Solving Probability Problems in Natural Language

Anton Dries[†], Angelika Kimmig[†], Jesse Davis[†], Vaishak Belle[‡], Luc De Raedt[†]
[†] Department of Computer Science, KU Leuven, Belgium
[‡] University of Edinburgh, UK
{firstname.lastname}@cs.kuleuven.be

Abstract

The ability to solve probability word problems such as those found in introductory discrete mathematics textbooks, is an important cognitive and intellectual skill. In this paper, we develop a two-step end-to-end fully automated approach for solving such questions that is able to automatically provide answers to exercises about probability formulated in natural language.

In the first step, a question formulated in natural language is analysed and transformed into a high-level model specified in a declarative language. In the second step, a solution to the high-level model is computed using a probabilistic programming system.

On a dataset of 2160 probability problems, our solver is able to correctly answer 97.5% of the questions given a correct model. On the end-to-end evaluation, we are able to answer 12.5% of the questions (or 31.1% if we exclude examples not supported by design).

1 Introduction

Since the early days of AI, people have been fascinated by machines solving puzzles, exercises and exams that are used to test the intelligence of humans, see [Hernández-Orallo *et al.*, 2016] for an overview. The ability to solve such problems is an important cognitive and intellectual skill as it is evaluated as part of academic admission tests such as SAT, GMAT and GRE. Recently, there has been a surge of interest in mathematical and scientific problem solving. Most prominently, [Kushman *et al.*, 2014] study how to answer algebraic word problems, which contain typical questions about the "rule of three". In later work, [Seo *et al.*, 2015] have developed an approach to solving geometry questions that occur in SAT tests, which involve both a textual and a schematic description of the question.

There are essentially two requirements for successfully solving such problems: (a) correctly parse and analyse the question phrased in natural language, and (b) solve the underlying mathematical problem. The parsing and analysis of natural language descriptions of such questions is receiving a lot of attention in the empirical natural language processing community. The standard approach is to assemble a corpus of questions and then to learn how to correctly map onto a mathematical problem definition.

Next on the agenda, then, is to solve the resulting mathematical problem. While for high school algebraic questions standard solvers can be used, more involved domains such as geometrical reasoning have motivated the development of special purpose solvers [Alvin *et al.*, 2014; Seo *et al.*, 2014]. While mathematical problem solving has been studied for a long time [Mukherjee and Garain, 2008], probability problems have not received much attention yet. Actually, to the best of our knowledge, there is only one very early approach to solving probability questions [Gelb, 1971]. While Gelb's high-level approach is similar to ours, the various components are tackled in substantially different ways, e.g. no learning in the NLP part, no probabilistic programming in the solver part. Furthermore, it is unclear how complex the questions can be as the paper says "very basic probability problems" and we were unable to obtain more information about this work.

This paper develops a fully automated approach to solving exercises about probability that can be found in introductory textbooks on discrete mathematics, see Figure 1 for an example. The probability questions are formulated in natural language and the task is to automatically answer these questions. We develop a two-step approach for tackling this task. In the first step, a question formulated in natural language is analysed and transformed into a high-level model specified in a declarative language. In the second step, the high-level model is transformed into a probabilistic program and solved using the inference mechanisms of the underlying probabilistic programming language. Probabilistic programming [De Raedt and Kimmig, 2015] is an increasingly popular programming paradigm in which declarative programming languages are extended with probabilistic primitives and inference mechanisms. They allow concisely representing complex probabilistic models and they support machine learning.

The key contributions of this paper are 1) the introduction of a dataset of 2160 probability word problems as a challenge for AI, 2) the introduction of a declarative language for specifying probabilistic problems, 3) an effective solver

[1] An online version of our system is available at `https://dtai.cs.kuleuven.be/problog/natural_language`.

Q1: In a group of 10 people, 60 percent have brown eyes. Two people are to be selected at random from the group. What is the probability that neither person selected will have brown eyes?
Q2: Mike has a bag of marbles with 4 white, 8 blue, and 6 red marbles. He pulls out one marble from the bag and it is red. What is the probability that the second marble he pulls out of the bag is white?

