Confidences for Commonsense Reasoning

Tanel Tammet¹(✉) , Dirk Draheim² , and Priit Järv¹

¹ Applied Artificial Intelligence Group,
Tallinn University of Technology, Tallinn, Estonia
{`tanel.tammet,priit.jarv1`}@taltech.ee

² Information Systems Group, Tallinn University of Technology, Tallinn, Estonia
`dirk.draheim@taltech.ee`

Abstract. Commonsense reasoning has long been considered one of the holy grails of artificial intelligence. Our goal is to develop a logic-based component for hybrid – machine learning plus logic – commonsense question answering systems. A critical feature for the component is estimating the confidence in the statements derived from knowledge bases containing uncertain contrary and supporting evidence obtained from different sources. Instead of computing exact probabilities or designing a new calculus we focus on extending the methods and algorithms used by the existing automated reasoners for full classical first-order logic. The paper presents the CONFER framework and implementation for confidence estimation of derived answers.

1 Introduction

The mainstream approaches for “commonsense reasoning” (CSR) before this century focused on rule based reasoning and building suitable logical systems. During the last ten years the focus has switched to machine learning and neural networks. Both of these approaches appear to be limited. A promising approach to practical question answering is building hybrid systems like Watson [17] which complement the current machine learning systems for natural language with logic-based reasoning systems specialized for CSR. In particular, hybrid systems have a good potential for progress towards explainable A.I. See Marcus [26] for an overview of the current work in the area. Our goal is to build upon the existing theory and reasoning systems for first order logic (FOL) to develop a framework and practical systems using FOL reasoners which could be incorporated into a hybrid system containing both machine learning components and rule-based reasoning components. This approach will also provide step-by-step proofs for the answers found, useful for building explainable systems.

We will present the design and implementation of the CONFER framework for extending existing automated reasoning systems with confidence calculation capabilities. We will not focus on other, arguably even more critical issues for CSR and question answering, like handling natural language itself, dialogues,

rules with exceptions and default logic [31] or circumscription, knowledge representation for space/time, epistemic reasoning, using context, building and collecting suitable rules, machine learning etc.

The specific CSR task targeted by the current paper is *question answering*: given either a knowledge base of facts and rules or a large corpora of texts (or both), plus optionally a situation description (assumptions) for the questions, answer questions posed either in logic or natural language.

Historically, the longest-going CSR project has been the logic-based CYC project [25], already in 1985 stating the focus on CSR. Despite several successes, the approach taken in the CYC project has often been viewed as problematic ([8], [10]) and has been repeatedly used as an argument against logic-based methods in CSR. Beltagy et al [5] experiment with Markov Logic Network for combining logical and distributional representations of natural language meaning. Domingos et al note in [13] that the CYC project has used Markov Logic for making a part of their knowledge base probabilistic. Khot et al [24] experiment with Markov Logic Networks for NLP question answering. Furbach et al [20] describe research and experiments with a system for natural language question answering, converting natural language sentences to logic and then performing proof search, using different existing FOL knowledge bases. The authors note a number of difficulties, with the most crucial being the lack of sufficiently rich FOL knowledge bases. The closest current approach to ours appears to be the Braid system [23] built by the team previously involved with the Watson system.

2 Interpretation and Encoding of Uncertainty

Reasoning under uncertainty has been thoroughly investigated for at least a century, leading to a proliferation of different theories and mechanisms. A classic example is the MYCIN system [6]. For newer approaches see, for example, [32] and [9]. Each of these is well suited for certain kinds of problems and ill-suited for other kinds. Underlying this is the philosophical complexity of interpreting probability: see [22] for an overview, see also [16], pp. 5-7.

Most of the previous work on combining logic with uncertainty has targeted propositional logic. First order logic is then handled by creating a finite set of weighted ground instances of formulas. This is the approach taken, for example, by the probabilistic logic programming systems ProbLog2 [18], PRISM [34] and the implementation of Markov Logic Networks [12,11] by the Alchemy 2 system [1]. These systems pose different restrictions to the FOL formulas and while well-suited for small domains in cases the restrictions can be followed, the approach becomes unfeasible if the domain is large or formulas complex. For example, neither the ProbLog 2 nor Alchemy 2 implementations manage to answer queries like $1.0::p(a) . 1.0::p(i(a,b)) . 1.0::p(Y) :- p(X), p(i(X,Y)) . query(p(b))$. The implementation of ProbLog2 [29] fails, presumably due to infinite recursion in searching for possible groundings for the variables, while Alchemy 2 does not allow function terms in grounded facts.

