their domain knowledge. Knowledge engineers can help with this construction process, but there is a deficit of such specialists. The earlier Knowledge Authoring Logic Machine (KALM) based on Controlled Natural Language (CNL) was shown to have very high accuracy for authoring facts and questions. More recently, KALM^{FL}, a successor of KALM, replaced CNL with *factual* English, which is much less restrictive and requires very little training from users. However, KALM^{FL} has limitations in representing certain types of knowledge, such as authoring rules for multi-step reasoning or understanding actions with timestamps. To address these limitations, we propose KALM^{RA} to enable authoring of rules and actions. Our evaluation using the UTI guidelines benchmark shows that KALM^{RA} achieves a high level of correctness (100%) on rule authoring. When used for authoring and reasoning with actions, KALM^{RA} achieves more than 99.3% correctness on the bAbI benchmark, demonstrating its effectiveness in more sophisticated KRR jobs. Finally, we illustrate the logical reasoning capabilities of KALM^{RA} by drawing attention to the problems faced by the recently made famous AI, ChatGPT.

KEYWORDS: knowledge authoring, knowledge representation and reasoning, natural language understanding, frame-based parsing

1 Introduction

Knowledge representation and reasoning (KRR) systems represent human knowledge as facts, rules, and other logical forms. However, transformation of human knowledge to these logical forms requires the expertise of knowledge engineers with KRR skills, which, unfortunately, is scarce.

To address the shortage of knowledge engineers, researchers have explored the use of different languages and translators for representing human knowledge. One idea was to use natural language (NL), but the NL-based systems, such as OpenSesame (Swayamdipta et al. 2017) and SLING (Ringgaard et al. 2017), had low accuracy, and led to significant errors in subsequent reasoning. The accuracy issue then motivated researchers to consider Controlled Natural Language (CNL) (Fuchs et al. 2008; Schwitter 2002) for knowledge authoring. Unfortunately, although CNL does improve accuracy, it is hard for a typical user (say, a domain expert) to learn a CNL grammar and its syntactic restrictions. Furthermore, systems based on either NL or CNL cannot

∗ Research partially funded by NSF grant 1814457.

identify sentences with the same meaning but different forms. For example, "*Mary buys a car*" and "*Mary makes a purchase of a car*" would be translated into totally different logical representations. This problem, known as *semantic mismatch* (Gao et al. 2018a), is a serious limitation affecting accuracy.

The Knowledge Authoring Logic Machine (KALM) (Gao et al. 2018b) was introduced to tackle semantic mismatch problem, but this approach was based on a CNL (Attempto (Fuchs et al. 2008)) and had heavy syntactic limitations. Recently, the KALM^{FL} system (Wang et al. 2022) greatly relaxed these restrictions by focusing on *factual* English sentences, which are suitable for expressing facts and queries and require little training to use. To parse factual sentences, KALM^{FL} replaced the CNL parser in the original KALM system with an improved neural NL parser called mSTANZA. However, this alteration brought about several new issues that are typical in neural parsers, such as errors in part-of-speech and dependency parsing. KALM^{FL} then effectively addressed these issues and achieved high accuracy in authoring facts and queries with factual sentences.

In this paper, we focus on other types of human knowledge that KALM^{FL} does not cover, such as, rules and actions. We further extend KALM^{FL} to support authoring of rules and actions, creating a new system called KALM for Rules and Actions (or KALM^{RA}).[1] KALM^{RA} allows users to author rules using factual sentences and perform multi-step frame-based reasoning using F-logic (Kifer and Lausen 1989). In addition to rule authoring, KALM^{RA} incorporates a formalism known as Simplified Event Calculus (SEC) (Sadri and Kowalski 1995) to represent and reason about actions and their effects. The use of authored knowledge (facts, queries, rules, and actions) allows for logical reasoning within an *underlying logical system for reasoning with the generated knowledge*. This system must align with the scope of the knowledge that KALM^{RA} can represent, and supports the inference of new knowledge from existing one. In terms of implementation, we found a Prolog-like system is more suitable for frame-based parsing, so we implemented KALM^{RA} in XSB (Swift and Warren 2012). However, the knowledge produced by KALM^{RA} contains disjunctive knowledge and function symbols, so we chose the answer set programming system DLV (Leone et al. 2006) as the logical system for reasoning about the generated knowledge.[2] Evaluation on benchmarks including the UTI guidelines (Shiffman et al. 2009) and bAbI Tasks (Weston et al. 2015) shows that KALM^{RA} achieves 100% accuracy on authoring and reasoning with rules, and 99.3% on authoring and reasoning about actions. Finally, we assess the recently released powerful dialogue model, ChatGPT[3], using bAbI Tasks, and highlight its limitations with respect to logical reasoning compared to KALM^{RA}.

The paper is organized as follows: Section 2 reviews the KALM^{FL} system and some logic programming techniques, Section 3 introduces the new KALM^{RA} system and describes how it represents rules and actions, Section 4 presents the evaluation settings and results, and Section 5 concludes the paper and discusses future work.