Figure 1: Example questions.

based on probabilistic programming, and 4) an initial version of an end-to-end system for tackling the overall task. The natural language component is deliberately kept simple, and does not yet employ rich corpora for training such as [Seo *et al.*, 2015].

In Section 2, we introduce our modeling language for probability problems; in Section 3, we develop a solver for that language using the probabilistic programming language ProbLog. Section 4 then describes the natural language component, whose main task is to analyse the probability question and to transform the natural language input into a formal model that can be used by the probability solver to infer an answer to the question. In Section 5, we evaluate the approach on a novel dataset of 2160 probability problems, and we conclude in Section 6.

2 Modeling Probability Problems

We now introduce our formal modeling language for probability questions. In line with the mathematical problem solving literature [Hosseini *et al.*, 2014], we consider *entities*, their *properties*, and *containers* containing certain numbers of entities with certain properties. We further consider *actions* that relate containers into a probabilistic generative model, which provides the mathematical basis for answering probability questions.

2.1 Entities, Properties and Containers

We start with the basic building blocks. *Entities* are objects mentioned in the probability question, such as people or marbles. They have *properties* based on *attributes*, such as color. Each attribute A has a finite set of possible values $\mathsf{Values}(A)$, where we assume that no two attributes share a value. Values of attributes can be combined into properties using Boolean operators. For instance, $((9 \vee 10) \wedge \diamondsuit)$ is a property of playing cards based on $rank$ and $suit$ attributes. *Containers* are collections of entities. Containers will be represented by multisets, as this allows us to abstract away from the identity of entities and reason about groups of exchangeable individuals instead. For instance, in Q1, all persons with brown eyes are exchangeable. We restrict the discussion to (unordered) multisets, but the same principles also apply to *ordered* containers, i.e., sequences of objects, and our language supports both. We denote the size of a multiset M by $\#M$. For instance, in Q1, one container is the multiset $people$ with $\#people = 10$. If M is a container and φ a logical formula over possible values of the attributes, we use $\varphi(M)$ to refer to the sub(multi)set of M of all entities in M that satisfy φ. For instance, in Q1, we have a subset $browneyes(people)$.

2.2 Actions

In probability questions, we can distinguish two kinds of containers: those for which information about properties of their elements is given (such as the number of marbles of each color in Mike's bag), and those for which this information is uncertain (such as the first marble pulled out of the bag). The relationship between such containers is given through *actions* that define new containers in terms of existing ones. For instance, in Q2, we have an initial container bag, and two pull actions that each create two new containers: one for the marble pulled from the existing container, and one for the marbles that remain in the previous container. These actions introduce uncertainty about the content of containers; we know the colors of the marbles in the initial bag, but for the new containers, we only have probability distributions over the colors.

We capture this set of actions as a Bayesian network with a node or random variable for each container, which takes the content of the container as its value. Actions introduce arrows directed from their input containers to their output containers, that is, the first kind of container forms the (deterministic) root nodes of the network.

Our modeling language defines each root node by providing an identifier for the container, the set of values of all relevant attributes, and a set of linear equality constraints on the sizes of subsets of the container defined through these attributes. For instance, the root nodes for Q1 are given by

$$\begin{aligned} \mathsf{Values}(eye) &= \{browneyes\} \\ \text{multiset } group & \\ \#group &= 10 \\ \#browneyes(group) &= 0.6 \cdot \#group \end{aligned}$$

and those of Q2 by

$$\begin{aligned} \mathsf{Values}(color) &= \{white, blue, red\} \\ \text{multiset } bag & \\ \#white(bag) &= 4 \\ \#blue(bag) &= 8 \\ \#red(bag) &= 6 \end{aligned}$$

The value of a root node R is the partition $R_1, \ldots, R_n$ such that all $\#R_i$ are known and n is maximal, i.e., no R_i can further be partitioned while maintaining full knowledge of sizes. Constraint systems for which this partition is not unique are considered invalid, as a probability task needs to provide sufficient information to uniquely derive the starting situation. Note that this information can be partial, e.g., instead of specifying the number of cards for all combinations of values for $Rank$ and $Suit$, a task may just state that there are ten cards, five of them are $\diamondsuit$ or $\heartsuit$, and of those, one is a king or queen. It is then the task of the solver to infer the partition $(\diamondsuit \vee \heartsuit) \wedge (K \vee Q)(cards)$, $(\diamondsuit \vee \heartsuit) \wedge \neg(K \vee Q)(cards)$ and $\neg(\diamondsuit \vee \heartsuit)(cards)$ from these constraints.

