Journal of Intelligent Information Systems (2022) 59:121–148
https://doi.org/10.1007/s10844-021-00682-5

A relational tsetlin machine with applications to natural language understanding

Rupsa Saha[1] · **Ole-Christoffer Granmo**[1] · **Vladimir I. Zadorozhny**[1,2] · **Morten Goodwin**[1]

Received: 19 March 2021 / Revised: 11 October 2021 / Accepted: 11 October 2021 /
Published online: 1 January 2022

Abstract
Tsetlin machines (TMs) are a pattern recognition approach that uses finite state machines for learning and propositional logic to represent patterns. In addition to being natively interpretable, they have provided competitive accuracy for various tasks. In this paper, we increase the computing power of TMs by proposing a first-order logic-based framework with Herbrand semantics. The resulting TM is *relational* and can take advantage of logical structures appearing in natural language, to learn rules that represent how actions and consequences are related in the real world. The outcome is a logic program of Horn clauses, bringing in a structured view of unstructured data. In closed-domain question-answering, the first-order representation produces 10× more compact KBs, along with an increase in answering accuracy from 94.83% to 99.48%. The approach is further robust towards erroneous, missing, and superfluous information, distilling the aspects of a text that are important for real-world understanding

Keywords Tsetlin machine · Natural language processing · Logic programming · Knowledge representation · Question answering

1 Introduction

Using Artificial Intelligences (AI) to answer natural language questions has long been an active research area, considered as an essential aspect in machines ultimately achieving human-level world understanding. Large-scale structured knowledge bases (KBs), such as Freebase (Bollacker et al., 2008), have been a driving force behind successes in this field. The KBs encompass massive ever-growing amounts of information, which enable easier handling of Open-Domain Question-Answering (QA) (Prager, 2006) by organizing a large

✉ Rupsa Saha
rupsa.saha@uia.no

[1] Centre for AI Research, Department of IKT, University of Agder, Agder, Norway

[2] School of Computing and Information, University of Pittsburgh, Pittsburgh, PA, USA

variety of answers in a structured format. The difficulty arises in successfully interpreting natural language by artificially intelligent agents, both to build the KBs from natural language text resources and to interpret the questions asked.

Generalization beyond the information stored in a KB further complicates the QA problem. Human-level world understanding requires abstracting from specific examples to build more general concepts and rules. When the information stored in the KB is error-free and consistent, generalization becomes a standard inductive reasoning problem. However, abstracting world-knowledge entails dealing with uncertainty, vagueness, exceptions, errors, and conflicting information. This is particularly the case when relying on Artificial Intelligences (AI) approaches to extract and structure information, which is notoriously error-prone.

This paper addresses the above QA challenges by proposing a Relational Tsetlin machine (TM) that builds non-recursive first-order *Horn clauses* from specific examples, distilling general concepts and rules.

Tsetlin Machines (Granmo, 2018) are a pattern recognition approach to constructing human-understandable patterns from given data, founded on propositional logic. While the idea of Tsetlin automaton (TA) (Tsetlin, 1961) have been around since 1960s, using them in pattern recognition is relatively new. TMs have successfully addressed several machine learning tasks, including natural language understanding (Yadav et al., 2021a, b; Bhattarai et al., 2021; Saha et al., 2020; Berge et al., 2019), image analysis (Granmo et al., 2019), classification (Abeyrathna et al., 2021), regression (Abeyrathna et al., 2019), and speech understanding (Lei et al., 2021). The propositional clauses constructed by a Tsetlin machine (TM) have high discriminative power and constitute a global description of the task learnt (Blakely & Granmo, 2020; Saha et al., 2020). Apart from maintaining accuracy comparable to state-of-the-art machine learning techniques, the method also has provided a smaller memory footprint and faster inference than more traditional neural network-based models (Wheeldon et al., 2020; Lei et al., 2020; Abeyrathna et al., 2021; Lei et al., 2021; Phoulady et al., 2019). Furthermore, Shafik et al. (2020) shows that TMs can be fault-tolerant, able to mask stuck-at faults. However, although TMs can express any propositional formula by using disjunctive normal form, first-order logic is required to obtain the computing power equivalent to a universal Turing machine. In this paper, we take the first steps towards increasing the computing power of TM by introducing a *first order* TM framework with Herbrand semantics, referred to as the *Relational* TM. Accordingly, we will in the following denote the original approach as *Propositional TM*.

Closed-Domain Question-Answering: As proof-of-concept, we apply our proposed Relational TM to so-called Closed-Domain QA. Closed-Domain QA assumes a text (single or multiple sentences) followed by a question which refers to some aspect of the preceding text. Accordingly, the amount of information that must be navigated is less than for open question-answering. Yet, answering closed-domain questions poses a significant natural language understanding challenge.

Consider the following example of information, taken from Rajpurkar et al. (2016): "The Black Death is thought to have originated in the arid plains of Central Asia, where it then travelled along the Silk Road, reaching Crimea by 1343. From there, it was most likely carried by Oriental rat fleas living on the black rats that were regular passengers on merchant ships."One can then have questions such as "Where did the black death originate?" or "How did the black death make it to the Mediterranean and Europe?". These questions can be

answered completely with just the information provided, hence it is an example of closed-domain question answering. However, mapping the question to the answer requires not only natural language processing, but also a fair bit of language understanding.

Here is a much simpler example: "Bob went to the garden. Sue went to the cafe. Bob walked to the office." This information forms the basis for questions like "Where is Bob?" or "Where is Sue?". Taking it a step further, given the previous information and questions, one can envision a model that learns to answer similar questions based on similar information, even though the model has never seen the specifics of the information before (i.e., the names and the locations).

With QA being such an essential area of Natural Language Understanding, there has been a lot of different approaches proposed. Common methods to QA include the following:

- Linguistic techniques, such as tokenization, POS tagging and parsing that transform questions into a precise query that merely extracts the respective response from a structured database;
- Statistical techniques such as Support Vector Machines, Bayesian Classifiers, and maximum entropy models, trained on large amount of data, specially for open QA;
- Pattern matching using surface text patterns with templates for response generation.

Many methods use a hybrid approach encompassing more than one of these approaches for increased accuracy. Most QA systems suffer from a lack of generality, and are tuned for performance in restricted use cases. Lack of available explainabilty also hinders researchers' quest to identify pain points and possible major improvements (Soares & Parreiras, 2020; Pundge et al., 2016).

Paper Contributions: Our main contributions in this paper are as follows:

- We introduce a *Relational* TM, as opposed to a propositional one, founded on non-recursive Horn clauses and capable of processing relations, variables and constants.
- We propose an accompanying relational framework for efficient representation and processing of the QA problem.
- We provide empirical evidence uncovering that the Relational TM produces at least one order of magnitude more compact KBs than the Propositional TM. At the same time, answering accuracy increases from 94.83% to 99.48% because of more general rules.
- We provide a model-theoretical interpretation for the proposed framework.

Overall, our Relational TM unifies knowledge representation, learning, and reasoning in a single framework.

Paper Organization: The paper is organized as follows. In Section 2, we present related work on Question Answering. Section 3 focuses on the background of the Propositional TM and the details of the new Relational TM. In Sections 4 and 5, we describe how we employ Relational TM in QA and related experiments.

2 Background and related work

The problem of QA is related to numerous aspects of Knowledge Engineering and Data Management.

Knowledge engineering deals with constructing and maintaining knowledge bases to store knowledge of the real world in various domains. Automated reasoning techniques use this knowledge to solve problems in domains that ordinarily require human logical reasoning. Therefore, the two key issues in knowledge engineering are how to construct and maintain knowledge bases, and how to derive new knowledge from existing knowledge effectively and efficiently. Automated reasoning is concerned with the building of computing systems that automate this process. Although the overall goal is to automate different forms of reasoning, the term has largely been identified with valid deductive reasoning as conducted in logical systems. This is done by combining known (yet possibly incomplete) information with background knowledge and making inferences regarding unknown or uncertain information.

Typically, such a system consists of subsystems like knowledge acquisition system, the knowledge base itself, inference engine, explanation subsystem and user interface. The knowledge model has to represent the relations between multiple components in a symbolic, machine understandable form, and the inference engine has to manipulate those symbols to be capable of reasoning. The "way to reason" can range from earlier versions that were simple rule-based systems to more complex and recent approaches based on machine learning, especially on Deep Learning. Recently, transformer based systems have significantly improved question-answering performance in general, even more so when they can be supplied with a very large amount of training data (Lu et al., 2020; Lewis et al., 2019; Wolf et al., 2019; Jung & Kim, 2020; Srinivasan et al., 2021). Typically, rule-based systems suffered from lack of generality, and the need for human experts to create rules in the first place. On the other hand most machine learning based approaches have the disadvantage of not being able to justify decisions taken by them in a human understandable form Ludwig (2010) and Cyras et al. (2020).

While databases have long been a mechanism of choice for storing information, they only had inbuilt capability to identify relations between various components, and did not have the ability to support reasoning based on such relations. Efforts to combine formal logic programming with relational databases led to the advent of deductive databases. In fact, the field of QA is said to have arisen from the initial goal of performing deductive reasoning on a set of given facts (Green, 1969). In deductive databases, the semantics of the information are represented in terms of mathematical logic. Queries to deductive databases also follow the same logical formulation (Gallaire et al., 1989). One such example is ConceptBase (Jarke et al., 1995), which used the Prolog-inspired language O-Telos for logical knowledge representation and querying using deductive object-oriented database framework.

With the rise of the internet, there came a need for unification of information on the web. The Semantic Web (SW) proposed by W3C is one of the approaches that bridges the gap between the Knowledge Representation and the Web Technology communities. However, reasoning and consistency checking is still not very well developed, despite the underlying formalism that accompanies the semantic web. One way of introducing reasoning is via descriptive logic. It involves concepts (unary predicates) and roles (binary predicates) and the idea is that implicitly captured knowledge can be inferred from the given descriptions of concepts and roles (Dong et al., 2003; Turhan, 2011).

One of the major learning exercises is carried out by the NELL mechanism proposed by Mitchell et al. (2018), which aims to learn many semantic categories from primarily unlabeled data. At present, NELL uses simple frame-based knowledge representation, augmented by the PRA reasoning system. The reasoning system performs tractable, but limited

types of reasoning based on restricted Horn clauses. NELL's capabilities is already limited in part by its lack of more powerful reasoning components; for example, it currently lacks methods for representing and reasoning about time and space. Hence, core Artificial Intelligence (AI) problems of representation and tractable reasoning are also core research problems for never-ending learning agents.