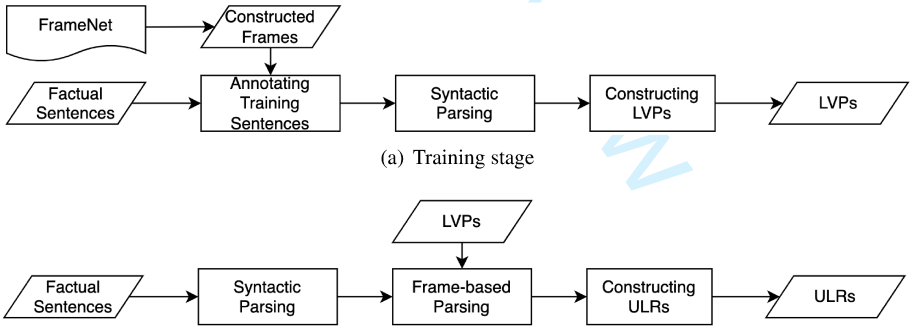
[1] `https://github.com/yuhengwang1/kalm-ra`
[2] Other ASP logic programming systems, such as Potassco (Gebser et al. 2019), lack the necessary level of support for function symbols and querying.
[3] `https://chat.openai.com/chat`

2 Background

2.1 Knowledge Authoring Logic Machine for Factual Language

The Knowledge Authoring Logic Machine (KALM) (Gao et al. 2018b; Gao et al. 2018a) allows users to author knowledge using Attempto Controlled English (ACE) (Fuchs and Schwitter 1996). However, ACE's grammar is too limiting and poses a high learning curve, particularly for non-technical users. To mitigate this problem, KALM was extended to KALM for Factual (English) Language (KALM^{FL}) (Wang et al. 2022) by introducing *factual (English) sentences* and focusing on authoring facts and simple queries. Factual sentences express atomic database facts and queries (*e.g.* "*Mary buys a car*"). They can become more complex with adnominal clauses (*e.g.* "*Mary buys a car that is old*") and can be combined via "*and*" and "*or*" (*e.g.* "*Mary buys a car and Bob buys a watch*"). In comparison, sentences not expressing factual information (*e.g.* "*Fetch the ball*" or "*Oh, well*") are non-factual and are not allowed. Factual sentences can be captured through properties based on dependency analysis and Part-of-Speech (POS) tagging (Wang et al. 2022), which is a very mild restriction compared to complex grammars such as in ACE. This means that users do not need to master complex grammars. Instead, they can simply write normal sentences that describe database facts, or basic Boolean combinations of facts and, as long as they avoid fancy language forms, their sentences will be accepted.

KALM^{FL} is a two-stage system following the *structured machine learning* paradigm. In the first stage, known as the *training stage*, KALM^{FL} constructs *logical valence patterns* (LVPs) by learning from *training sentences*. An LVP is a specification that tells how to extract *role fillers* for the concepts represented by the English sentences related to that LVP. In the second stage, known as the *deployment stage*, the system does semantic parsing by applying the constructed LVPs to convert factual English sentences into *unique logical representations* (ULRs). Fig. 1(a) depicts the training stage of KALM^{FL}, with the key steps explained in the accompanying text.



(a) Training stage

(b) Deployment stage

Fig. 1. The frameworks of the KALM^{FL} system

Annotating Training Sentences. To enable semantic understanding of a domain of discourse, knowledge engineers must first construct the required background knowledge in the form of KALM^{FL} frames. The overall structure of most of these frames can be adopted from FrameNet (Baker et al. 1998) and converted into the logic form required by KALM^{FL}. Then, knowledge engineers compose training sentences and annotate them using KALM^{FL} frames. For example, the annotated training sentence (1), below, indicates that the meaning of "*Mary buys a car*" is captured by the `Commerce_buy` frame; the word that triggers this frame, a.k.a. the *lexical unit* (LU), is the

2nd word "*buy*" or its synonym "*purchase*"; and, the 1st and the 4th words, "*mary*" and "*car*", play the roles of `Buyer` and `Goods` in the frame.

```
train("Mary buys a car","Commerce_buy","LU"=2,[purchase],
      ["Buyer"=1+required,"Goods"=4+required]).
```
(1)

Syntactic Parsing. KALM^{FL} then performs syntactic parsing using mSTANZA[4] (Wang et al. 2022) and automatically corrects some parsing errors. Fig. 2 shows two mSTANZA parses, where the colored boxes contain POS tags and the labeled arrows display dependency relations.

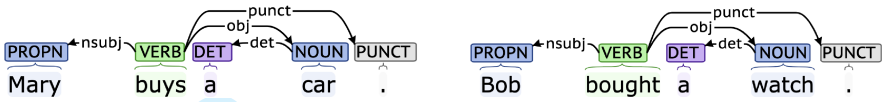


Fig. 2. mSTANZA parses

Construction of LVPs. mSTANZA parses, along with annotations of sentences, allow KALM^{FL} to construct LVPs that specify how to fill the roles of a frame triggered by an LU. For example, by synthesizing the information in training sentence (1) and the mSTANZA parse in Fig. 2(a), KALM^{FL} learns that, to fill the roles `Buyer` and `Goods` of the `Commerce_buy` frame triggered by the LU "*buys*", one should extract the subject and object of "*buys*" through the dependency relations `nsubj` and `obj`, respectively. This learned knowledge about role-filling is encoded as an LVP (2) as follows:

```
lvp(buy,"Commerce_buy",[pattern("Buyer",[nsubj],required),
                        pattern("Goods",[obj],required)]).
```
(2)

The deployment stage of KALM^{FL} is illustrated in Fig. 1(b). Two key steps in this stage are further explained below.