We now specify how the actions defining new containers are modeled in our language:

If D is the result of *drawing N elements without replacement* from an already defined multiset M, the model contains two container definitions $D = \text{take}(M)$ and $\text{rest}(D) = M \setminus D$, and the size constraint $\#D = N$.

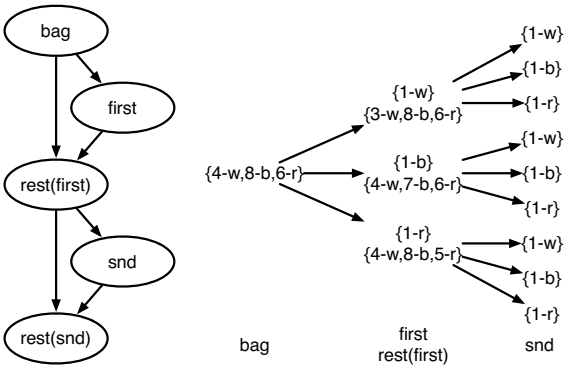

Figure 2: Bayesian network structure for Q2 (left) and tree illustrating possible container content for relevant nodes (right).

If D is the result of *drawing N elements with replacement* from an already defined multiset M, the model contains $D = \mathrm{takeReplace}(M)$ and $\#D = N$.

If D is the *disjoint multiset union* of a finite number k of already existing multisets M_i, the model contains the statement $D = \bigsqcup(M_1, \ldots, M_k)$.

Each definition of the form $D = \mathrm{action}(\mathbf{M})$ creating a new multiset D from a set of multisets $\mathbf{M}$ adds a node D whose parents are the nodes in $\mathbf{M}$ to the Bayesian network. For each of these cases, it is straightforward to define and derive the conditional probability distribution of the new nodes given their parents in the network.

We can model this information for Q1 as

$$\begin{aligned} twopers &= \mathrm{take}(group) \\ \mathrm{rest}(twopers) &= group \setminus twopers \\ \#twopers &= 2 \end{aligned}$$

and for Q2 as

$$\begin{aligned} first &= \mathrm{take}(bag) \\ \mathrm{rest}(first) &= bag \setminus first \\ \#first &= 1 \\ snd &= \mathrm{take}(\mathrm{rest}(first)) \\ \mathrm{rest}(snd) &= \mathrm{rest}(first) \setminus snd \\ \#snd &= 1 \end{aligned}$$

The left part of Figure 2 shows the network structure for Q2.

2.3 Questions and Observations

Finally, we also need to model the *observations* (in Q2, that the first marble is red) and *questions* (in Q2, whether the second marble is white) in our probability problems, which we want to answer based on the probability distribution over the content of containers defined above. More specifically, both observations and questions again refer to properties of objects, but now of objects in containers that result from actions. Furthermore, they are not restricted to linear equality constraints on numbers of objects with certain properties, but can refer to more complex constraints.

Specifically, our modeling language currently supports the following kinds of constraints, where $A(M)$ denotes the multiset of values attribute A takes on multiset M, and $|A(M)|$ denotes the number of different values in $A(M)$ (the number of count-value tuples):

size constraints of the form $X\, c\, Y$ with $c \in \{=, \leq, \geq, <, >\}$ and X and Y size variables or non-negative numbers;

type constraints of the form $|A(M)| = 1$ or $|A(M)| = \#M$ requiring all objects in M to have the same value for attribute A or each a different one, respectively.

aggregation constraints of the form $\bigoplus(A(M))\ c\ C$ or $\Psi(\bigoplus(A(M)))$ with a numerical attribute A (i.e., the objects in $A(M)$ are numbers), an aggregate function $\bigoplus \in \{\min, \max, \sum, \prod, \mathrm{avg}\}$ operating on multisets of numbers, a comparison operator $c \in \{=, \leq, \geq, <, >\}$, a constant C and a Boolean predicate Ψ (such as "is even", "is odd").

sequence constraints of the form $\mathrm{nth}(M, N) \in A(M)$ requiring that the N-th element of (an ordered non-root node) M satisfies attribute A.