Previous approaches to full first order logic tend to fall into one of the three camps: either using fuzzy logic [41], representing probabilities as an interval (see [15] for the axiomatic derivation of Dempster-Schafer rules) or interpreting probabilities via many worlds similarly to modalities [4].

For the sake of this work, we largely follow the *subjective interpretation* of probability as a degree of belief, originating from Ramsey and De Finetti. We use the word *confidence* to denote our rough adherence to this interpretation. We avoid using complex measures such as intervals, distributions or fuzzy functions.

In the context of question answering we assume that confidences are typically used for sorting a list of candidate answers by their calculated confidence and optionally applying a filter to eliminate answers with a confidence under a certain threshold. Answers provided may be also annotated with a confidence number. If we are given or can calculate several different confidences for the same answer, we always prefer the higher confidence. The question of calculating a correct probability rarely arises or is considered to be unfeasible.

2.1 Sources, Representation and Meaning of Statements, Confidences and Dependencies

We assume that the confidence in a fact or rule in our common sense knowledge base (KB in the following) typically arises from a large number of human users via crowd-sourcing like in ConceptNet [35,7], NLP-analyzed scraped text from the web like NELL [27], and/or combining different knowledge bases with weights like in [14] and [7] or assigned to the equivalence of name pairs in the vocabulary like in [28] and [19]. There is recent progress towards making knowledge bases for common sense reasoning where the relation strengths (typicality, saliency) have been empirically evaluated [7,33].

To each FOL statement S we will assign both a confidence c and a set L of unique identifiers of (non-derived) input statements used for deriving this statement: a triple $\langle S, c, L \rangle$. Lists of such triples are then treated as sets. The dependency lists L are used in the formula estimating the cumulated confidence. The algorithm for calculating confidences c for derivations will be presented later.

To be more exact, we will not allow assigning confidences to arbitrary statements. Instead, we will assume that the FOL statements are converted to a conjunctive normal form: a conjunction of Skolemized disjunctions, where each disjunction only consists of atomic statements (a predicate applied to arguments) or negations of atomic statements. Such disjunctions are called *clauses*. We will not allow nested triples, i.e. S is always a pure FOL clause not containing any confidence or dependency information usable by the presented algorithms. However, for each single FOL clause S there may be many different derivable triples $\langle S, c, L \rangle$ for different c and L , stemming from different derivation trees of S . They are assumed to be independent statements, possibly allowing the calculation of the cumulative confidence for S higher than $\max(c, c')$ where c and c' come from different triples.

A KB may contain logical contradictions and identical FOL clauses with different confidences given by different sources. For example, the following is a logically contradictory KB containing several copies of the same clause with different confidences. The CONFER algorithm presented later gives us the confidence of $bird(a) : 0.682$ from this KB:

$$\langle bird(X), 0.1, L_1 \rangle, \langle bird(a), 0.8, L_2 \rangle, \langle bird(a), 0.9, L_3 \rangle, \langle \neg bird(a), 0.3, L_4 \rangle$$

We interpret the confidence as estimating the lower limit of the probability of a statement, i.e., $\langle S, c, L \rangle$ is interpreted as “statements L support the claim that $probability(S) \geq c$ ”. Thus two different confidence statements for the same clause are never contradictory, even if given by the same source.

3 The CONFER Extension Framework for CSR

In the following we will present the CONFER framework of extensions to the mainstream resolution-based search methods. We expect that the same framework can be adapted to search methods different from resolution, i.e. the specific aspects of resolution are not relevant for the main principles of the approach.