Frame-based Parsing. When an unseen factual sentence comes in, KALM^{FL} triggers all possible LVPs using the words in the sentence. Then, the triggered LVPs are applied to this sentence to extract role fillers and a frame-based parse of this sentence is generated. For example, a new sentence "*Bob bought a watch*" is parsed as Fig. 2(b) by mSTANZA. It triggers LVP (2) using the LU "*bought*" (whose base form is "*buy*"). KALM^{FL} then extracts the role filler "*Bob*" for the role `Buyer` according to the dependency list `[nsubj]`. Similarly, "*watch*" is extracted for the role `Goods` according to the dependency `[obj]`.

Constructing ULRs. Ultimately, frame-based parses are represented as ULR facts that capture the meaning of the original English sentences and are suitable for querying. For example, the ULR for the factual sentence "*Bob bought a watch*", below, indicates that the meaning of the sentence is captured by the `Commerce_buy` frame, that "*Bob*" is the `Buyer`, and "*watch*" plays the role of `Goods`, where `rl/2` represents instances of (role, role-filler) pairs.

```
frame("Commerce_buy",[rl("Buyer","Bob"),rl("Goods",watch)]).
```

[4] A modification of STANZA (Qi et al. 2020) that returns ranked lists of parses rather than just one parse.

2.2 Disjunctive Information and Frame Reasoning

Our reasoning subsystem combines Answer Set Programming (ASP) with aspects of frame-based reasoning.

DLV (Leone et al. 2006) is a disjunctive version of Datalog that operates under the ASP paradigm. It extends Datalog by adding support for disjunction in facts and rule heads, thus providing greater expressiveness for disjunctive information than KRR systems based on the well-founded semantics (*e.g.*, XSB (Swift and Warren 2012)). Furthermore, DLV's support for function symbols and querying makes it more convenient for working with frames (Fillmore et al. 2006) than other ASP systems, such as Potassco (Gebser et al. 2019).

F-logic (Kifer et al. 1995; Kifer and Lausen 1989) is a knowledge representation and ontology language that combines the benefits of conceptual modeling with object-oriented and frame-based languages. One of its key features is the ability to use *composite frames* to reduce long conjunctions of roles into more compact forms, matching ideally the structure of FrameBase's frames. For example, F-logic frames[5] can be used to answer the question "*What did Mary buy?*" given the fact "*Mary bought a car for Bob*," whose ULRs, shown below, are not logically equivalent (the fact has more roles than the query).

```
frame("Commerce_buy",[rl("Buyer","Mary"),rl("Goods",car),rl("Recipient","Bob")]).
?- frame("Commerce_buy",[rl("Buyer","Mary"),rl("Goods",What)]).   What=car.
```

2.3 Event Calculus for Reasoning about Actions and their Effects

The event calculus (EC) (Kowalski and Sergot 1989) is a set of logical axioms that describe the *law of inertia* for actions. This law states that time-dependent facts, *fluents*, that are not explicitly changed by an action preserve their true/false status in the state produced by that action. Here we use the simplified event calculus (SEC) (Sadri and Kowalski 1995), which is a simpler and more tractable variant of the original EC. A fluent in SEC is said to hold at a particular timestamp if it is initiated by an action and not terminated subsequently. This is formalized by these DLV rules:

```
holdsAt(F,T2) :-
    happensAt(A,T1), initiates(A,F), timestamp(T2), T1 < T2,
    not stoppedIn(T1,F,T2).
stoppedIn(T1,F,T2) :-
    happensAt(A,T), terminates(A,F), timestamp(T1), T1 < T, timestamp(T2), T < T2.
```

Here `happensAt/2` represents a momentary occurrence of action `A` at a timestamp. If an action is exogenous insertion of a fluent f at time t then we also represent it as `happensAt(f,t)`. Example 1 demonstrates the use of `happensAt/2`.

Example 1 The sentence "*Mary goes to the bedroom. The bedroom is north of the garden.*" is represented as follows:

```
happensAt(frame("Travel",[rl("Person","Mary"),rl("Place",bedroom)]),1).
happensAt(frame("North_of",[rl("Entity1",bedroom),rl("Entity2",garden)]),2).
person("Mary"). place(bedroom). entity(bedroom). entity(garden). timestamp(1..2).
```

[5] We depart from the actual syntax of F-logic as it is not supported by the DLV system. Instead, we implemented a small subset of that logic by casting it directly into the already supported DLV syntax.