These types of constraints can be combined into arbitrary Boolean formulas using $\wedge, \vee, \neg$.

Our two examples use size constraints only. In Q1, the question is $\#browneyes(twopers) = 0$, and in Q2, we observe $\#red(first) = \#first$ and ask for $\#white(snd) = \#snd$.

2.4 Models

To summarize, the declarative modeling language for probability problems provides statements to declare a set of containers (i.e., multisets) $\mathcal{M}$, a set of attributes $\mathcal{A}$ and their associated values, a set of size constraints $\mathcal{S}$ for the initial containers, a set of multiset relations $\mathcal{R}$ defining the other containers through actions, a set of observations $\mathcal{O}$, and a set of queries $\mathcal{Q}$.

A valid set of such statements defines a joint probability distribution $P(\mathcal{M})$ over the set of containers $\mathcal{M} = \{M_1, \ldots, M_n\}$ it defines, as specified above. The probability that a (question or observation) constraint C holds on a multiset M_i is then defined as usual as the sum of the probabilities of those assignments $M_1 = m_1, \ldots, M_n = m_n$ for which m_i satisfies C.

3 Solving Probability Problems

Given a probability problem definition in the language of Section 2, the task of the solver is to compute the conditional probability of each question given all observations. Note that every aspect of the Bayesian network we define depends on the specific task: the network structure is given by the containers and actions, the domains of the random variables depend on the attributes, the subset size constraints, and the actions, and the parameters of the conditional distributions in turn depend on the domains of random variables and the actions. Rather than materializing the Bayesian network for use with traditional Bayesian network software, we have implemented the solver in a probabilistic programming language. This has two key advantages. First, the expressivity of these languages allows us to implement the probabilistic model at the abstract level, by providing generic templates for its building blocks (e.g., for defining the nodes

and probability distributions from the actions). Second, by leaving all aspects of reasoning, including the instantiation of the model for the given problem definition, to the probabilistic programming system, we can take full advantage of the developments in this field. Such systems have been optimized to generate only those parts of the model that are relevant to the query. This is akin to what happens in statistical relational learning [Getoor and Taskar, 2007; De Raedt *et al.*, 2016], where only those parts of a template graphical model relevant to answering a query are grounded out.

Specifically, our solver is implemented in ProbLog [De Raedt *et al.*, 2007], and uses the following principles to implement the building blocks of the abstract model.

Root nodes. For each root node of the Bayesian network, the solver starts from the set of linear equality constraints on the sizes of the corresponding container and its property-based subsets in the input. It extends this equation system with the relevant instances of the generic size constraint that states that the size of a multiset union is the sum of the sizes of its parts. For instance, in the case of Q2, where we have a single attribute only, this adds the constraint $\#bag = \#white(bag) + \#blue(bag) + \#red(bag) + \#other_{color}(bag)$. Note that we add an explicit "other" value to the listed values of the attribute. We do this because probability problems do not necessarily list all values exhaustively. For instance, Q1 does not mention that there are people whose eyes have other colors, but this information is necessary to solve the problem. In Q2, the list of colors happens to be exhaustive, but if this problem would also state that the bag contains twenty marbles, the solver would have to infer that there are two marbles with a color different from the listed ones. In general, we add such constraints for all ways in which we can "sum out" the value of one attribute (in our example, color) from a conjunction of values from different attributes. For instance, if we consider a multiset $cards$, attribute $color$ with value red, and attribute $type$ with value $face$, we get (abbreviating $c(ards)$ and $o(ther)$)