While other approaches such as neural networks are considered to provide attribute-based learning, Inductive Logic Programming (ILP) is an attempt to overcome their limitations by moving the learning away from the attributes themselves and more towards the level of first-order predicate logic. ILP builds upon the theoretical framework of logic programming and looks to construct a predicate logic given background knowledge, positive examples and negative examples. One of the main advantages of ILP over attribute-based learning is ILP's generality of representation for background knowledge. This enables the user to provide, in a more natural way, domain-specific background knowledge to be used in learning. The use of background knowledge enables the user both to develop a suitable problem representation and to introduce problem-specific constraints into the learning process. Over the years, ILP has evolved from depending on hand-crafted background knowledge only, to employing different technologies in order to learn the background knowledge as part of the process. In contrast to typical machine learning, which uses feature vectors, ILP requires the knowledge to be in terms of facts and rules governing those facts. Predicates can either be supplied or deducted, and one of the advantages of this method is that newer information can be added easily, while previously learnt information can be maintained as required (Cropper et al., 2020). Probabilistic inductive logic programming is an extension of ILP, where logic rules, as learnt from the data, are further enhanced by learning probabilities associated with such rules (Bratko & Muggleton, 1995; Nickles & Mileo, 2014; De Raedt & Kersting, 2008; Kramer et al., 2001).

To sum up, none of the above approaches can be efficiently and systematically applied to the QA problem, especially in uncertain and noisy environments. In this paper we propose a novel approach to tackle this problem. Our approach is based on relational representation of QA, and using a novel Relational TM technique for answering questions. We elaborate on the proposed method in the next two sections.

3 Building a relational tsetlin machine

3.1 Tsetlin machine foundation

A Tsetlin Automaton (Tsetlin Automaton (TA)) is a deterministic automaton that learns the optimal action among the set of actions offered by an environment. It performs the action associated with its current state, which triggers a reward or penalty based on the ground truth. The state is updated accordingly, so that the Tsetlin automaton (TA) progressively shifts focus towards the optimal action (Tsetlin, 1961). A 2-action Tsetlin automaton (TA), as shown in Fig. 1a, has $2N$ states, such that, a final state between 1 and N indicates Action 1, while final state between $N + 1$ and $2N$ indicates Action 2. A reward reinforces a particular action by changing state away from the central state, i.e. moving "deeper" towards Action 1 or 2, as the case maybe. The opposite happens in the case of a penalty, where the resultant state after an iteration is closer to the central states ($N/N + 1$). A TM consists of a collection of such TAs, which together create complex propositional formulas using conjunctive clauses.

3.1.1 Classification

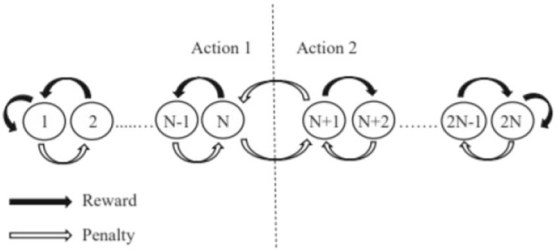
A TM takes a vector $X = (x_1, \ldots, x_f)$ of propositional features as input, to be classified into one of two classes, $y = 0$ or $y = 1$. Together with their negated counterparts, $\bar{x}_k = \neg x_k = 1 - x_k$, the features form a literal set $L = \{x_1, \ldots, x_f, \bar{x}_1, \ldots, \bar{x}_f\}$. We refer to this "regular" TM as a Propositional TM, due to the input it works with and the output it produces.

A TM pattern is formulated as a conjunctive clause C_j, formed by ANDing a subset $L_j \subseteq L$ of the literal set:

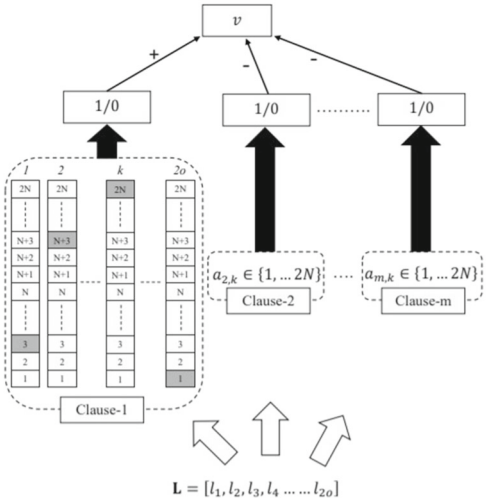
$$C_j(X) = \bigwedge_{l_k \in L_j} l_k = \prod_{l_k \in L_j} l_k. \tag{1}$$

E.g., the clause $C_j(X) = x_1 \wedge x_2 = x_1 x_2$ consists of the literals $L_j = \{x_1, x_2\}$ and outputs 1 iff $x_1 = x_2 = 1$.

Each clause has access to $2 \times f$ Tsetlin automaton (TAs), one per literal k. The bidirectional action of each TA decides which of the literals form a part of a clause. The TA (TA) states from 1 to N map to the exclude action, which means that the corresponding literal



(a) Transition graph of a two-action Tsetlin Automaton



(b) Tsetlin Machine Clause structure

Fig. 1 Clause Formation in a Tsetlin Machine

is excluded from the clause. For states from $N+1$ to $2N$, the decision becomes include, i.e., the literal is included instead. The states of all the TAs in all of the clauses are jointly stored in the matrix $A : A = (a_{j,k}) \epsilon \{1, \ldots, 2N\}^{m \times 2f}$ with j and k referring to the clause and the literal respectively. Thus $I_j^I = \{k | a_{j,k} > N, 1 \leq k \leq 2f\}$ encompasses the literal indices contained in I_j^I. Figure 1b shows the structure of the clauses and the voting scheme diagrammatically.

The number of clauses employed is a user set parameter m. Half of the m clauses are assigned positive polarity (C_j^+). The other half is assigned negative polarity (C_j^-). The clause outputs, in turn, are combined into a classification decision through summation:

$$v = \sum_{j=1}^{m/2} C_j^+(X) - \sum_{j=1}^{m/2} C_j^-(X). \tag{2}$$

In effect, the positive clauses vote for $y = 1$ and the negative for $y = 0$. Classification is performed based on a majority vote, using the unit step function: $\hat{y} = u(v) = 1$ **if** $v \geq 0$ **else** 0. The classifier $\hat{y} = u\,(x_1 \bar{x}_2 + \bar{x}_1 x_2 - x_1 x_2 - \bar{x}_1 \bar{x}_2)$, for instance, captures the XOR-relation.

3.1.2 Learning

Alg. 1 encompasses the entire learning procedure. We observe that learning is performed by a team of $2f$ Tsetlin automaton (TAs) per clause, one Tsetlin automaton (TA) per literal l_k (Alg. 1, Step 2). Each Tsetlin automaton (TA) has two actions – Include or Exclude – and decides whether to include its designated literal l_k in its clause.

TM learn on-line, processing one training example (X, y) at a time (Step 7). The Tsetlin automaton (TAs) first produce a new configuration of clauses (Step 8), $C_1^+, \ldots, C_{n/2}^-$, followed by calculating a voting sum v (Step 9).

Feedback are then handed out stochastically to each Tsetlin automaton (TAs) team. The difference ϵ between the clipped voting sum v^c and a user-set voting target T decides the probability of each Tsetlin automaton (TAs) team receiving feedback (Steps 12-20). Note that the voting sum is clipped to normalize the feedback probability. The voting target for $y = 1$ is T and for $y = 0$ it is $-T$. Observe that for any input X, the probability of reinforcing a clause gradually drops to zero as the voting sum approaches the user-set target. This ensures that clauses distribute themselves across the frequent patterns, rather than missing some and over-concentrating on others.

Clauses receive two types of feedback. Type I feedback produces frequent patterns, while Type II feedback increases the discrimination power of the patterns.

Type I feedback is given stochastically to clauses with positive polarity when $y = 1$ and to clauses with negative polarity when $y = 0$. Each clause, in turn, reinforces its Tsetlin automaton (TAs) based on: (1) its output $C_j(X)$; (2) the action of the Tsetlin automaton (TA) – Include or Exclude; and (3) the value of the literal l_k assigned to the Tsetlin automaton (TA). Two rules govern Type I feedback:

- *Include* is rewarded and *Exclude* is penalized with probability $\frac{s-1}{s}$ whenever $C_j(X) = 1$ **and** $l_k = 1$. This reinforcement is strong (triggered with high probability) and makes the clause remember and refine the pattern it recognizes in X.[1]
- *Include* is penalized and *Exclude* is rewarded with probability $\frac{1}{s}$ whenever $C_j(X) = 0$ **or** $l_k = 0$. This reinforcement is weak (triggered with low probability) and coarsens infrequent patterns, making them frequent.

[1] Note that the probability $\frac{s-1}{s}$ is replaced by 1 when boosting true positives.

Above, the user-configurable parameter s controls pattern frequency, i.e., a higher s produces less frequent patterns.

Type II feedback is given stochastically to clauses with positive polarity when $y = 0$ and to clauses with negative polarity when $y = 1$. It penalizes *Exclude* whenever $C_j(X) = 1$ **and** $l_k = 0$. Thus, this feedback produces literals for discriminating between $y = 0$ and $y = 1$, by making the clause evaluate to 0 when facing its competing class. Further details can be found in Granmo (2018).