The first `happensAt/2` introduces an action of traveling from place to place while the second `happensAt/2` uses an observed (i.e., exogenously inserted) fluent `"North_of"(bedroom,garden)`. Observable fluents are supposed to be disjoint from action fluents, and we will use a special predicate, `observable/1`, to recognize them in SEC rules. Timestamps indicate the temporal relation between the action and the observed fluent. Predicates `person/1`, `place/2`, `entity/2`, define the domain of roles, while `timestamp/1` restricts the domain of timestamps.

The predicates `initiates(Action,Fluent)` and `terminates(Action,Fluent)` in SEC are typically used to specify domain-specific axioms that capture the initiation and termination of fluents.

3 Extending KALM^{FL} or Rules and Actions

This section describes an extension of KALM^{FL} to handle rules and actions (KALM^{RA}).

Since we want to be able to handle disjunctive information required by some of the bAbI tasks, we made a decision to switch the reasoner from XSB which was used in KALM^{FL} to an ASP-based system DLV (Leone et al. 2006) that can handle disjunction in the rule heads. Thus, the syntax of the ULR, i.e., the logical statements produced by KALM^{RA}, follows that of DLV. A number of examples inspired by the UTI guidelines and bAbI Tasks are used in this section to illustrate the workings of KALM^{RA}.

3.1 Authoring and Reasoning with KALM^{RA} ules

Rules are important to KRR systems because they enable multi-step logical inferences needed for real-world tasks, such as diagnosis, planning, and decision making. Here we address the problem of rule authoring.

3.1.1 Enhancements for Representation of Facts

First we discuss the representation of disjunction, conjunction, negation, and coreference, which is not covered in KALM^{FL}.

Conjunction and Disjunction. The KALM^{RA} system prohibits the use of a mixture of conjunction and disjunction within a single factual sentence to prevent ambiguous expressions such as "*Mary wants to have a sandwich or a salad and a drink.*" To represent a factual sentence with homogeneous conjunction or disjunction, the system first parses the sentence into a set of component ULRs. For conjunction, KALM^{RA} uses this set of ULRs as the final representation. For disjunction, the component ULRs are assembled into a single disjunctive ULR using DLV's disjunction `v` as shown in Example 2.

Example 2 The factual sentence with conjunction "*Daniel administers a parenteral and an oral antimicrobial therapy for Mary*" is represented as the following set of ULRs:

```
frame("Cure",[rl("Doctor","Daniel"),rl("Patient","Mary"),
              rl("Therapy",antimicrobial),rl("Method",parenteral)]).
frame("Cure",[rl("Doctor","Daniel"),rl("Patient","Mary"),
                    rl("Therapy",antimicrobial),rl("Method",oral)]).
doctor("Daniel"). patient("Mary"). therapy(antimicrobial).
method(parenteral). method(oral).
```

where the predicates `doctor`, `patient`, `therapy`, and `method` define the domains for the roles. These domain predicates will be omitted in the rest of the paper, for brevity.

The disjunctive factual sentence "*Daniel administers a parenteral or an oral antimicrobial therapy for Mary*" is represented as the following ULR:

```
frame("Cure",[rl("Doctor","Daniel"),rl("Patient","Mary"),
              rl("Therapy",antimicrobial),rl("Route",parenteral)])
v frame("Cure",[rl("Doctor","Daniel"),rl("Patient","Mary"),
                rl("Therapy",antimicrobial),rl("Route",oral)]).
```

Negation. The KALM^{RA} system supports *explicit negation* through the use of the negative words "*not*" and "*no*". Such sentences are captured by appending the suffix "`_not`" to the name of the frame triggered by this sentence.

Example 3 The explicitly negated factual sentence "*Daniel's patient Mary does not have UTI*" is represented by

```
frame("Medical_issue_not",[rl("Doctor","Daniel"),rl("Patient","Mary"),
                           rl("Ailment","UTI")]).
```

Coreference. Coreference occurs when a word or a phrase refers to something that is mentioned earlier in the text. Without coreference resolution, one gets unresolved references to unknown entities in ULRs. To address this issue, KALM^{RA} uses a coreference resolution tool neuralcoref,[6] which identifies and replaces coreferences with the corresponding entities from the preceding text.

Example 4 The factual sentences "*Daniel's patient Mary has UTI. He administers an antimicrobial therapy for her.*" are turned into

```
frame("Medical_issue",[rl("Doctor","Daniel"),rl("Patient","Mary"),
                       rl("Ailment","UTI")]).
frame("Cure",[rl("Doctor","Daniel"),rl("Patient","Mary"),
              rl("Therapy",antimicrobial)]).
```

where the second ULR uses entities `"Daniel"` and `"Mary"` instead of the pronouns "*he*" and "*she*."

3.1.2 Rule Representation

Rules in KALM^{RA} are expressed in a much more restricted syntax compared to facts since, for knowledge authoring purposes, humans have little difficulty learning and complying with the restrictions. Moreover, since variables play such a key role in rules, complex coreferences must be specified unambiguously. All this makes writing rules in a natural language into a very cumbersome, error-prone, and ambiguity-prone task compared to the restricted syntax below.