$$\begin{aligned}
\#c &= \#red(c) + \#o_{color}(c)\\
\#c &= \#face(c) + \#o_{type}(c)\\
\#red(c) &= \#(red \wedge face)(c) + \#(red \wedge o_{type})(c)\\
\#o_{color}(c) &= \#(o_{color} \wedge face)(c) + \#(o_{color} \wedge o_{type})(c)\\
\#face(c) &= \#(red \wedge face)(c) + \#(o_{color} \wedge face)(c)\\
\#o_{type}(c) &= \#(red \wedge o_{type})(c) + \#(o_{color} \wedge o_{type})(c)
\end{aligned}$$

We then solve this extended system of linear equalities, subject to the additional constraints $\#\varphi(M) \geq 0$ for all its variables, using a combination of standard constraint solving techniques and a few domain specific steps that detect when size variables whose property φ involves "other" values can be set to zero.

Child nodes. For child nodes corresponding to multiset union or multiset difference (for the rest of taking without replacement), the child's content is deterministically computed from the content of the parents.

For child nodes corresponding to the result of taking N elements from the parent with or without replacement, the conditional probability distribution is defined through a chain of N probabilistic primitives, where each step copies or moves a single element from the parent's content to the child's content.

Question and Observations. For the constraints used in questions and observations, the solver contains definitions on the level of multiset content that are used for evaluating them.

While the components as described so far fully capture the model, and thus could directly be used for inference, we use the power of the probabilistic programming language to include two program transformations that further increase efficiency.

First, to exploit exchangeability even further, the solver program avoids distinctions based on irrelevant properties. This idea is central to *lifted* probabilistic inference [Poole, 2003; Niepert and Van den Broeck, 2014], and the use of probabilistic programming allows us to benefit from the corresponding efficiency improvements with a grounding-based inference engine. In Section 2, we have defined the content of root containers based on *maximally fine grained* partitions, as such partitions capture all the information available for such a container. However, often only a small part of this information is relevant to solve the specific task of interest. For instance, assume the most fine grained partition describes a deck of cards in terms of color, suit and value, and thus has 52 parts of size one each (such as $red \wedge \heartsuit \wedge king$), but the constraints only mention color (e.g., same color) and $queen$. In this case, it is sufficient to consider the partition based on the properties $black \wedge queen$ (two cards), $black \wedge \neg queen$ (24 cards), $red \wedge queen$ (two cards), and $red \wedge \neg queen$ (24 cards). In terms of the Bayesian network, this reduces the number of possible values (and thus the size of the CPT) of a child node that takes a single card from the deck from 52 to four. Our solver therefore collects for each root node all properties appearing in questions or observations on the node or its descendants, computes the smallest partition that allows to check those properties, and uses this partition instead of the maximal one.

Second, as only the parts of the model that satisfy the evidence are relevant to answer conditional queries, we insert tests for observation constraints on individual parent nodes directly below these nodes. This is similar to the inline observe statements offered by some probabilistic programming languages [Gordon *et al.*, 2014], and allows the solver to identify parts of the model that, while relevant to the unconditional query, can be safely ignored to answer the conditional query, which often further speeds up inference. For instance, in the case of Q2, Figure 2 shows on the right the possible values of all random variables relevant to answer questions about snd, using the Bayesian network as shown on the left. As the top two values for $first$ violate the observation that the first marble is red, the corresponding values for $\mathrm{rest}(first)$ and snd are irrelevant. By inserting this constraint directly below node *first*, we make this information available to the inference engine.

4 Natural Language Processing

The goal of the NLP component is to convert the natural language description of the problem into the formal description outlined in the previous section. In this initial approach, we focused on problems that explicitly mention groups of objects and only contain a single action. We also combine the matching of handcrafted templates with a simple neural network for learning the role of the numbers occurring in the questions.

Extraction and classification of numbers. In the first step, we use Stanford CoreNLP[1] to generate multiple, alternative parse trees for each of the sentences in the problem.