Algorithm 1 Propositional TM.

input Tsetlin Machine TM, Example pool S, Training rounds e, Clauses n, Features f, Voting target T, Specificity s

procedure TRAIN(TM, S, e, n, f, T, s)
  for $j \leftarrow 1, \ldots, n/2$ **do**
    $TA_j^+ \leftarrow$ RandomlyInitializeClauseTATeam$(2f)$
    $TA_j^- \leftarrow$ RandomlyInitializeClauseTATeam$(2f)$
  end for
  for $i \leftarrow 1, \ldots, e$ **do**
    $(X_i, y_i) \leftarrow$ ObtainTrainingExample(S)
    $C_1^+, \ldots, C_{n/2}^- \leftarrow$ ComposeClauses$(TA_1^+, \ldots, TA_{n/2}^-)$
    $v_i \leftarrow \sum_{j=1}^{n/2} C_j^+(X_i) - \sum_{j=1}^{n/2} C_j^-(X_i)$ ▷ Vote sum
    $v_i^c \leftarrow$ **clip** $(v_i, -T, T)$ ▷ Clipped vote sum
    for $j \leftarrow 1, \ldots, n/2$ **do** ▷ Update Tsetlin automaton (TA) teams
      if $y_i = 1$ **then**
        $\epsilon \leftarrow T - v_i^c$ ▷ Voting error
        TypeIFeedback(X_i, TA_j^+, s) **if** rand() $\leq \frac{\epsilon}{2T}$
        TypeIIFeedback(X_i, TA_j^-) **if** rand() $\leq \frac{\epsilon}{2T}$
      else
        $\epsilon \leftarrow T + v_i^c$ ▷ Voting error
        TypeIIFeedback(X_i, TA_j^+) **if** rand() $\leq \frac{\epsilon}{2T}$
        TypeIFeedback(X_i, TA_j^-, s) **if** rand() $\leq \frac{\epsilon}{2T}$
      end if
    end for
  end for
end procedure

Algorithm 2 Relational TM.

input Convolutional Tsetlin Machine TM, Example pool S, Number of training rounds e

procedure TRAIN(TM, S, e)
  for $i \leftarrow 1, \ldots, e$ **do**
    $(\tilde{\mathcal{X}}, \tilde{\mathcal{Y}}) \leftarrow$ ObtainTrainingExample(S)
    $A' \leftarrow$ ObtainConstants$(\tilde{\mathcal{Y}})$
    $(\tilde{\mathcal{X}}', \tilde{\mathcal{Y}}') \leftarrow$ VariablesReplaceConstants$(\tilde{\mathcal{X}}, \tilde{\mathcal{Y}}, A')$
    $A'' \leftarrow$ ObtainConstants$(\tilde{\mathcal{X}}')$
    $Q \leftarrow$ GenerateVariablePermutations$(\tilde{\mathcal{X}}', A'')$
    UpdateConvolutionalTM(TM, $Q, \tilde{\mathcal{Y}}'$)
  end for
end procedure

3.2 Relational tsetlin machine

In this section, we introduce the Relational TM, which is a major contribution of this paper. It is designed to take advantage of logical structures appearing in natural language and process them in a way that leads to a compact, logic-based representation, which can ultimately reduce the gap between structured and unstructured data. While the Propositional TM operates on propositional input variables $X = (x_1 \ldots, x_f)$, building propositional conjunctive clauses, the Relational TM processes relations, variables and constants, building Horn clauses. Based on Fig. 2 and Alg. 2, we here describe how the Relational TM builds upon the original TM in three steps. First, we establish an approach for dealing with relations and constants. This is done by mapping the relations to propositional inputs, allowing the use of a standard TM. We then introduce Horn clauses with variables, showing how this representation detaches the TM from the constants, allowing for a more compact representation compared to only using propositional clauses. We finally introduce a novel convolution scheme that effectively manages multiple possible mappings from constants to variables. While the mechanism of convolution remains the same as in the original (Granmo et al., 2019), what we wish to attain from using it in a Relational TM context is completely different, as explained in the following.

3.2.1 Model-theoretical interpretation

The concept of Relational TM can be grounded in the model-theoretical interpretation of a logic program without functional symbols and with a finite *Herbrand model* (Lloyd, 1984; Kowalski, 2014). The ability to represent learning in the form of Horn clauses is extremely useful due to the fact that Horn clauses are both simple, as well as powerful enough to describe any logical formula (Kowalski, 2014).

Next, we define the *Herbrand model* of a logic program. A *Herbrand Base (HB)* is the set of all possible ground atoms, i.e., atomic formulas without variables, obtained using predicate names and constants in a logic program P. A *Herbrand Interpretation* is a subset I of the *Herbrand Base* ($I \subseteq HB$). To introduce the *Least Herbrand Model* we define the immediate consequence operator $TP : P(HB) \rightarrow P(HB)$, which for an *Herbrand Interpretation* I produces the interpretation that immediately follows from I by the rules (Horn clauses) in the program P:

$$TP(I) = \{A_0 \in HB \mid A_0 \leftarrow A_1, ..., A_n \in ground(P) \wedge \{A_1, ..., A_n\} \subseteq I\} \cup I.$$

The least fixed point *lfp(TP)* of the immediate consequence operator with respect to subset-inclusion is the *Least Herbrand Model (LHM)* of the program P. *LHM* identifies the semantics of the program P: it is the *Herbrand Interpretation* that contains those and only those atoms that follow from the program:

$$\forall A \in HB : P \models A \Leftrightarrow A \in \text{LHM}.$$

As an example, consider the following program P:

$$p(a).\ q(c).$$
$$q(X) \leftarrow p(X).$$

Its Herbrand base is

$$HB = \{p(a), p(c), q(a), q(c)\},$$

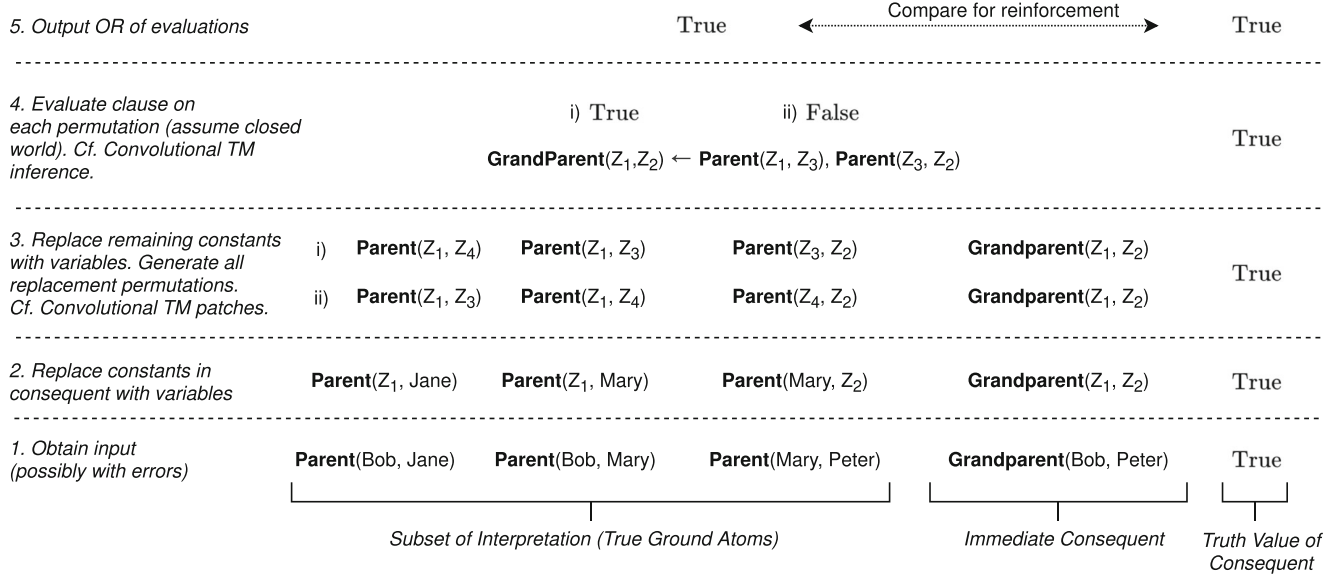


Fig. 2 Relational TM processing steps

and its Least Herbrand Model is:

$$\text{LHM} = \text{lfp}(TP) = \{p(a), q(a), q(c)\},$$

which is the set of atoms that follow from the program P.

3.2.2 Learning problem

Let $A = \{a_1, a_2, \ldots, a_q\}$ be a finite set of constants and let $R = \{r_1, r_2, \ldots, r_p\}$ be a finite set of relations of arity $w_u \geq 1, u \in \{1, 2, \ldots, p\}$, which forms the alphabet Σ. The Herbrand base

$$\begin{aligned} HB \quad &= \{r_1(a_1, a_2, \ldots, a_{w_1}), r_1(a_2, a_1, \ldots, a_{w_1}), \\ &\ldots, r_p(a_1, a_2, \ldots, a_{w_p}), r_p(a_2, a_1, \ldots, a_{w_p}), \ldots\} \end{aligned} \tag{3}$$

is then also finite, consisting of all the ground atoms that can be expressed using A and R.

We also have a logic program P, with program rules expressed as Horn clauses without recursion. Each Horn clause has the form:

$$B_0 \leftarrow B_1, B_2, \cdots, B_d. \tag{4}$$

Here, $B_l, l \in \{0, \ldots, d\}$, is an atom $r_u(Z_1, Z_2, \ldots, Z_{w_u})$ with variables $Z_1, Z_2, \ldots, Z_{w_u}$, or its negation $\neg r_u(Z_1, Z_2, \ldots, Z_{w_u})$. The arity of r_u is denoted by w_u.

Now, let $\mathcal{X}$ be a subset of the LHM of P, $\mathcal{X} \subseteq \text{lfp}(TP)$, and let $\mathcal{Y}$ be the subset of the LHM that follows from $\mathcal{X}$ due to the Horn clauses in P. Further assume that atoms are randomly removed and added to $\mathcal{X}$ and $\mathcal{Y}$ to produce a possibly noisy observation $(\tilde{\mathcal{X}}, \tilde{\mathcal{Y}})$, i.e., $\tilde{\mathcal{X}}$ and $\tilde{\mathcal{Y}}$ are not necessarily subsets of lfp(TP). *The learning problem is to predict the atoms in $\mathcal{Y}$ from the atoms in $\tilde{\mathcal{X}}$ by learning from a sequence of noisy observations $(\tilde{\mathcal{X}}, \tilde{\mathcal{Y}})$, thus uncovering the underlying program P.*

3.2.3 Relational tsetlin machine with constants

We base our Relational TM on mapping the learning problem to a Propositional TM pattern recognition problem. We consider Horn clauses without variables first. In brief, we map every atom in HB to a propositional input x_k, obtaining the propositional input vector $X = (x_1, \ldots, x_o)$ (cf. Section 3.1). That is, consider the w-arity relation $r_u \in R$, which takes w symbols from A as input. This relation can thus take q^w unique input combinations. As an example, with the constants $A = \{a_1, a_2\}$ and the binary relations $R = \{r_1, r_2\}$, we get 8 propositional inputs: $x_1^{1,1} \equiv r_1(a_1, a_1)$; $x_1^{1,2} \equiv r_1(a_1, a_2)$; $x_1^{2,1} \equiv r_1(a_2, a_1)$;$x_1^{2,2} \equiv r_1(a_2, a_2)$; $x_2^{1,1} \equiv r_2(a_1, a_1)$; $x_2^{1,2} \equiv r_2(a_1, a_2)$; $x_2^{2,1} \equiv r_2(a_2, a_1)$; and $x_2^{2,2} \equiv r_2(a_2, a_2)$. Correspondingly, we perform the same mapping to get the propositional output vector Y.