The intuition behind CONFER is preserving first order classical logic (FOL) intact as an underlying machinery for derivations in CSR. The core methods of automated reasoning used by most of the high-performance automated reasoning systems remain usable as core methods for CSR. Essentially, FOL with the resolution method produces all combinations of derivable sentences (modulo simplifications like subsumption) which could lead to a proof. The main difference between strict FOL and CONFER extensions is in the handling of constructed proof trees: the outcome of a CONFER reasoner is a set of combined FOL proofs with the confidence measures added.

Importantly, the framework does not generally calculate the exact maximal confidence for derived statements, since this is, in nontrivial cases, either impossible or unfeasible. Our goal is to give a practically useful estimation of the maximal confidence without causing a large overhead on the FOL proof search and avoiding combinatorial explosion while calculating the confidences.

3.1 Resolution Method

In the following we will assume that the underlying first order reasoner uses the resolution method, see [3] for details. The rest of the paper assumes familiarity with the basic concepts, terminology and algorithms of the resolution method.

3.2 Queries and Answers

We assume the question posed is in one of two forms: (1) Is the statement Q true? (2) Find values V for existentially bound variables in Q so that Q is true. For simplicity's sake we will assume that the statement Q is in the prefix form, i.e., no quantifiers occur in the scope of other logical connectives.

In the second case, it could be that several different value vectors can be assigned to the variables, essentially giving different answers. We also note that an answer could be a disjunction, giving possible options instead of a single definite answer. However, as shown in [38], in case a single definite answer exists, it will be derived eventually.

A widely used machinery in resolution-based theorem provers for extracting values of existentially bound variables in Q is to use a special *answer predicate*, converting a question statement Q to a formula

$$\exists X_1, \dots, \exists X_n (Q(X_1, \dots, X_n) \& \neg \text{answer}(X_1, \dots, X_n))$$

for existentially quantified variables in Q [21]. Whenever a clause is derived which consists of only answer predicates, it is treated as a contradiction (essentially, answer) and the arguments of the answer predicate are returned as the values looked for. A common convention is to call such clauses *answer clauses*. We will require that the proof search does not stop whenever an answer clause is found, but will continue to look for new answer clauses until a predetermined time limit is reached. See [37] for a framework of extracting multiple answers.

We also assume that queries take a general form $(KB \& A) \Rightarrow Q$ where KB is a commonsense knowledge base, A is an optional set of precondition statements for this particular question and Q is a question statement.

Since we assume the use of the resolution method for proof search, the whole general query form is negated and converted to clauses, i.e., disjunctions of literals (positive or negative atoms). We will call the clauses stemming from the question statement *question clauses*.

3.3 Top Level of the Algorithm

Calculating confidences for question answering requires, at least, the ability to calculate (a) the decreasing confidence of a conjunction of clauses as performed by the resolution and paramodulation rule, (b) the increasing confidence of a disjunction of clauses for cumulating evidence, (c) the decreasing confidence of considering negative evidence for a clause.

While the systems based on, say, Bayes networks and Markov logic, perform these operations in a combined manner, our framework will split the whole search into separate phases for each. First we perform a modified resolution search we call *c-resolution* calculating the decreasing confidence and potentially giving a large number of different answers and proofs. Next we will combine the different proofs using the cumulation operation. Finally we will collect negative evidence for all the answers obtained so far, separately for each individual answer. The latter search is also split into the *c-resolution* phase and the cumulating phase. Since we assume the use of full FOL, the *c-resolution* search will not necessarily terminate, thus we will use a time limit. The top level of the algorithm is presented in the following section as Algorithm 1.

Algorithm 1 CONFER algorithm**Input:** Common sense knowledge base KB, question Q , time limit t .**Output:** Set of answers R with attached confidences.