We then find all numbers in the problem based on Part-Of-Speech tags. We also include some phrases that describe an unspecified amount such as "the rest", "the others", etc. and we handle compound statements involving numbers such as "X percent". For each number, we need to determine two things:

1. Which container size, if any, does it specify? That is, does it provide the size of the root multiset, the size of a partition, the size of a selected set, or some other number. This classification is based on a neural-network classifier (using scikit-learn's[2] MLPClassifier) trained on 200 randomly selected examples. As features, we use 45 features that describe the structure of the parse tree around the number (see Table 1 for a summary of these features).
2. A textual description of the properties of the objects in the container, see the text in Table 2. This description is obtained by selecting a relevant subtree of the parse tree based on a set of hand-crafted rules, based on the same 200 examples.

Additionally, if it is determined that a number refers to the size of a partition or a selected set, we also extract a fragment describing the parent multiset. Figures 3 and 4 show examples of such extraction rules. For numbers that specify the size of a selection, we also determine whether the selection is with or without replacement. Table 2 shows an example of the output of this stage.

In this phase, we process each sentence individually. We apply the classifier model and extraction rules to each of the multiple alternative parse trees for the sentence being processed. For each parse tree we compute a confidence score which is the average classification probability of the neural network for the numbers in the parse tree, that is, a parse tree on which the neural network can make more confident predictions gets a higher score. After this phase, we only keep the parse tree for each sentence with the highest score.

Find and parse the question. Next, we extract the question from the problem by looking for the occurrences of the word "probability" (or synonyms) in a question or imperative context (e.g. "What are the odds that ...", "Find the probability of ..."). We then parse the question into a structured form using a set of handcrafted rules. This structure can contain quantifiers (e.g. "exactly 3") and logical operators.

[1] http://stanfordnlp.github.io/CoreNLP/

[2] http://scikit-learn.org/

Table 1: Summary of binary features used for classification.

- There is an ancestor of type VP of Number in the parse tree; ... of type PP ...; ... of type PP with preposition *of* ...; ... with preposition *in*...; ... with preposition *with* ...; ... with preposition *from* ...; ...
- There is an ancestor A of Number in the parse tree that has a sibling mentioning the word *and* (indicating that the number occurs in a list);
- The Number in the parse tree is the subject of a VP containing a PP with preposition *of*; ... containing a PP with preposition *in*; ... *with*; ... *from*;
- the Number occurs before a colon; ... after a colon.
- The Number occurs near the word *replacement*; ... near the word *random*; ...
- The Number occurs in a question;
- The Number is a relative number;
- The Number is preceded by a quantifier (e.g. more than).

Table 2: Number extraction for "A jar contains 1 white, 2 red and 3 black marbles. One selects with replacement 3 marbles. Find the probability that precisely one of those marbles is white."

position	word	class	text	parent
1-4	1	part	white	a jar
1-7	2	part	red	a jar
1-10	3	part	black marbles	a jar
2-1	one	ignore	-	-
2-5	3	take (wr)	3 marbles	unknown
3-6	one	ignore	-	-

Post-processing. In the last step, we post-process the model to remove inconsistencies. In this phase, we match

- the properties in the partition descriptions and the questions;
- the multiset identifiers in the setup, actions and questions; and
- we add missing information such as implicit actions (e.g. "what is the probability that a student ..." implies we take one student from the root set).

5 Evaluation

Solver. In order to evaluate our system, we hired three students to collect and label probability problems from textbooks and online sources. This has resulted in 2376 probability-related problem descriptions. For 2160 (90.9%) of these examples, we could derive a formal model (either automatically from the labeling, or manually for 70 examples which could not be labelled in our labelling system). The other examples either require advanced mathematical knowledge or contain constraints that we currently do not support.