Definition 1 A *rule* in KALM^{RA} is an if-then statement of the form "If P_1, P_2, ..., and P_n, then C_1, C_2, ..., or C_m", where

1. each P_i ($i = 1..n$) is a factual sentence without disjunction;
2. each C_j ($j = 1..m$) is a factual sentence without conjunction;

[6] `https://github.com/huggingface/neuralcoref`

3. variables in C_j ($j = 1..m$) must use the *explicitly typed syntax* (Gao et al. 2018a) and must appear in at least one of the P_i ($i = 1..n$). *E.g.*, in the rule "*If Mary goes to the hospital, then $doctor sees Mary*", the explicitly typed variable *$doctor* appears in the conclusion without appearing in the premise, which is prohibited. Instead, the rule author must provide some information about the doctor in a rule premise (*e.g.*, "and she has an appointment with *$doctor*"). This corresponds to the well-known "rule safety" rule in logic programming.
4. variables that refer to the same thing must have the same name. *E.g.*, in the rule "*If $patient is sick, then $patient goes to see a doctor*", the two *$patient* variables are intended to refer to the same person and thus have the same name.

Here are some examples of rules in KALM^{RA}.

Example 5 The KALM^{RA} rule "If *$doctor's $patient is a young child and has an unexplained fever*, then *$doctor assesses $patient's degree of toxicity or dehydration*" is represented as follows:

```
frame("Assessing",[rl("Doctor",Doctor),rl("Patient",Patient),
                   rl("Item",toxicity)])
v frame("Assessing",[rl("Doctor",Doctor),rl("Patient",Patient),
                     rl("Item",dehydration)]) :-
    frame("People_by_age",[rl("Person",Patient),rl("Type",child)]),
    frame("Medical_issues",[rl("Doctor",Doctor),rl("Patient",Patient),
                            rl("Ailment",fever),rl("Cause",unexplained)]).
```

KALM^{RA} supports two types of negation in rules: *explicit negation* (Gelfond and Lifschitz 1991) and *negation as failure* (with the stable model semantics (Gelfond and Lifschitz 1988)). The former allows users to specify explicitly known negative factual information while the latter lets one derive negative information from the lack of positive information. Explicit negation in rules is handled the same way as in fact representation. Negation as failure must be indicated by the rule author through the idiom "*not provable*", which is then converted into the predicate `not/1`. The idiom "*not provable*" is prohibited in rule heads.

Example 6 The KALM^{RA} rule "If *not provable $doctor does not administer $therapy for $patient*, then *$patient undergoes $therapy from $doctor*"is represented as follows:

```
frame("Undergoing",[rl("Doctor",Doctor),rl("Patient",Patient),
                    rl("Therapy",Therapy)]) :-
    not frame("Cure_not",[rl("Doctor",Doctor),rl("Patient",Patient),
                          rl("Therapy",Therapy)]),
    patient(Patient), doctor(Doctor), therapy(Therapy).
```

where `patient/1`, `doctor/1`, and `therapy/1` are domain predicates that ensure that variables that appear under negation have well-defined domains.

3.1.3 Queries and Answers

Queries in KALM^{RA} must be in factual English and end with a question mark. KALM^{RA} translates both *Wh*-variables and explicitly typed variables into the corresponding DLV variables. Example 7 shows how KALM^{RA} represents a query with variables.

Example 7 The query "*Who undergoes $therapy?*" has the following ULR:

```
frame("Undergoing",[rl("Patient",Who),rl("Therapy",Therapy)])?
```

KALM^{RA} then invokes the DLV reasoner to compute query answers. DLV has two inference modes: *brave reasoning* and *cautious reasoning*. In brave reasoning, a query returns answers that are true in *at least one* model of the program and cautious reasoning returns the answers that are true in *all* models. Users are free to choose either mode.

Example 8 For instance, if the underlying information contains only this single fact

```
{frame("Undergoing",[rl("Patient","Mary"),rl("Therapy",mental)]).}
```

then there is only one model and both modes return the same result:

```
{Who="Mary",Therapy=mental}
```

In case of "*Mary or Bob undergoes a mental therapy*", two models are are computed:

```
{frame("Undergoing",[rl("Patient","Mary"),rl("Therapy",antimicrobial)]).}
{frame("Undergoing",[rl("Patient","Bob"),rl("Therapy",antimicrobial)]).}
```

In the cautious mode there would be no answers while the brave mode yields two:

```
{Who="Mary",Therapy=mental}
{Who="Bob",Therapy=mental}
```

3.2 Authoring and Reasoning with Actions

Time-independent facts and rules discussed earlier are knowledge that persists over time. In contrast, actions are momentary occurrences of events that change the underlying knowledge, so actions are associated with timestamps. Dealing with actions and their effects, also known as *fluents*, requires an understanding of the passage of time. KALM^{RA} allows users to state actions using factual English and then formalizes actions as temporal database facts using SEC discussed Section 2.3. The following discussion of authoring and reasoning with actions will be in the SEC framework.

Reasoning based on SEC requires the knowledge of fluent initiation and termination. This information is part of the commonsense and domain knowledge supplied by knowledge engineers and domain experts via high-level fluent initiation and termination statements (Definition 2) and KALM^{RA} translates them into facts and rules that involve the predicates `initiates/2` and `terminates/2` used by Event Calculus. Knowledge engineers supply the commonsense part of these statements and domain experts supply the domain-specific part.