- 1: Let $R = \{\}$.
- 2: Find a set of initial positive answers with confidences and dependencies $IPA = \{\langle A_1, c_1, L_1 \rangle, \dots, \langle A_p, c_p, L_p \rangle\}$ for Q from KB using c-resolution with the time limit $t/2$.
- 3: Calculate a set of cumulative positive answers $CPA = \{\langle B_1, d_1, E_1 \rangle, \dots, \langle B_r, d_r, E_r \rangle\}$ from IPA.
- 4: Let $i = 1$.
- 5: **while** $i \leq r$ **do**
- 6: Form the negated question NQ_i from $\neg Q$ with a substitution s given by B_i
- 7: Find a set of initial negative answers N_i with confidences and dependencies for NQ_i from KB using c-resolution with the time limit $t/(2 * r)$.
- 8: **if** N_i is empty **then**
- 9: Let $nc_i = 0$.
- 10: **else**
- 11: Calculate the cumulative negative confidence nc_i from N_i .
- 12: **end if**
- 13: Add a pair $\langle B_i, (c_i - nc_i) \rangle$ to R .
- 14: Let $i = i + 1$.
- 15: **end while**
- 16: For each pair $\langle B_i, d_i \rangle, \langle B_j, d_j \rangle$ in R where $i \neq j$, $B_i = B_j$ and $d_i > d_j$, remove $B_j : d_j$ from R .
- 17: Remove from R all elements $\langle B_i, d_i \rangle$ where $d_i \leq 0$.
- 18: **return** the set of answers with confidences R .

3.4 C-Resolution

The core part of the algorithm described above is c-resolution: a relatively simple modification of the resolution method calculating and keeping track of the (multiplied) confidences of premisses of each step along with the union of their dependencies.

Definition 1 (C-Resolution). *A modification of the resolution method computing an ever-increasing set of different proofs for different answers (substitutions to the question clauses) while employing the relevance filter (definition 2), performing basic confidence calculation for resolution steps (definition 3), assigning the union of the dependency lists of premisses to each derived clause, restricting subsumption to c-subsumption (definition 5) and restricting simplification steps according to c-subsumption.*

Inconsistencies. A KB with a nontrivial structure may contain inconsistencies in the sense that a contradiction can be derived from the KB. Looking at existing KBs mentioned earlier, we observe that they either are already inconsistent (for example, the largest FOL version of OpenCyc [30] in TPTP [40] is inconsistent) or would become inconsistent in case intuitively valid inequalities are added,

for example, inequalities of classes such as “a cat is not a dog”, “a male is not a female” or default rules such as “birds can fly”, “dead birds cannot fly”, “penguins cannot fly”. We note that several large existing KBs do not contain such inequalities explicitly, although they are necessary for nontrivial question answering under the open-world assumption.

Since classical FOL allows to derive anything from a contradiction, it is clearly unsuitable for a large subset of KB-s. Two possible ways of overcoming this issue are: (a) using some version of relevance logic or other paraconsistent logics or (b) defining a filter for eliminating irrelevant classical proofs. We argue that despite a lot of theoretical work in the area, only little work has been done in automated proving for relevance logic, thus using it directly is likely to create significant complexities. Instead, we introduce a simple relevance filter:

Definition 2 (Relevance Filter). *Each resolution derivation of a contradiction not containing any answer clauses is discarded.*

Since a standard resolution derivation of a contradiction does not lead to any further derivations, this filter is completeness-preserving in the sense that all resolution derivations containing an answer clause are still found.

Confidences of Derived Clauses. We take the approach of (a) providing a simple sensible baseline algorithm for calculating confidences of derived clauses, and (b) leaving open ways to modify this algorithm for specific cases as need arises. We will use a single rational number in the range 0...1 as a measure of a confidence of a clause, with 1 standing for perfect confidence and 0 standing for no information. Confidence of an atomic clause not holding is represented as a confidence of the negation of the clause.

As a baseline we use the standard approach of computing uncertainties of clauses derived from independent parent clauses A and B as:

$$P(A \wedge B) = P(A) * P(B)$$

Notice that for dependent parent clauses this formula *under-estimates* the confidence of the result.