Finally, obtaining an input $(\tilde{\mathcal{X}}, \tilde{\mathcal{Y}})$, we set the propositional input x_k to true iff its corresponding atom is in $\tilde{\mathcal{X}}$, otherwise it is set to false. Similarly, we set the propositional output variable y_m to true iff its corresponding atom is in $\tilde{\mathcal{Y}}$, otherwise it is set to false.

Clearly, after this mapping, we get a Propositional TM pattern recognition problem that can be solved as described in Section 3.1 for a single propositional output y_m. This is illustrated as Step 1 in Fig. 2.

3.2.4 Detaching the relational TM from constants

The TM can potentially deal with thousands of propositional inputs. However, we now detach our Relational TM from the constants, introducing Horn clauses with variables. Our intent

is to provide a more compact representation of the program and to allow generalization beyond the data. Additionally, the detachment enables faster learning even with less data.

Let $\mathcal{Z} = \{Z_1, Z_2, \ldots, Z_z\}$ be z variables representing the constants appearing in an observation $(\tilde{\mathcal{X}}, \tilde{\mathcal{Y}})$. Here, z is the largest number of unique constants involved in any particular observation $(\tilde{\mathcal{X}}, \tilde{\mathcal{Y}})$, each requiring its own variable.

Seeking Horn clauses with variables instead of constants, we now only need to consider atoms over variable configurations (instead of over constant configurations). Again, we map the atoms to propositional inputs to construct a propositional TM learning problem. That is, each propositional input x_k represents a particular atom with a specific variable configuration: $x_k \equiv r_u(Z_{\alpha_1}, Z_{\alpha_2}, \ldots, Z_{\alpha_{w_u}})$, with w_u being the arity of r_u. Accordingly, the number of constants in A no longer affects the number of propositional inputs x_k needed to represent the problem. Instead, this is governed by the number of variables in $\mathcal{Z}$ (and, again, the number of relations in R). That is, the number of propositional inputs is bounded by $O(z^w)$, with w being the largest arity of the relations in R.

To detach the Relational TM from constants, we first replace the constants in $\tilde{\mathcal{Y}}$ with variables, from left to right. Accordingly, the corresponding constants in $\tilde{\mathcal{X}}$ is also replaced with the same variables (Step 2, Fig. 2). Finally, the constants now remaining in $\tilde{\mathcal{X}}$ is arbitrarily replaced with additional variables (Step 3, Fig. 2).

3.2.5 Relational tsetlin machine convolution over variable assignment permutations

Since there may be multiple ways of assigning variables to constants, the above approach may produce redundant rules. One may end up with equivalent rules whose only difference is syntactic, i.e., the same rules are expressed using different variable symbols. This is illustrated in Step 3 of Fig. 2, where variables can be assigned to constants in two ways. To avoid redundant rules, the Relational TM produces all possible permutations of variable assignments. To process the different permutations, we finally perform a convolution over the permutations in Step 4, employing a TM convolution operator (Granmo et al., 2019). The target value of the convolution is the truth value of the consequent (Step 5).

3.2.6 Walk-through of algorithm with example learning problem

Figure 2 contains an example of the process of detaching a Relational TM from constants. We use the parent-grandparent relationship as an example, employing the following Horn clause.

$$grandparent(Z_1, Z_2) \leftarrow parent(Z_1, Z_3), parent(Z_3, Z_2).$$

We replace the constants in each training example with variables, before learning the clauses. Thus the Relational TM never "sees" the constants, just the generic variables.

Assume the first training example is

$Input: \ parent(Bob, \ Mary) = 1;$

$Target\ output: \ child(Mary, \ Bob) = 1.$

Then Mary is replaced with Z_1 and Bob with Z_2 in the target output:

$Input: \ parent(Bob, \ Mary) = 1;$

$Target\ output: \ child(Z_1, \ Z_2) = 1.$

We perform the same exchange for the input, getting:

$Input: \ parent(Z_2, \ Z_1) = 1;$

$Target\ output: \ child(Z_1, \ Z_2) = 1.$

Here, "parent(Z_2, Z_1)" is treated as an input feature by the Relational TM. That is, "parent(Z_2, Z_1)" is seen as a single propositional variable that is either 0 or 1, and the name

of the variable is simply the string "parent(Z_2, Z_1)". The constants may be changing from example to example, so next time it may be Mary and Ann. However, they all end up as Z_1 or Z_2 after being replaced by the variables. After some time, the Relational TM would then learn the following clause:

$child(Z_1,\ Z_2) \leftarrow parent(Z_2, Z_1)$.

This is because the feature "parent(Z_2, Z_1)" predicts "child(Z_1, Z_2)" perfectly. Other candidate features like "parent(Z_1, Z_1)" or "parent(Z_2, Z_3)" are poor predictors of "child(Z_1, Z_2)" and will be excluded by the TM. Here, Z_3 is a free variable representing some other constant, different from Z_1 and Z_2.

Then the next training example comes along:

$Input:\ parent(Bob,\ Mary) = 1;$

$Target\ output:\ child(Jane,\ Bob) = 0.$

Again, we start with replacing the constants in the target output with variables:

$Input:\ parent(Bob,\ Mary) = 1;$

$Target\ output:\ child(Z_1,\ Z_2) = 0$

which is then completed for the input:

$Input:\ parent(Z_2,\ Mary) = 1;$

$Target\ output:\ child(Z_1,\ Z_2) = 0.$

The constant Mary was not in the target output, so we introduce a free variable Z_3 for representing Mary:

$Input:\ parent(Z_2,\ Z_3) = 1;$

$Target\ output:\ child(Z_1,\ Z_2) = 0.$

The currently learnt clause was:

$child(Z_1,\ Z_2) \leftarrow parent(Z_2, Z_1)$.

The feature "parent(Z_2, Z_1)" is not present in the input in the second training example, only "parent(Z_2, Z_3)". Assuming a closed world, we thus have
$parent(Z_2, Z_1) = 0$.

Accordingly, the learnt clause correctly outputs 0.

For some inputs, there can be many different ways variables can be assigned to constants (for the output, variables are always assigned in a fixed order, from Z_1 to Z_z). Returning to our grandparent example in Fig. 2, if we have:

$Input:\ parent(Bob, Mary);\ parent(Mary, Peter);\ parent(Bob, Jane)$

$Target\ output:\ grandparent(Bob, Peter).$

replacing Bob with Z_1 and Peter with Z_2, we get:

$Input:\ parent(Z_1, Mary);\ parent(Mary, Z_2);$
$parent(Z_1, Jane)$

$Target\ output:\ grandparent(Z_1, Z_2).$

Above, both Mary and Jane are candidates for being Z_3. One way to handle this ambiguity is to try both, and pursue those that make the clause evaluate correctly, which is exactly how the TM convolution operator works (Granmo et al., 2019).

Note that above, there is an implicit existential quantifier over Z_3. That is, $\forall Z_1, Z_2$ $(\exists Z_3(parent(Z_1, Z_3) \wedge parent(Z_3, Z_2) \rightarrow grandparent(Z_1, Z_2))$.

A practical view of how the TM learns in such a scenario is shown in Fig. 3. Continuing in the same vein as the previous examples, the Input Text in the figure is a set of statements, each followed by a question. The text is reduced to a set of relations, which act as the features for the TM to learn from. The figure illustrates how the TM learns relevant information (while disregarding the irrelevant), in order to successfully answer a new question

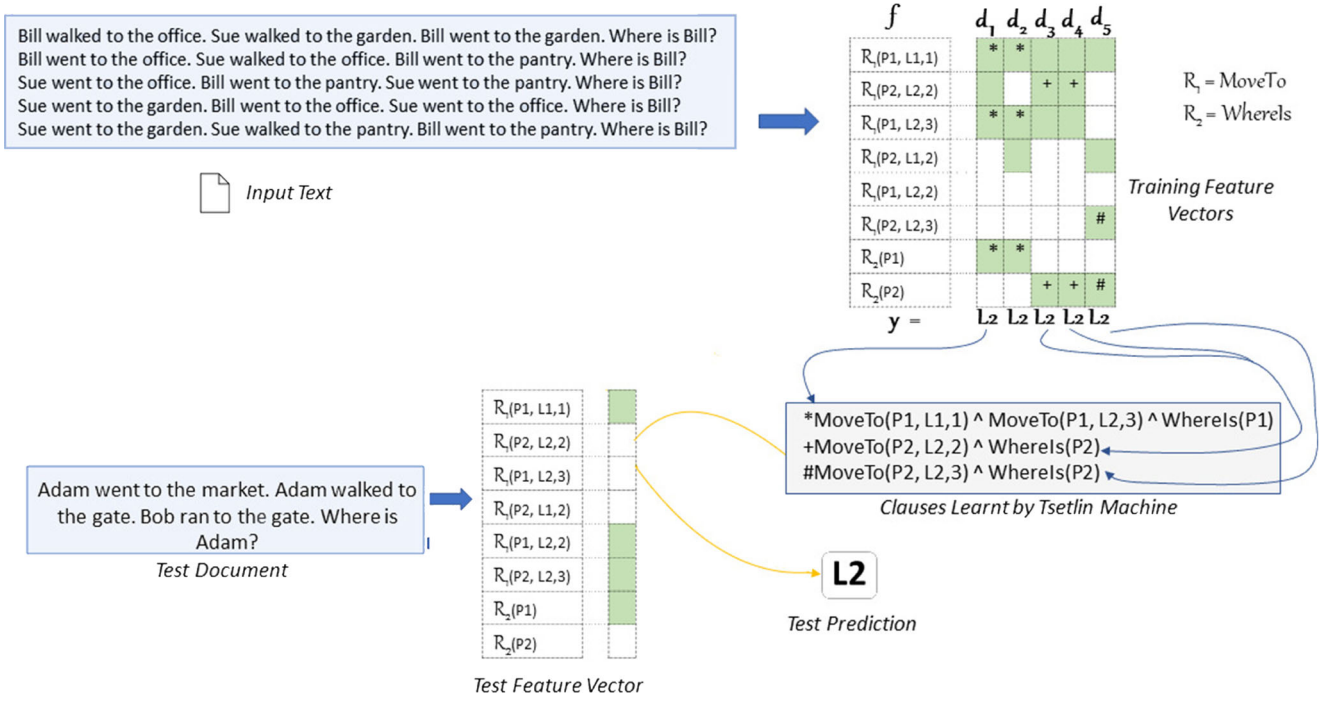


Fig. 3 The Relational TM in operation

(Test Document). The input text is converted into a features vector which indicates the presence or absence of relations (R_1, R_2) in the text, where R_1 and R_2 are respectively $MoveTo$ (in the statements) and $WhereIs$ (in the question) relations. For further simplification and compactness of representation, instead of using person and location names, those specific details are replaced by (P_1, P_2) and (L_1, L_2), respectively. In each sample, the person name that occurs first is termed P_1 throughout, the second unique name is termed P_2, and so on, and similarly for the locations. As seen in the figure, the TM reduces the feature-set representation of the input into a set of logical conditions or clauses, all of which together describe scenarios in which the answer is L_2 (or Location 2).