Of the 2160, our solver could solve 2106 correctly within a time limit of 60 seconds per task. The remaining 54 did not complete within this limit. All cases that reach the time limit have outcome spaces which are too large for the current solver. Examples of such questions include "What is the

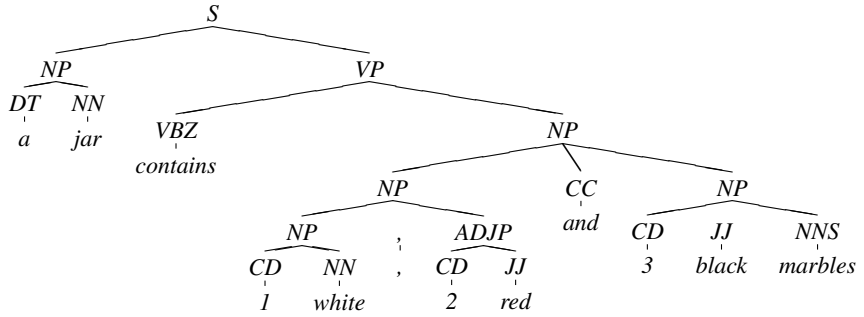

Figure 3: Parse tree for an example sentence. The numbers are classified as partition sizes because they occur in a list. This list is part of a verb phrase, so its subject is taken as the parent of this set.

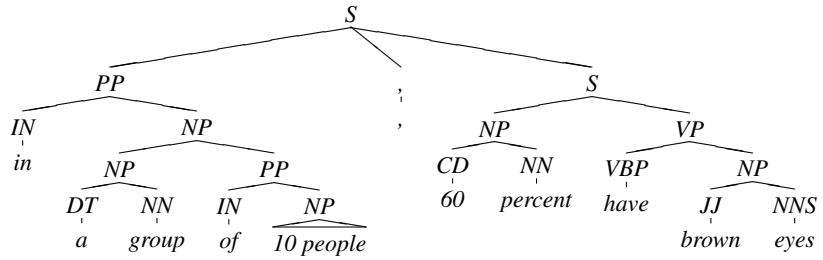

Figure 4: Parse tree with prepositional phrases. The number "60 percent" is the subject, its description is the VP attached to it ("have brown eyes"), and its parent is the PP preceding it ("in a group of 10 people").

probability of all digits in a seven digit phone number being different?", where the question refers to a sequence container with 10^7 possible contents, of which $604\,800$ satisfy the constraint, or "What is the probability that the number of 2s or 3s seen in 240 rolls of a fair die is between 75 and 83, inclusive?", where after reducing the root partition to $2 \vee 3$ and $\neg(2 \vee 3)$ the step-based implementation of take actions faces a tree of depth 240 and branching factor two. Clearly, such examples require alternative, more abstract implementations of take actions that exploit exchangeability with respect to the order of drawing objects, e.g., based on the counting strategies humans use to solve such tasks.

Language + Solver. We also verified how many example could be solved directly from the English text. The NLP extractor only supports a subset of the complete formal language. For example, we excluded problems that are based on events (e.g., coin tosses), require observations (e.g., Q2 in Figure 1), or that have aggregate or sequence constraints. Because of this, only 869 out of the 2106 examples are supported based on the problem structure or the constraints present in them. Of these remaining examples, 31.1% (270, or 12.5% of the total set of examples) were solved correctly, 138 produce a wrong result, and the rest did not produce any result (because of an incomplete model). These number include the 200 training examples (of which 88 where solved correctly).

It must be noted that the language in the examples was taken directly, without any further processing, from textbooks and online sources. We performed a qualitative analysis on a sample of the questions that could not be solved. Common issues are the lack of background information, words that could not be matched (e.g., "male" vs "men"), alternative phrasing (e.g., "there are 13 of each suit"), and the use of constants and enumeration. For example, in the question "John, Kevin, Larry, Mary and Nancy all volunteered to so [*sic*] some math tutoring. If their teacher randomly chooses two of the five students, what is the probability of selecting the two girls?", it is impossible to determine that Mary and Nancy are girls without extensive background knowledge.

The NLP component can clearly be improved, but it should also be clear that probabilistic word problems pose some interesting challenges from an NLP point of view.

6 Conclusions

Mathematical and scientific problem solving has a long and distinguished history in AI and cognitive science. In this paper, we developed an end-to-end fully automated approach that provides answers to exercises about probability formulated in natural language.