Definition 3 (Basic Confidence Calculation for Resolution Steps). *For binary resolution and paramodulation steps, the confidence of a result is obtained by multiplying the confidences of the premises. For the factorization step, the confidence of the result is the confidence of the premise, unchanged. Question clauses have a confidence 1.*

A simple example employing forward reasoning (concretely, *negative ordered resolution*):

```
0.8:: bird(tweety).
0.9:: bird(X) => canfly(X).
0.7:: canfly(X) => fast(X).
1.0:: fast(X) => answer(X).
```

leads to a sequential derivation of

```
0.72:: canfly(tweety).
0.504:: fast(tweety).
0.504:: answer(tweety).
```

Recall that the confidences are assumed to be lower bounds of probabilities. Notice that the possible dependence of the premises could be taken into account, as in the following section for cumulative evidence. This would result in higher confidence numbers for derivations with dependent premises. Consider the following example:

```
0.9:: bird(X) => canfly(X).
0.1:: -bird(X) => canfly(X).
```

Using the basic calculation step we can derive that anything can fly:
 0.09:: canfly(X). However, since anything is either a bird or is not a bird, the confidence of canfly(X) should be at least 0.1, and possibly higher, depending on the ratio of birds to non-birds.

Generally, we can use the minimization operation leading to a higher confidence value than the multiplication of the confidences of premises in the following special case. The standard resolution inference rule used by a large class of automated reasoners is defined as

$$\frac{A_1 \vee A_2 \vee \dots \vee A_n \quad \neg B_1 \vee B_2 \vee \dots \vee B_m}{(A_2 \vee \dots \vee A_n \vee B_2 \vee \dots \vee B_m)\sigma}$$

where σ is the most general unifier of A_1 and B_1 . A clause A *subsumes* a clause B if the literals of $A\delta$ are a subset of literals of B for some substitution δ .

Definition 4 (Extended Confidence Calculation for Resolution Steps). *If $(A_2 \vee \dots \vee A_n)\sigma$ subsumes $(B_2 \vee \dots \vee B_m)\sigma$ in the resolution inference defined above then the confidence of the result is the minimum of the confidences of premises.*

C-Subsumption and Simplifications. Since standard subsumption used by resolution provers to clean up search space may remove clauses with a higher confidence or fewer dependencies than the subsuming clause, it may cause the prover to lose derivations potentially leading to a higher confidence. Thus we use *c-subsumption* instead of the standard subsumption:

Definition 5 (C-Subsumption). *A triple $T_1 = \langle A_1, c_1, L_1 \rangle$ consisting of a clause A_1 , confidence c_1 and a dependency list L_1 c-subsumes a triple $T_2 = \langle A_2, c_2, L_2 \rangle$ if and only if A_1 subsumes A_2 , $c_1 \geq c_2$ and $L_1 \subseteq L_2$.*

We can prove the following lemma:

Lemma 1 (C-Subsumption Preserves Completeness). *When a c-resolution proof can be found without using subsumption, it can be also found with c-subsumption.*

The proof holds for strategies of resolution for which standard subsumption is complete for ordinary proof search without confidences.

We restrict the simplification operations like demodulation and subsuming resolution accordingly: a derivation step must keep the original premiss P if the result has a lower confidence or a longer list of dependencies than P .

3.5 Cumulative Confidence

We will now look at the situation with additional evidence for the derived answer. In our context, using additional evidence is possible if a clause C can be derived in different ways, giving two different derivations d_1 and d_2 with confidences c_1 and c_2 . In case the derivations d_1 and d_2 are independent, we could apply the standard formula

$$P(A \vee B) = P(A) + P(B) - P(A \wedge B)$$

to c_1 and c_2 to calculate the cumulative confidence for C .

What would it mean for derivations to be “independent”? In the context of commonsense reasoning we cannot expect to have an exact measure of independence. However, suppose the derivations d_1 and d_2 consist of exactly the same initial clauses, but used in a different order. In this case $c_1 = c_2$ and the cumulative confidence should intuitively be also just c_1 : no additional evidence is provided. On the other hand, in case that the non-question input clauses of d_1 and d_2 are mutually disjoint, then the derivations are also independent (assuming all the input clauses are mutually independent), and we should apply the previous rule for $P(A \vee B)$ for computing the cumulative confidence.

We will estimate the independence i of two derivations d_1 and d_2 simply as

$$1 - \frac{\text{number of shared input clauses of } d_1 \text{ and } d_2}{\text{total number of input clauses in } d_1 \text{ and } d_2} \quad (1)$$

Thus, if no clauses are shared between d_1 and d_2 , then $i = 1$ and if all the clauses are shared, then $i = 0$.