Remark 1 We now return to the implicit existential and universal quantifiers of the Horn clauses, exemplified in: $\forall Z_1, Z_2(\exists Z_3(parent(Z_1, Z_3) \wedge parent(Z_3, Z_2) \rightarrow grandparent(Z_1, Z_2))$. A main goal of the procedure in Fig. 2 is to correctly deal with the quantifiers "for all" and "exists". "For all" maps directly to the TM architecture because the TM is operating with conjunctive clauses and the goal is to make these evaluate to 1 (True) whenever the learning target is 1. "For all" quantifiers are taken care of in Step 3 of the relational learning procedure.

Remark 2 "Exists" is more difficult because it means that we are looking for a specific value for the variables in the scope of the quantifier that makes the expression evaluate to 1. This is handled in the Steps 4-6 in Fig. 2, by evaluating all alternative values (all permutations with multiple variables involved). Some values make the expression evaluate to 0 (False) and some make the expression become 1. If none makes the expression 1, the output of the clause is 0. Otherwise, the output is 1. Remarkably, this is exactly the behavior of the TM convolution operator defined in Granmo et al. (2019), so we have an existing learning procedure in place to deal with the "Exists" quantifier. (If there exists a patch in the image that makes the clause evaluate to 1, the clause evaluates to 1).

4 QA in a relational TM framework

In this section, we describe the general pipeline to reduce natural language text into a machine-understandable relational representation to facilitate question answering. Fig. 4 shows the pipeline diagrammatically, with a small example. Throughout this section, we make use of two toy examples in order to illustrate the steps. One of them is derived from a standard question answering dataset (Weston et al., 2015). The other is a simple handcrafted dataset, inspired by Kowalski (1979), that refers to parent-child relationships among a group of people. Both datasets consist of instances, where each instance is a set of two or more statements, followed by a query. The expected output for each instance is the answer to the query based on the statements. We would like to clarify here, that we have chosen to highlight the details of our method with toy examples in this work, due to ease of explainability. The ease arises primarily from the fact that the relations that form a part of these two sets are well-understood, simple to describe and have well-defined interactions between them. Further expansions on more advanced datasets, such as SQUAD/SQUADv2.0 (which involves question-answering from longer context texts), as well as on other real-world data is currently work in progress and beyond the scope of this particular paper.

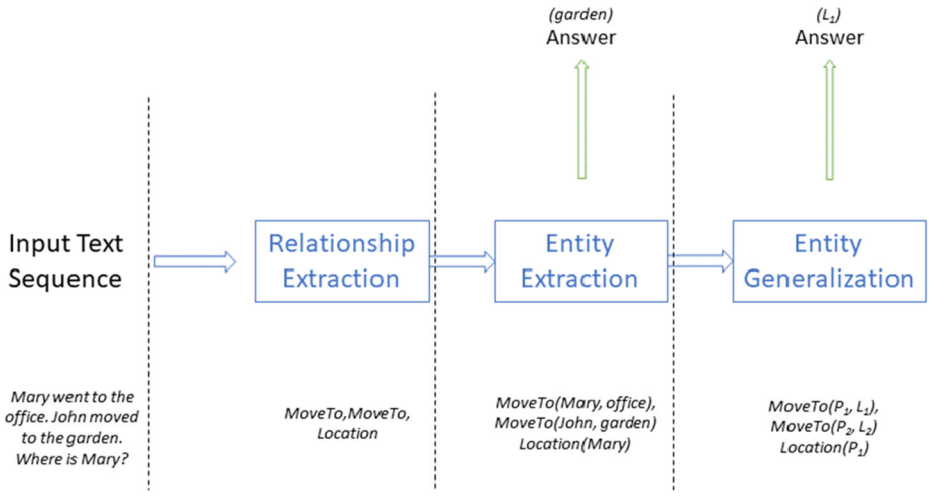


Fig. 4 General Pipeline

4.1 Relation extraction

As a first step, we need to extract the relation(s) present in the text. A relation here is a connection between two (or more) elements of a text. As discussed before, relations occur in natural language, and reducing a text to its constituent relations makes it more structured while ignoring superfluous linguistic elements, leading to easier understanding. We assume that our text consists of simple sentences, that is, each sentence contains only one relation. The relation found in the query is either equal to, or linguistically related to the relations found in the statements preceding the query.

Table 1 shows examples of Relation Extraction on our two datasets. In Example:Movement, each statement has the relation "MoveTo", while the query is related to "Location". The "Location" can be thought of as a result of the "MoveTo" relations. Example:Parentage has "Parent" relations as the information and "Grandparent" as the query.

4.2 Entity extraction

Once the relations have been identified, the next step is to identify the elements of the text (or the entities) that are participating in the respective relations. Doing so allows us to further enrich the representation with the addition of restrictions (often real-world ones), which allow the Relational TM to learn rules that best represent actions and their consequences in a concise, logical form. Since the datasets we are using here consist only of simple sentences, each relation is limited to having a maximum of two entities (the relations are unary or binary).

Table 1 Relation extraction

Sentence	*Relation*
Mary went to the office.	**MoveTo**
John moved to the hall	**MoveTo**
Where is Mary?	**Location**

Example:Movement

Sentence	*Relation*
Bob is a parent of Mary.	**Parent**
Bob is a parent of Jane.	**Parent**
Mary is a parent of Peter	**Parent**
Is Bob a grandparent of Peter?	**Grandparent**

Example:Parentage

Table 2 Entity extraction

Sentence	*Relation*	*Entities*
Mary went to the office.	**MoveTo**	**Mary, office**
John moved to the hall	**MoveTo**	**John, hall**
Where is Mary?	**Location**	**Mary, ?**

Example:Movement

Sentence	*Relation*	*Entities*
Bob is a parent of Mary.	**Parent**	**Bob, Mary**
Bob is a parent of Jane.	**Parent**	**Bob, Jane**
Mary is a parent of Peter	**Parent**	**Mary, Peter**
Is Bob a grandparent of Peter?	**Grandparent**	**Bob, Peter**

Example:Parentage

In this step, the more external world knowledge that can be combined with the extracted entities, the richer the resultant representation. In Table 2, Example:Movement, we could add the knowledge that "MoveTo" relation always involves a "Person" and a "Location". Or in Example:Parentage, "Parent" is always between a "Person" and a "Person". This could, for example, prevent questions like "Jean-Joseph Pasteur was the father of Louis Pasteur. Louis Pasteur is the father of microbiology. Who is the grandfather of microbiology?"

Note that, as per Fig. 4, it is only possible to start answering the query after both Relation Extraction and Entity Extraction have been performed, and not before. Knowledge of the Relation also allows us to narrow down possible entities for answering the query successfully.

4.3 Entity generalization

One of the drawbacks of the relational representation is that there is a huge increase in the number of possible relations as more and more examples are processed. One way to reduce the spread is to reduce individual entities from their specific identities to a more generalised identity. Let us consider two instances : "Mary went to the office. John moved to the hall. Where is Mary?" and "Sarah moved to the garage. James went to the kitchen. Where is Sarah?". Without generalization, we end up with six different relations : MoveTo(Mary, Office), MoveTo(John, Hall), Location(Mary), MoveTo(Sarah, Garage), MoveTo(James, Kitchen), Location(Sarah). However, to answer either of the two queries, we only need the relations pertaining to the query itself. Taking advantage of that, we can generalize both instances to just 3 relations: MoveTo(Person_1, Location_1), MoveTo(Person_2, Location_2) and Location(Person_1).

In order to prioritise, the entities present in the query relation are the first to be generalized. All occurrences of those entities in the relations preceding the query are also replaced by suitable placeholders (Table 3).

The entities present in the other relations are then replaced by what can be considered as free variables (since they do not play a role in answering the query) (Table 4).

In the next section we explain how this relational framework is utilized for question answering using TM.

We have not gone into the details of the relation extraction and entity extraction processes in this work. Depending upon the initial data, there can be different strategies for the same. In a related work using TM on natural text data, the researchers show that a TM based classification mechanism successfully identifies various semantic relation the text. Such a system could be a possible way to identify and extract various relations from the text. On the other hand, there are also existing methods that can be used for relation identification.

Table 3 Entity generalization : Part 1

Sentence	*Relation*	*Entities*		*Reduced Relation*
Mary went to the office.	**MoveTo**	**Mary, office**	Source	**MoveTo**(X, office)
John moved to the hall.	**MoveTo**	**John, hall**	Source	**MoveTo**(John, hall)
Where is Mary?	**Location**	**Mary, ?**	Target	**Location**(X, ?)

Example:Movement

Sentence	*Relation*	*Entities*		*Reduced Relation*
Bob is a parent of Mary.	**Parent**	**Bob, Mary**	Source	**Parent**(X, Mary)
Bob is a parent of Jane.	**Parent**	**Bob, Jane**	Source	**Parent**(X,Jane)
Mary is a parent of Peter	**Parent**	**Mary, Peter**	Source	**Parent**(Mary, Y)
Is Bob a grandparent of Peter?	**Grand-parent**	**Bob, Peter**	Target	**Grandparent**(X, Y)

Example:Parentage

For example, in one of our ongoing work, we use data derived from the PULSAR system, where raw natural language text on various human rights violations from across the world has already been processed by experts into a database-based ecosystem (Park et al., 2020). The relational aspect of a database system allows for access to relations and entities that have already been extracted with a certain level of confidence, and leaves our system to focus on enhancing existing logical constructs or extracting newer ones based on the given information. A similar approach would be to use available knowledge-bases, e.g. NELL or Freebase, to use extracted relations as an initial 'seed' set to infer or extract newer ones.

4.4 Computational complexity in relation TM

One of the primary differences between the relational framework proposed in this paper versus the existing TM framework lies in Relation Extraction and Entity Generalization. The reason for these steps is that they allow us more flexibility in terms of what the TM learns, while keeping the general learning mechanism unchanged.

Extracting relations allows the TM to focus only on the major operations expressed via language, without getting caught up in multiple superfluous expressions of the same thing. It also enables to bridge the gap between structured and unstructured data. Using relations helps the resultant clauses be closer to real-world phenomena, since they model actions and consequences, rather than the interactions between words.