Definition 2 A *fluent initiation statement* in KALM^{RA} has the form "*A/F_{obs} initiates F_{init}*" and a *fluent termination statement* in KALM^{RA} has the form "*A/F_{obs} terminates F_{term}*", where

1. action *A*, observed fluent F_{obs}, initiated fluent F_{init}, and terminated fluent F_{term} are factual sentences without conjunction or disjunction;
2. variables in F_{init} use explicitly typed syntax and must appear in *A* (or in F_{obs} when a fluent is observed) to avoid unbound variables in initiated fluents;
3. variables that refer to the same thing must have the same name.

Example 9 shows how KALM^{RA} represents fluent initiation and termination.

Example 9 The commonsense initiation statement "*$person travels to $place initiates $person is located in $place*" would be created by a knowledge engineer and translated by KALM^{RA} as the following rule:

```
initiates(frame("Travel",[rl("Person",Person),rl("Place",Place)]),
          frame("Located",[rl("Entity",Person),rl("Location",Place)])):-
    person(Person), place(Place).
```

Here `person/1` and `place/1` are used to guarantee rule safety. Since, any object can be in one place only at any given time, we have a commonsense termination statement "*$person travels to $place1 terminates $person is located in $place2.*" This statement would also be created by knowledge engineers and translated by KALM^{RA} as follows:

```
terminates(frame("Travel",[rl("Person",Person),rl("Place",Place)]),
           frame("Located",[rl("Entity",Person),rl("Location",Place2)])):-
    person(Person), place(Place), entity(Person), location(Place2), Place!=Place2.
```

KALM^{RA} also enhances rules by incorporating temporal information, allowing the inference of new knowledge under the SEC framework. The process begins by requiring users to specify their domain knowledge on fluents in the form of rules described in Definition 1. Then KALM^{RA} translates these rules into ULRs, with each premise and conclusion linked to a timestamp via the `holdsAt/2` predicate. We call these rules *time-related* because they enable reasoning with fluents containing temporal information. Here is an example of a time-related rule.

```
holdsAt(ULRC1,T) v ... v holdsAt(ULRCm,T) :-
    holdsAt(ULRP1,T), ..., holdsAt(ULRPn,T).
```

where all `holdsAt/2` terms share the same timestamp `T`, since the disjunction of conclusion ULRs `ULRC1`, ..., `ULRCm` holds immediately if all premise ULRs `ULRP1`, ..., `ULRPn` hold simultaneously at `T`.

KALM^{RA} incorporates temporal information in queries also using `holdsAt/2`. In this representation, the second argument of `holdsAt/2` is set to the highest value in the temporal domain extracted from the narrative. For Example 1, a time-related query can be represented as `holdsAt(ULRQ,3)?`, where `ULRQ` is the ULR of the query and `3` is the timestamp that exceeds all the explicitly given timestamps.

4 KALM^{RA} valuation

In this section, we assess the effectiveness of KALM^{RA}-based knowledge authoring using two test suites, the clinical UTI guidelines (Committee on Quality Improvement 1999) and the bAbI Tasks (Weston et al. 2015).

4.1 Evaluation of Rule Authoring

The UTI guidelines (Committee on Quality Improvement 1999) is a set of therapeutic recommendations for the initial Urinary Tract Infection (UTI) in febrile infants and young children. The original version in English was rewritten into the ACE CNL (Shiffman et al. 2009) for the assessment of ACE's expressiveness. We rewrite the original English version into factual English, as shown in Appendix A. This new version has a significant number of rules with disjunctive heads, as is common in the real-world medical domain.

The experimental results show that KALM^{RA} is able to convert the UTI guidelines document into ULRs with 100% accuracy.

4.2 Evaluation of Authoring of Actions

The 20 bAbI tasks (Weston et al. 2015) were designed to evaluate a system's capacity for natural language understanding, especially when it comes to actions. They cover a range of aspects,

such as moving objects (tasks 1-6), positional reasoning (task 17), and path finding (task 19). Each task provides a set of training and test data, where each data point consists of a textual narrative, a question about the narrative, and the correct answer. Fig. B 1 in Appendix B presents 20 data points from 20 bAbI tasks respectively. We used the test data for evaluation. Each task in the test data has 1,000 data points.