We provided a declarative language to encode probability word problems, and a corresponding solver implemented as a probabilistic program, which is a natural and emerging framework for inference problems described on an abstract level. While we restricted ourselves to a fragment of the natural language, we intend to explore richer subsets of natural language in the near future. We are also considering further refinements of the modeling language as well as the implementation of alternative probabilistic primitives in the solver to capture an even wider range of problems.

Acknowledgments

Anton Dries and Angelika Kimmig contributed equally to this paper. This work was supported through an IBM Faculty Award and by the Flemish Research Foundation (FWO-Vlaanderen). The authors thank Liselot Beckwé, Monica Brocca and Hannah Davidoff for their efforts in collecting and labelling the data, and Hendrik Blockeel, Wannes Meert and Guy Van den Broeck for inspiring discussions and suggestions.

References

[Alvin *et al.*, 2014] Chris Alvin, Sumit Gulwani, Rupak Majumdar, and Supratik Mukhopadhyay. Synthesis of geometry proof problems. In *AAAI*, pages 245–252, 2014.

[De Raedt and Kimmig, 2015] Luc De Raedt and Angelika Kimmig. Probabilistic (logic) programming concepts. *Machine Learning*, pages 1–43, 2015.

[De Raedt *et al.*, 2007] Luc De Raedt, Angelika Kimmig, and Hannu Toivonen. ProbLog: A probabilistic Prolog and its application in link discovery. In *IJCAI*, pages 2462–2467, 2007.

[De Raedt *et al.*, 2016] Luc De Raedt, Kristian Kersting, Sriraam Natarajan, and David Poole. Statistical relational artificial intelligence: Logic, probability, and computation. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 10(2):1–189, 2016.

[Gelb, 1971] Jack P Gelb. Experiments with a natural language problem-solving system. In *IJCAI*, pages 455–462, 1971.

[Getoor and Taskar, 2007] Lise Getoor and Ben Taskar. *Introduction to statistical relational learning*. MIT press, 2007.

[Gordon *et al.*, 2014] Andrew D Gordon, Thomas A Henzinger, Aditya V Nori, and Sriram K Rajamani. Probabilistic programming. In *Proceedings of the on Future of Software Engineering*, pages 167–181. ACM, 2014.

[Hernández-Orallo *et al.*, 2016] José Hernández-Orallo, Fernando Martínez-Plumed, Ute Schmid, Michael Siebers, and David L Dowe. Computer models solving intelligence test problems: Progress and implications. *Artificial Intelligence*, 230:74–107, 2016.

[Hosseini *et al.*, 2014] Mohammad Javad Hosseini, Hannaneh Hajishirzi, Oren Etzioni, and Nate Kushman. Learning to solve arithmetic word problems with verb categorization. In *EMNLP 2014*, pages 523–533, 2014.

[Kushman *et al.*, 2014] Nate Kushman, Yoav Artzi, Luke Zettlemoyer, and Regina Barzilay. Learning to automatically solve algebra word problems. *ACL (1)*, pages 271–281, 2014.

[Mukherjee and Garain, 2008] Anirban Mukherjee and Utpal Garain. A review of methods for automatic understanding of natural language mathematical problems. *Artificial Intelligence Review*, 29(2):93–122, 2008.

[Niepert and Van den Broeck, 2014] Mathias Niepert and Guy Van den Broeck. Tractability through exchangeability: A new perspective on efficient probabilistic inference. In *AAAI*, 2014.

[Poole, 2003] David Poole. First-order probabilistic inference. *IJCAI*, 3:985–991, 2003.

[Seo *et al.*, 2014] Min Joon Seo, Hannaneh Hajishirzi, Ali Farhadi, and Oren Etzioni. Diagram understanding in geometry questions. In *AAAI*, 2014.

[Seo *et al.*, 2015] Minjoon Seo, Hannaneh Hajishirzi, Ali Farhadi, Oren Etzioni, and Clint Malcolm. Solving geometry problems: Combining text and diagram interpretation. *EMNLP*, 2015.