Entity Generalization allows the clauses to be succinct and precise, adding another layer of abstraction away from specific literals, much like Relation Extraction. It also gives the

Table 4 Entity Generalization : Part 2

Sentence	*Relation*	*Entities*		*Reduced Relation*
Mary went to the office.	**MoveTo**	**Mary, office**	Source	**MoveTo**(X, A)
John moved to the hall.	**MoveTo**	**John, hall**	Source	**MoveTo**(Y, B)
Where is Mary?	**Location**	**Mary, ?**	Target	**Location**(X, ?)

Example:Movement

Sentence	*Relation*	*Entities*		*Reduced Relation*
Bob is a parent of Mary.	**Parent**	**Bob, Mary**	Source	**Parent**(X, Z)
Bob is a parent of Jane.	**Parent**	**Bob, Jane**	Source	**Parent**(X, W)
Mary is a parent of Peter	**Parent**	**Mary, Peter**	Source	**Parent**(Z, Y)
Is Bob a grandparent of Peter?	**Grand-parent**	**Bob, Peter**	Target	**Grandparent**(X, Y)

Example:Parentage

added benefit of making the learning of the TM more generalized. In fact, due to this process, the learning reflects 'concepts' gleaned from the training examples, rather than the examples themselves.

To evaluate computational complexity, we employ the three costs α, β, and γ , where in terms of computational cost, α is cost to perform the conjunction of two bits, β is cost of computing the summation of two integers, and γ is cost to update the state of a single automaton. In a Propositional TM, the worst case scenario would be the one in which all the clauses are updated. Therefore, a TM with m clauses and an input vector of o features, needs to perform $(2o + 1) \times m$ number of Tsetlin automaton (TA) updates for a single training sample. For a total of d training samples, the total cost of updating is $d \times \gamma \times (2o + 1) \times m$.

Once weight updates have been successfully performed, the next step is to calculate the clause outputs. Here the worst case is represented by all the clauses including all the corresponding literals, giving us a cost of $\alpha \times 2o \times m$ (for a single sample).

The last step involves calculating the difference in votes from the clause outputs, thus incurring a per-sample cost of $\beta \times (m - 1)$.

Taken together, the total cost function for a Propositional TM can be expressed as:

$$f(d) = d \times [(\gamma \times (2o + 1) \times m) + (\alpha \times 2o \times m) + (\beta \times (m - 1))] \tag{5}$$

Expanding this calculation to the Relation TM scenario, we need to account for the extra operations being performed, as detailed earlier: Relation Extraction and Entity Generalization. The number of features per sample is restricted by the number of possible relations, both in the entirety of the training data, as well as only in the context of a single sample. For example, in the experiments involving "MoveTo" relations, we have restricted our data to have 3 statements, followed by a question (elaborated further in the next section). Each statement gives rise to a single "MoveTo" relation, which has 2 entities (a location and a person).

When using the textual constants (i.e., without Entity Generalization), the maximum number of possible features thus becomes equal to the number of possible combination between the unique entities in the relations. Thus if each sample contains r Relations, and a Relation R involves e different entities (E_1, E_2, ..., E_e), and cardinality of sets E_1, E_2, ..., E_e be represented as $|E_1|$, $|E_2|$, ..., $|E_e|$, the number of features in Eq. 5 can be re-written as

$o = [|E_1| \times |E_2| \times ... \times |E_e|] \times r$.

As discussed earlier in Section 4.3 as well as shown in the previous paragraph, this results in a large o, since it depends on the number of unique textual elements in each of the entity sets. Using Entity Generalization, the number of features no longer depends on the cardinality of set E_n, $1 \leq n \leq e$ in the context of the whole dataset. Instead, it only depends on the context of the single sample. Thus, if each sample contains r Relations, and a Relation R involves e different entities (E_1, E_2, ..., E_e), and maximum possible cardinality of sets E_1, E_2, ..., E_e are $|E_1| = |E_2| = ... = |E_n| = r$, the number of features simply become :

$o = r^{(e+1)}$.

In most scenarios, this number is much lower than the one obtained without the use of Entity Generalization.

A little further modification is required when using the convolutional approach. In calculating $f(d)$, the measure of o remains the same as we just showed with/without Entity Generalization. However the second term in the Eq. 5, which refers to the calculation of clause outputs ($d \times \alpha \times 2o \times m$), changes due to the difference in the mechanism of calculating outputs for convolutional and non-convolutional approaches. In the convolutional approach, with Entity Generalization, we need to consider free and bound variables in the feature representation. Bound variables are the ones which are linked by appearance in the

source and target relations, while the other variables, which are independent of that restriction, can be referred to as the free variables. Each possible permutation of the free variables in different positions are used by the convolutional approach to determine the best generic rule that describes the scenario. In certain scenarios, it may be possible to have certain permutations among the bound variables as well, without violating the restrictions added by the relations. One such scenario is detailed with an example in Section 5.2, where a bound "Person" entity can be permuted to allow any other "Person" entity, as long as the order of "MoveTo" relations are not violated. However, it is difficult to get a generic measure for the same, which would work irrespective of the nature of the relation (or their restrictions). Therefore, for the purpose of this calculation, we only take into account the permutations afforded to us by the free variable. Using the same notation as before, if each sample contains r Relations, and a Relation R involves e different entities, the total number of variables is $R \times e$. Of these, if v is the number of free variables, then they can be arranged in $v!$ different ways. Assuming v is constant for all samples, the worst case $f(d)$ from Eq. 5 can thus be rewritten as:

$$f(d) = d \times [(\gamma \times (2o + 1) \times m) + (v! \times \alpha \times 2o \times m) + (\beta \times (m - 1))] \tag{6}$$

5 Experimental study

To further illustrate how the TM based logic modelling works practically, we employ examples from a standard question answering dataset (Weston et al., 2015), which defines a set of prerequisite tasks as part of QA. For the scope of this work, we limit ourselves to the first subtask as defined in the dataset, viz. a question that can be answered by the preceding context, and the context contains a single supporting fact.

To start with, there is a set of statements, followed by a question, as discussed previously. For this particular subtask, the answer to the question is obtained by a single statement out of the set of statements provided (hence the term, single supporting fact). Choosing samples where the context contained a maximum of 6 sentences, this resulted in a set of 6000 unique sentences.

Input:*William moved to the office. Susan went to the garden. William walked to the pantry. Where is William?*

Output: pantry

We assume the following knowledge to help us construct the task:

1. All statements only contain information pertaining to relation MoveTo
2. All questions only relate to information pertaining to relation CurrentlyAt
3. Relation MoveTo involves 2 entities, such that $MoveTo(a, b) : a \in \{Persons\}$, $b \in \{Locations\}$
4. Relation CurrentlyAt involves 2 entities, such that $CurrentlyAt(a, b) : a \in \{Persons\}, b \in \{Locations\}$
5. MoveTo is a time-bound relation, it's effect is superseded by a similar action performed at a later time. In this scenario, 'later time' is indicated by position in text.

5.1 Without entity generalization

The size of the set of statements from which the model has to identify the correct answer influences the complexity of the task. For the purpose of this experiment, the data is capped

to have a maximum of three statements per input, and over all has five possible locations. This means that the task for the TM model is reduced to classifying the input into one of five possible classes.

To prepare the data for the TM, the first step involves reducing the input to relation-entity bindings. These bindings form the basis of our feature set, which is used to train the TM. We encode sentence order into the features as well, since MoveTo is time-sensitive (as mentioned in assumed world knowledge previously). Consider the following input example:

Input $=>$ $MoveTo(S1, William, Office)$, $MoveTo(S2, Susan, Garden)$, $MoveTo(S3, William, Pantry)$, $Q(William)$.

Since the TM requires binary features, each input is converted to a vector, where each element represents the presence (or absence) of the relationship instances. Secondly, the list of possible answers is obtained from the data, which is the list of class labels. Continuing our example, possible answers to the question could be:

Possible Answers : [Office, Pantry, Garden, Foyer, Kitchen]

Once training is complete, we can use the inherent interpretability of the TM to get an idea about how the model learns to discriminate the information given to it. The set of all clauses arrived at by the TM at the end of training represents a global view of the learning, i.e. what the model has learnt in general. The global view can also be thought of as a description of the task itself, as understood by the machine. We also have access to a local snapshot, which is particular to each input instance. The local snapshot involves only those clauses that help in arriving at the answer for that particular instance.

Table 5 shows the local snapshot obtained for the above example. As mentioned earlier, the TM model depends on two sets of clauses for each class, a positive set and a negative set. The positive set represents information favour of the class, while the negative set represents the opposite. The sum of the votes given by these two sets thus represent the final class the model decides on. As seen in the example, all the classes other than "Pantry" receive more negative votes than positive, making it the winning class. The clauses themselves allow us

Table 5 Local Snapshot of Clauses for example "William moved to the office. Susan went to the garden. William walked to the pantry. Where is William?" The winning vote is indicated in bold

Class	Clause	+/-	Votes	Total Votes for Class
Office	Q(William) AND MoveTo(S1,William, Office)	+	12	-35
	Q(William) AND MoveTo(S1,William, Office) AND MoveTo(S3,William, Pantry)	-	47	
Pantry	Q(William) AND MoveTo(S1,William, Office) AND MoveTo(S3,William, Pantry)	+	64	**+49**
	Q(William) AND MoveTo(S1,William, Office)	-	15	
Garden	MoveTo(S2,Susan, Garden)	+	12	-36
	Q(William) AND MoveTo(S1,William, Office) AND MoveTo(S3,William, Pantry)	-	48	
Foyer	-	+	0	-106
	Q(William) AND MoveTo(S1,William, Office) AND MoveTo(S3,William, Pantry) AND MoveTo(S2,Susan, Garden)	-	106	
Kitchen	-	+	0	-113
	Q(William) AND MoveTo(S1,William, Office) AND MoveTo(S3,William, Pantry) AND MoveTo(S2,Susan, Garden)	-	113	

to peek into the learning mechanism. For the class "Office", a clause captures the information that (a) the question contains "William", and (b) the relationship MoveTo(S1,William, Office) is available. This clause votes in support of the class, i.e. this is an evidence that the answer to the question "Where is William?" may be "Office". However, another clause encapsulates the previous two pieces of information, as well as something more : (c) the relationship MoveTo(S3,William, Pantry) is available. Not only does this clause vote against the class "Office", it also ends up with a larger share of votes than the clause voting positively, thus pushing "Office" out of contention.

The accuracy obtained over 100 epochs for this experiment was 94.83%, with a F-score of 94.80.