Table 1. Result Comparisons

	STM	ILA	KALM^{RA}		
TASK	Acc.	Acc.	#I&T	#Rules	Acc.
1 Single Supporting Fact	100	100	2	0	100
2 Two Supporting Facts	99.79	100	4	1	100
3 Three Supporting Facts	97.87	100	4	1	100
4 Two Argument Relations	100	100	4	0	100
5 Three Argument Relations	99.43	100	4	0	100
6 Yes/No Questions	100	100	2	0	100
7 Counting	99.19	100	4	0	100
8 Lists/Sets	99.88	100	4	0	100
9 Simple Negation	100	100	4	0	100
10 Indefinite Knowledge	99.97	100	2	0	100
11 Basic Coreference	99.99	100	2	0	100
12 Conjunction	99.96	100	2	0	100
13 Compound Coreference	99.99	100	2	0	93.1
14 Time Reasoning	99.84	100	2	0	100
15 Basic Deduction	100	100	0	1	100
16 Basic Induction	99.71	93.6	2	1	93.6
17 Positional Reasoning	98.82	100	8	20	100
18 Size Reasoning	99.73	100	0	1	100
19 Path Finding	97.94	100	12	4	100
20 Agent's Motivations	100	100	5	6	100
Average	99.61	99.68	3.45	1.75	99.34

The comparison systems in this evaluation include a state-of-the-art neural model on bAbI Tasks, STM (Le et al. 2020); an approach based on inductive learning and logic programming (Mitra and Baral 2016) that we call LPA here; and a recent sensation, ChatGPT. The comparison with STM and ILA results are displayed in Table 1, where "#I&T" denotes the number of user-given initiation and termination statements (Definition 2) used to specify each particular task in KALM^{RA}. The table shows that KALM^{RA} achieves accuracy comparable to STM and ILA. ChatGPT has shown impressive ability to give correct answers for some manually-entered bAbI tasks even though (we assume) it was not trained on that data set. However, it quickly became clear that it has no robust semantic model behind its impressive performance and it makes many mistakes on bAbI Tasks. The recent (Jan 30, 2023) update of ChatGPT fixed some of the cases, while still not being able to handle slight perturbations of those cases. Three such errors are shown in Table 2, which highlights the need for authoring approaches, like KALM^{RA}, which are based on robust semantic models.

Table 2. ChatGPT Error Cases

Task 2 **2 Supporting Facts**	**Task 17** **Positional Reasoning**	**Task 19** **Path Finding**
Mary went to the kitchen. Mary got the apple. Mary got the ball. Mary got the book. Mary went to the bedroom. Mary went to the garden. Mary dropped the book.	The red square is below the blue square. The red square is left of the pink rectangle.	The garden is west of the hallway. The kitchen is west of the garden. The garden is north of the bathroom. The bedroom is east of the bathroom. The hallway is west of the office.
Q: Where is the apple?	Q: Is the blue square below the pink rectangle?	Q: How do you go from the bathroom to the hallway?
ChatGPT: ... not specified Correct: garden	ChatGPT: ... not specified Correct: no	ChatGPT: ... east..., ... south... Correct: east, north

As to KALM^{RA}, it does not achieve 100% correctness on Tasks 13 (Compound Coreference) and 16 (Basic Induction). In Task 13, the quality of KALM^{RA}'s coreference resolution is entirely dependent on the output of neuralcoref, the coreference resolver we used. As this technology improves, so will KALM^{RA}. Task 16 requires the use of the induction principles adopted by bAbI tasks, some of which are questionable. For instance, in Case 2 of Table 3, the color is determined by the maximum frequency of that type, whereas in Case 3, the latest evidence determines the color. Both of these principles are too simplistic and, worse, contradict each other.

Table 3. KALM^{RA} Error Cases

Case1	**Case 2**	**Case 3**
Task 13 Compound Coreferences	**Task 16 Basic Induction**	
Mary and Sandra went back to the bedroom. Then they moved to the kitchen. Sandra and Daniel went back to the bathroom. Then they went to the office.	Brian is a swan. Greg is a swan. Julius is a swan. Greg is gray. Julius is gray. Bernhard is a lion. Lily is a swan. Berhnard is green. Brian is white.	Berhnard is a rhino. Brian is a rhino. Bernhard is white. Brian is white. Lily is a lion. Lily is yellow. Greg is a rhino. Greg is green. Julius is a rhino.
Q: Where is Daniel?	Q: What color is Lily?	Q: What color is Julius?
KALM^{RA}: bathroom bAbI Correct: office	KALM^{RA}: gray, white bAbI Correct: gray	KALM^{RA}: green, white bAbI Correct: green

5 Conclusion and Future Work

The KALM^{FL} system (Wang et al. 2022) was designed to address the limitations of KALM (Gao et al. 2018a) in terms of expressive power and the costs of the actual authoring of knowledge by human domain experts. KALM did not support authoring of rules and actions, and it required abiding a hard-to-learn grammar of the ACE CNL. In this paper, we introduced KALM^{RA}, an NLP system that extends KALM^{FL} to authoring of rules and actions by tackling a slew of problems. The evaluation results show that KALM^{RA} achieves 100% accuracy on authoring rules, and 99.34% accuracy on authoring and reasoning with actions, demonstrating the effectiveness of KALM^{RA} at capturing knowledge via facts, actions, rules, and queries. In future work, we plan to add non-monotonic extensions of factual English to support defeasible reasoning (Wan et al. 2009), a more natural way of human reasoning in real life, where conclusions are derived from default assumptions, but some conclusions may be retracted when the addition of new knowledge violates these assumptions.

References

BAKER, C. F., FILLMORE, C. J., AND LOWE, J. B. 1998. The berkeley framenet project. In *36th Annual Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics*. COLING, 86–90.