5.1.1 Allowing negative literals in clauses

Above results were obtained by only allowing positive literals in the clauses. The descriptive power of the TM goes up if negative literals are also added. However, the drawback to that is, while the TM is empowered to make more precise rules (and by extension, decisions), the overall complexity of the clauses increase, making them less readable. Using the above example again, if we do allow negative literals the positive evidence for Class Office looks like:

$$Q(William)\ AND\ Not(Q(Susan))\ AND\ MoveTo(S1, William, Office)\ AND\ Not(MoveTo(S3, William, Garden))\ AND\ Not(MoveTo(S3, William, Foyer))\ AND\ Not(MoveTo(S3, William, Kitchen))\ AND\ Not(MoveTo(S3, Susan, Garden))\ AND\ Not(MoveTo(S3, Susan, Office))\ AND\ Not(MoveTo(S3, Susan, Pantry))\ AND\ Not(MoveTo(S3, Susan, Foyer))\ AND\ Not(MoveTo(S3, Susan, Kitchen)).$$

At this point, we can see that the use of constants lead to a large number of repetitive information in terms of the rules learnt by the TM. Generalizing the constants as per their entity type can prevent this.

5.2 Entity generalization

Given a set of sentences and a following question, the first step remains the same as in the previous subsection, i.e. reducing the input to relation-entity bindings. In the second step, we carry out a grouping by entity type, in order to generalize the information. Once the constants have been replaced by general placeholders, we continue as previously, collecting a list of possible outputs (to be used as class labels), and further, training a TM based model with binary feature vectors.

As before, the data is capped to have a maximum of three statements per input. Continuing with the same example as above,

Input:*William moved to the office. Susan went to the garden. William walked to the pantry. Where is William?*

Output: pantry

1. Reducing to relation-entity bindings:

Input => $MoveTo(S1, William, Office)$, $MoveTo(S2, Susan, Garden)$, $MoveTo(S3, William, Pantry)$, $Q(William)$.

2. Generalizing bindings:

Modified=> $MoveTo(S1, Per1, Loc1)$, $MoveTo(S2, Per2, Loc2)$, $MoveTo(S3, Per1, Loc3)$, $Q(Per1)$

3. Possible Answers : [Loc1, Loc2, Loc3]

Table 6 Clause snapshot for "William moved to the office. Susan went to the garden. William walked to the pantry. Where is William?" after generalization. The winning vote is indicated in bold

Class	Clause	+/-	Votes	Total Votes for Class
Loc1	Q(Per1) AND MoveTo(S1,Per1, Loc1)	+	3	-44
	Q(Per1) AND MoveTo(S1,Per1, Loc1) AND MoveTo(S3,Per1, Loc3)	-	47	
Loc2	MoveTo(S1,Per1, Loc1)	+	2	-88
	Q(Per1) AND MoveTo(S3,Per1, Loc3)	-	90	
Loc3	Q(Per1) AND MoveTo(S1,Per1, Loc1) AND MoveTo(S3,Per1, Loc3)	+	51	**+51**
	-	-	0	

The simplifying effect of generalization is seen right away: even though there are 5 possible location in the whole dataset, for any particular instance there are always maximum of three possibilities, since there are maximum three statements per instance.

As seen from the local snapshot (Table 6), the clauses formed are much more compact and easily understandable. The generalization also releases the TM model from the restriction of having had to seen definite constants before in order to make a decision. The model can process "Rory moved to the conservatory. Rory went to the cinema. Cecil walked to the school. Where is Rory?", without needing to have encountered constants "Rory", "Cecil", "school", "cinema" and "conservatory".

Accuracy for this experiment over 100 epochs was 99.48% , with a F-score of 92.53.

A logic based understanding of the relation "Move" could typically be expressed as :

$\rightarrow MoveTo(S1, William, office) + MoveTo(S2, Susan, garden) + MoveTo(S3, William, pantry)$

$\rightarrow MoveTo(S1, P_1, office) + MoveTo(S2, P_2, garden) + MoveTo(S3, P_1, pantry)$

$\rightarrow MoveTo(S1, P_1, L_1) + MoveTo(S2, P_2, L_2) + MoveTo(S3, P_1, L_3)$

$\rightarrow MoveTo(S1, P_1, L_1) + * + MoveTo(Sn, P_1, L_n)$

$\implies Location(P_1, L_n).$

From the above two subsections, we can see that with more and more generalization, the learning encapsulated in the TM model can approach what could possibly be a human-level understanding of the world.

Moreover, as shown in Table 7, compared to the Propositional TM, not only does the Relational TM give a more compact and easily understandable explanation, it also has the

Table 7 Representative Winning Clauses for "William moved to the office

Propositional TM	Relational TM	Relational TM with entity generalization
Q(William) AND Q(Where) AND S1_William AND S1_Office AND S1_moved AND S3_William AND S3_Pantry AND S3_walked AND NOT(S3_Garden) AND NOT(S3_Susan)	Q(William) AND MoveTo (S1,William, Office) AND MoveTo (S3,William, Pantry)	Q(Per1) AND MoveTo(S1,Per1, Loc1) AND MoveTo(S3,Per1, Loc3)

Susan went to the garden. William walked to the pantry. Where is William?" with Propositional, Relational and Relational TM with Entity Generalization

added advantage of allowing reasoning to be built on top of obtained explanations (as discussed above). How different entities interact with each other in a given environment, and what is the result of those interactions, is something that is very important to be able to successfully model the world around us, rather than just the presence (or absence) of certain words in a textual description of that environment. This can only be captured by the Relational TM, and not by the propositional one.

5.3 Variable permutation and convolution

As described in Section 3.2.5, we can produce all possible permutations of the available variables in each sample (after Entity Generalization) as long as the relation constraints are not violated. Doing this gives us more information per sample:

Input:*William moved to the office. Susan went to the garden. William walked to the pantry. Where is William?*

Output: pantry

1. Reducing to relation-entity bindings:

Input => $MoveTo(S1, William, Office)$, $MoveTo(S2, Susan, Garden)$, $MoveTo(S3, William, Pantry)$, $Q(William)$.

2. Generalizing bindings:

Modified=> $MoveTo(S1, Per1, Loc1)$, $MoveTo(S2, Per2, Loc2)$, $MoveTo(S3, Per1, Loc3)$, $Q(Per1)$

3. Performing Permutations:

Permuted=> $MoveTo(S1, Per2, Loc1)$, $MoveTo(S2, Per1, Loc2)$, $MoveTo(S3, Per2, Loc3)$, $Q(Per2)$

4. Possible Answers : [Loc1, Loc2, Loc3]

This has two primary benefits. Firstly, in a scenario where the given data does not encompass all possible structural differences in which a particular information maybe represented, using the permutations allows the TM to view a closer-to-complete representation from which to build it's learning (and hence, explanations). Moreover, since the TM can learn different structural permutations from a single sample, it ultimately requires fewer clauses to learn effectively. In our experiments, permutations using Relational TM Convolution allowed for up to 1.5 times less clauses than using a non-convolutional approach.

As detailed in Section 4.4, convolutional and non-convolutional approaches have different computational complexity. Hence, the convolutional approach makes sense only when the reduction in complexity from fewer clauses balance the increase due to processing the convolutional window itself.

5.4 Comparison with some other QA approaches

We use the following related approaches as a guide for judging the performance of the proposed method on the task (Table 8). All the approaches were trained (where applicable) and tested on the same set as described earlier for our proposed approach, i.e. collection

Table 8 Comparing accuracies with other QA approaches

Method	Relational TM	RTM+ Entity Gen.	Mem Net	Cross- encoders	BART	BERT+ Bi-Encoder
Accuracy	94.8	99.5	98.2	99.6	98.5	84.5

of 6000 samples from the single supporting fact dataset. Implementations of the same have been suitably modified from the ParlAI platform (Miller et al., 2017).

- MemNet (Weston et al., 2014) integrates basic inference with long-term memory component that is used for prediction of output (or answer).
- Cross-encoders (Wolf et al., 2019) perform full (cross) self-attention over a given input and label candidate (with pre-trained representations). This is a neural network based model.
- BART (Lewis et al., 2019) adds noise to the text and trains by learning a model to reconstruct the original. This is a Tranformer-based neural machine translation model.
- BERT-based Bi-encoder (Lu et al., 2020) which uses two separate representations of the document and query based on a BERT-like model. We use the pre-trained large language model as available and do not train it specifically on our data.

While detailed comparisons in terms of memory usage and training times with these models are beyond the scope of this work, other researchers have favourably compared the performance of Tsetlin Machine based systems to similar models (Yadav et al., 2021b; Bhattarai et al., 2021). As an example, while the MemNet takes an average of 220 seconds to train on the current dataset, the RTM takes 218 seconds (both averaged over 100 separate runs).

5.5 Noise tolerance

To verify our claims of noise tolerance as shown by the TM based architecture, the above experiments were repeated, but with increasing amount of noise artificially introduced into the training data. The results are shown in Table 9. We observe that with 1%, 2%, 5% and 10% of noise, the testing accuracy fell only by approximately 1.1 percentage points each time when entity generalization was used.

Along with the obvious strengths of having a transparent explainable system, the noise tolerance is one of the more compelling pros of the TM based solution. Typical methods are more isomorphic to the text provided, along with its imperfections, and not capable of generalizing like the TM. Comparing with accuracies obtained from a few other standard classification techniques, viz. CNN-LSTM, Naive Bayes, and Random Forests, the drop is on average 3-5%.

5.6 Horn clause representation

The example elaborated in the previous section can be formulated as the following Horn clause representation:

1. $Person(Susan)$.
2. $Person(William)$.
3. $Location(Office)$.
4. $Location(Garden)$.

Table 9 Average accuracy on test with increase in error in training data

Error Rate	0%	1%	2%	5%	10%
Accuracy	99.48	98.79	98.24	97.02	95.08

5. $Location(Pantry)$.
6. $CurrentlyAt(Susan, Pantry)$.
7. $CurrentlyAt(William, Pantry)$.
8. $MoveTo(Susan, Garden) \leftarrow Person(Susan), Location(Garden), \quad not\ CurrentlyAt(Susan, Garden)$
9. $MoveTo(William, Office) \leftarrow Person(William), Location(Office), not\ CurrentlyAt(William, Office)$.

After generalization, we substitute ground rules 8 and 9 with the following rule:

10. $MoveTo(P, L) \leftarrow Person(P), Location(L), not\ CurrentlyAt(P, L)$.

The above set of Horn clauses define the immediate consequences operator whose LFP represents the Herbrand interpretation of our QA framework.

6 Conclusion

Making interpretable logical decisions in question answering system is an area of active research. In this work, we propose a novel relational logic based TM framework to approach QA tasks systematically. Our proposed method takes advantage of noise tolerance showed by TMs to work in uncertain or ambiguous contexts. We reduce the context-question-answer tuples to a set of logical arguments, which is used by the TM to determine rules that mimic real-world actions and consequences.