COMMITTEE ON QUALITY IMPROVEMENT, S. O. U. T. I. 1999. Practice parameter: the diagnosis, treatment, and evaluation of the initial urinary tract infection in febrile infants and young children. *Pediatrics 103,* 4, 843–852.

FILLMORE, C. J. ET AL. 2006. Frame semantics. *Cognitive linguistics: Basic readings 34*, 373–400.

FUCHS, N. E., KALJURAND, K., AND KUHN, T. 2008. Attempto controlled english for knowledge representation. In *Reasoning Web*. Springer, 104–124.

FUCHS, N. E. AND SCHWITTER, R. 1996. Attempto controlled english (ace). *arXiv preprint cmp-lg/9603003*.

GAO, T., FODOR, P., AND KIFER, M. 2018a. High accuracy question answering via hybrid controlled natural language. In *2018 IEEE/WIC/ACM International Conference on Web Intelligence (WI)*. IEEE, 17–24.

GAO, T., FODOR, P., AND KIFER, M. 2018b. Knowledge authoring for rule-based reasoning. In *OTM Confederated International Conferences" On the Move to Meaningful Internet Systems"*. Springer, 461–480.

GEBSER, M., KAMINSKI, R., KAUFMANN, B., AND SCHAUB, T. 2019. Multi-shot asp solving with clingo. *Theory and Practice of Logic Programming 19,* 1, 27–82.

GELFOND, M. AND LIFSCHITZ, V. 1988. The stable model semantics for logic programming. In *ICLP/SLP*. Vol. 88. Cambridge, MA, 1070–1080.

GELFOND, M. AND LIFSCHITZ, V. 1991. Classical negation in logic programs and disjunctive databases. *New generation computing 9*, 365–385.

KIFER, M. AND LAUSEN, G. 1989. F-logic: a higher-order language for reasoning about objects, inheritance, and scheme. In *Proceedings of the 1989 ACM SIGMOD international conference on Management of data*. 134–146.

KIFER, M., LAUSEN, G., AND WU, J. 1995. Logical foundations of object-oriented and frame-based languages. *42*, 741–843.

KOWALSKI, R. AND SERGOT, M. 1989. A logic-based calculus of events. In *Foundations of knowledge base management*. Springer, 23–55.

LE, H., TRAN, T., AND VENKATESH, S. 2020. Self-attentive associative memory. In *International Conference on Machine Learning*. PMLR, 5682–5691.

LEONE, N., PFEIFER, G., FABER, W., EITER, T., GOTTLOB, G., PERRI, S., AND SCARCELLO, F. 2006. The DLV system for knowledge representation and reasoning. *ACM Transactions on Computational Logic (TOCL) 7,* 3, 499–562.

MITRA, A. AND BARAL, C. 2016. Addressing a question answering challenge by combining statistical methods with inductive rule learning and reasoning. In *AAAI*.

QI, P., ZHANG, Y., ZHANG, Y., BOLTON, J., AND MANNING, C. D. 2020. Stanza: A python natural language processing toolkit for many human languages. *arXiv preprint arXiv:2003.07082*.

RINGGAARD, M., GUPTA, R., AND PEREIRA, F. C. 2017. Sling: A framework for frame semantic parsing. *arXiv preprint arXiv:1710.07032*.

SADRI, F. AND KOWALSKI, R. A. 1995. Variants of the event calculus. In *ICLP*. 67–81.

SCHWITTER, R. 2002. English as a formal specification language. In *Proceedings. 13th International Workshop on Database and Expert Systems Applications*. IEEE, 228–232.

SHIFFMAN, R. N., MICHEL, G., KRAUTHAMMER, M., FUCHS, N. E., KALJURAND, K., AND KUHN, T. 2009. Writing clinical practice guidelines in controlled natural language. In *International Workshop on Controlled Natural Language*. Springer, 265–280.

SWAYAMDIPTA, S., THOMSON, S., DYER, C., AND SMITH, N. A. 2017. Frame-semantic parsing with softmax-margin segmental rnns and a syntactic scaffold. *arXiv preprint arXiv:1706.09528*.

SWIFT, T. AND WARREN, D. S. 2012. XSB: Extending prolog with tabled logic programming. *Theory and Practice of Logic Programming 12,* 1-2, 157–187.

WAN, H., GROSOF, B. N., KIFER, M., FODOR, P., AND LIANG, S. 2009. Logic programming with defaults and argumentation theories. In *Logic Programming, 25th International Conference, ICLP 2009, Pasadena, CA, USA, July 14-17, 2009. Proceedings*, P. M. Hill and D. S. Warren, Eds. Lecture Notes in Computer Science, vol. 5649. Springer, 432–448.

WANG, Y., BORCA-TASCIUC, G., GOEL, N., FODOR, P., AND KIFER, M. 2022. Knowledge authoring with factual english. *arXiv preprint arXiv:2208.03094*.

WESTON, J., BORDES, A., CHOPRA, S., RUSH, A. M., VAN MERRIËNBOER, B., JOULIN, A., AND MIKOLOV, T. 2015. Towards AI-complete question answering: A set of prerequisite toy tasks. *arXiv preprint arXiv:1502.05698*.