The resulting TM is relational (as opposed to the previously propositional TM) and can take work on logical structures that occur in natural language in order to encode rules representing actions and effects in the form of Horn clauses. We show initial results using the Relational TM on artificial datasets of closed-domain question answering, and those results are extremely promising. The use of first-order representations, as described in this paper, allows KBs to be up to 10 times smaller, while at the same time showing an answering accuracy increase of almost 5% to 99.48%.

Further work on this framework will involve a larger number of relations, with greater inter-dependencies, and analyzing how well the TM can learn the inherent logical structure governing such dependencies. We also intend to introduce recursive Horn clauses to make the computing power of the Relational TM equivalent to a universal Turing machine. Moreover, we wish to experiment with this framework on real-world natural language datasets, rather than on toy ones. A prominent example is exploring large corpus of documents related to human rights violation and using them to assess risks of social instability. We expect that the resultant logic structures will be large and complicated, however, once obtained, can be used to effectively translate to and fro between the machine world and the real world.

Data Availability The toy tasks dataset analysed during the current study are available publicly in the BABI repository (http://www.thespermwhale.com/jaseweston/babi/tasks_1-20_v1-2.tar.gz). The data referring to parent-child relationships, generated for analysis during the current study are available from the corresponding author on reasonable request.

Declarations

Conflict of Interest The Authors declare that there is no conflict of interest.

References

Abeyrathna, K. D., Granmo, O. C., & Goodwin, M. (2021). extending the tsetlin machine with Integer-Weighted clauses for increased interpretability. *IEEE Access*, 9.

Abeyrathna, K. D., Granmo, O. C., Zhang, X., Jiao, L., & Goodwin, M. (2019). The regression tsetlin machine - a novel approach to interpretable Non-Linear regression. *Philosophical Transactions of the Royal Society A*, 378.

Berge, G. T., Granmo, O. C., Tveit, T. O., Goodwin, M., Jiao, L., & Matheussen, B.V. (2019). Using the Tsetlin Machine to learn human-interpretable rules for high-accuracy text categorization with medical applications. *IEEE Access, 7,* 115134–115146.

Bhattarai, B., Jiao, L., & Granmo, O.C. (2021). Measuring the Novelty of Natural Language Text Using the Conjunctive Clauses of a Tsetlin Machine Text Classifier. In *13Th international conference on agents and artificial intelligence (ICAART 2021)*. INSTICC.

Blakely, C. D., & Granmo, O. C. (2020). Closed-Form Expressions for Global and Local Interpretation of Tsetlin Machines with Applications to Explaining High-Dimensional Data. arXiv:2007.13885.

Bollacker, K., Evans, C., Paritosh, P., Sturge, T., & Taylor, J. (2008). Freebase: a collaboratively created graph database for structuring human knowledge. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data* (pp. 1247–1250).

Bratko, I., & Muggleton, S. (1995). Applications of inductive logic programming. *Communications of the ACM*, *38*(11), 65–70.

Cropper, A., Dumančić, S., & Muggleton, S.H. (2020). Turning 30: New ideas in inductive logic programming. arXiv:2002.11002.

Cyras, K., Badrinath, R., Mohalik, S. K., Mujumdar, A., Nikou, A., Previti, A., Sundararajan, V., & Feljan, A. V. (2020). Machine reasoning explainability. arXiv:2009.00418.

De Raedt, L., & Kersting, K. (2008). Probabilistic inductive logic programming. In *Probabilistic inductive logic programming* (pp. 1–27). Springer.

Dong, J. S., Sun, J., & Wang, H. (2003). Checking and reasoning about semantic web through alloy. In *International symposium of formal methods europe* (pp. 796–813). Springer.

Gallaire, H., Minker, J., & Nicolas, J. M. (1989). Logic and databases: a deductive approach. In *Readings in artificial intelligence and databases* (pp. 231–247). Elsevier.

Granmo, O. C. (2018). The Tsetlin Machine - A Game Theoretic Bandit Driven Approach to Optimal Pattern Recognition with Propositional Logic. arXiv:1804.01508.

Granmo, O. C., Glimsdal, S., Jiao, L., Goodwin, M., Omlin, C. W., & Berge, G.T. (2019). The Convolutional Tsetlin Machine. arXiv:1905.09688.

Green, C. (1969). Theorem proving by resolution as a basis for question-answering systems. *Machine Intelligence*, *4*, 183–205.

Jarke, M., Gallersdörfer, R., Jeusfeld, M. A., Staudt, M., & Eherer, S. (1995). Conceptbase—a deductive object base for meta data management. *Journal of Intelligent Information Systems*, *4*(2), 167–192.

Jung, H., & Kim, W. (2020). Automated conversion from natural language query to sparql query. *Journal of Intelligent Information Systems*, 1–20.

Kowalski, R. (1979). Algorithm= logic+ control. *Communications of the ACM*, *22*(7), 424–436.

Kowalski, R. (2014). Logic programming. In J. H. Siekmann (Ed.) *Computational logic, handbook of the history of logic*, (Vol. 9 pp. 523–569). North-holland. https://doi.org/10.1016/B978-0-444-51624-4.50012-5.

Kramer, S., Lavrač, N., & Flach, P. (2001). Propositionalization approaches to relational data mining. *Relational data mining*, 262–291.

Lei, J., Rahman, T., Shafik, R., Wheeldon, A., Yakovlev, A., Granmo, O. C., Kawsar, F., & Mathur, A. (2021). Low-Power Audio Keyword Spotting using Tsetlin Machines. arXiv:2101.11336.

Lei, J., Wheeldon, A., Shafik, R., Yakovlev, A., & Granmo, O.C. (2020). From arithmetic to logic based AI: a comparative analysis of neural networks and tsetlin machine. In *27Th IEEE international conference on electronics circuits and systems (ICECS2020)*. IEEE.

Lewis, M., Liu, Y., Goyal, N., Ghazvininejad, M., Mohamed, A., Levy, O., Stoyanov, V., & Zettlemoyer, L. (2019). Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. arXiv:1910.13461.

Lloyd, J. (1984). *Foundations of logic programming*. New York: Springer.

Lu, W., Jiao, J., & Zhang, R. (2020). Twinbert: Distilling knowledge to twin-structured compressed bert models for large-scale retrieval. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management* (pp. 2645–2652).

Ludwig, S. A. (2010). Comparison of a deductive database with a semantic web reasoning engine. *Knowledge-Based Systems*, *23*(6), 634–642.

Miller, A. H., Feng, W., Fisch, A., Lu, J., Batra, D., Bordes, A., Parikh, D., & Weston, J. (2017). Parlai: A dialog research software platform. arXiv:1705.06476.

Mitchell, T., Cohen, W., Hruschka, E., Talukdar, P., Yang, B., Betteridge, J., Carlson, A., Dalvi, B., Gardner, M., Kisiel, B., & et al. (2018). Never-ending learning. *Communications of the ACM*, *61*(5), 103–115.

Nickles, M., & Mileo, A. (2014). Probabilistic inductive logic programming based on answer set programming. arXiv:1405.0720.

Park, B., Greene, K., & Colaresi, M. (2020). How to teach machines to read human rights reports and identify judgments at scale. *Journal of Human Rights*, *19*(1), 99–116.

Phoulady, A., Granmo, O. C., Gorji, S. R., & Phoulady, H.A. (2019). The Weighted Tsetlin Machine: Compressed Representations with Weighted Clauses. arXiv:1911.12607.

Prager, J. M. (2006). Open-domain question-answering. *Found Trends Inf Retr*, *1*(2), 91–231.

Pundge, A. M., Khillare, S., & Mahender, C. N. (2016). Question answering system, approaches and techniques: a review. *International Journal of Computer Applications*, *141*(3), 0975–8887.

Rajpurkar, P., Zhang, J., Lopyrev, K., & Liang, P. (2016). Squad: 100,000+ questions for machine comprehension of text. arXiv:1606.05250.

Saha, R., Granmo, O. C., & Goodwin, M. (2020). Mining Interpretable Rules for Sentiment and Semantic Relation Analysis using Tsetlin Machines. In *Lecture Notes in Computer Science: Proceedings of the 40th International Conference on Innovative Techniques and Applications of Artificial Intelligence (SGAI-2020)*. Springer.

Shafik, R., Wheeldon, A., & Yakovlev, A. (2020). Explainability and Dependability Analysis of Learning Automata based AI Hardware. In *IEEE 26Th international symposium on on-line testing and robust system design (IOLTS)*. IEEE.

Soares, M. A. C., & Parreiras, F. S. (2020). A literature review on question answering techniques, paradigms and systems. *Journal of King Saud University-Computer and Information Sciences*, *32*(6), 635–646.

Srinivasan, V., Santhanam, S., & Shaikh, S. (2021). Using reinforcement learning with external rewards for open-domain natural language generation. *Journal of Intelligent Information Systems*, *56*(1), 189–206.

Tsetlin, M. L. (1961). On behaviour of finite automata in random medium. *Avtom I Telemekhanika*, *22*(10), 1345–1354.

Turhan, A. Y. (2011). Description logic reasoning for semantic web ontologies. In *Proceedings of the International Conference on Web Intelligence, Mining and Semantics* (pp. 1–5).

Weston, J., Bordes, A., Chopra, S., Rush, A. M., van Merriënboer, B., Joulin, A., & Mikolov, T. (2015). Towards ai-complete question answering: A set of prerequisite toy tasks. arXiv:1502.05698.

Weston, J., Chopra, S., & Bordes, A. (2014). Memory networks. arXiv:1410.3916.

Wheeldon, A., Shafik, R., Yakovlev, A., Edwards, J., Haddadi, I., & Granmo, O.C. (2020). Tsetlin machine: a new paradigm for pervasive AI. In *SCONA Workshop at design, automation and test in europe (DATE 2020)*.

Wolf, T., Sanh, V., Chaumond, J., & Delangue, C. (2019). Transfertransfo: A transfer learning approach for neural network based conversational agents. arXiv:1901.08149.

Yadav, R. K., Jiao, L., Granmo, O. C., & Goodwin, M. (2021). Human-Level interpretable learning for Aspect-Based sentiment analysis. In *The thirty-fifth AAAI conference on artificial intelligence (AAAI-21)*. AAAI.

Yadav, R. K., Jiao, L., Granmo, O. C., & Goodwin, M. (2021). Interpretable classification of word sense disambiguation using tsetlin machine. In *13Th international conference on agents and artificial intelligence (ICAART 2021)*. INSTICC.