In addition, we also know that it is highly unlikely that all the input clauses are mutually independent. Again, lacking a realistic way to calculate the dependencies, we give a heuristic estimate h in the range 0...1 to the overall independence of the input clause set, where 1 stands for total independence and 0 for total dependence.

Finally, we will calculate the overall independence of two derivations d_1 and d_2 as $i * h$. Next, we will postulate a heuristic rule for the combination of these two independence measures as follows.

Definition 6 (Confidence Calculation for Cumulative Evidence). *Given two derivations d_1 and d_2 of the search result C with confidences c_1 and c_2 , calculate the updated confidence of C as*

$$\max(c_1 + c_2 * i * h, \quad c_1 * i * h + c_2) - c_1 * c_2 * i * h$$

where

- independence of derivations i is defined as 1 above,
- h is the heuristic estimate of the independence of the total set of input clauses from 1 for total independence to 0 for total dependence.

The formula satisfies the following *intuitive requirements for cumulative evidence*:

- If d_1 and d_2 do not share non-question input clauses and all the input clauses are mutually independent, $i * h = 1$ and the formula turns into $c_1 + c_2 - (c_1 * c_2)$.
- If d_1 and d_2 have the same non-question input clauses or the total set of input clauses is mutually totally dependent, $i * h = 0$ and the formula turns into $\max(c_1, c_2)$.

3.6 Negative Evidence

Recall the standard mechanism employed in FOL provers for finding concrete answers: transforming existentially quantified goal clauses to clauses containing a special **answer** predicate and treating clauses containing only **answer** predicates as actual answers to the question found.

Once negation is present, the reasoning system using the CONFER framework has to attempt to find both positive and negative evidence for any potential answer. This cannot be easily done in a single proof search run.

Observe that giving a general search question containing variables like $\text{bird}(X) \vee \text{answer}(X)$ may produce a different set of answers than the positive question $\neg \text{bird}(X) \vee \text{answer}(X)$. Also observe that the potential set of answers may be huge for both positive and negative answers: in a large KB there may be millions of statements about birds and our reasoning system will be able to derive only a small fraction of potential answers in any given time slot. Thus, even if negative evidence is potentially derivable for some positive answer, the system is unlikely to find it.

A reasonable solution to this problem is to run the searches for negative evidence only for the concrete instances of positive answers found. More concretely, we conduct additional proof search for the negations of two types of questions Q : (a) If Q contains no existentially quantified variables, is the statement $\neg Q$ true? (b) For all i vectors of values $C_1i, \dots, C_ni$ found for existentially bound variables $X_1, \dots, X_n$ in Q making Q true, is $\neg Q$ true when we substitute the values in $C_1i, \dots, C_ni$ for corresponding variables in Q ?. The final confidence of an answer to Q is calculated by subtracting from the confidence of the positive answer the confidence of the answer to the corresponding negated instance of the question.

Using negative evidence may lead to unexpected results. Consider the following trivial example in the ProbLog syntax:

```
0.5::bird(a). 0.5::not bird(a). query(bird(a)).
```

CONFER gives us confidence 0, which we interpret as “no information”, not as “false”. However, ProbLog2 gives confidence 0.25, which is explained by one of

the authors in private correspondence thus: an atom (head) is satisfied if any of the rules that make it true fire and none of the rules that make it false fire. In this example ProbLog2 gets $0.5 * (1 - 0.5) = 0.25$. On the other hand, the three different algorithms of the Alchemy 2 system – MC-SAT explained in [12], exact and approximate probabilistic theorem proving explained in [11] – give answers 0.015, 0 and 0.082, respectively. To be concrete, we are using the Alchemy 2 versions from [2]. For this and the following Alchemy 2 examples we prepared an MLN file with no weights and a training data file with some generated facts for each example. Then we ran the *learnwts* program with default parameters, which created the MLN file with weights for each example.

Next, consider a previous example augmented with the “birds fly” rule:

```
0.5::bird(a). 0.5::not bird(a). 0.9::flies(X) :- bird(X).
query(flies(_)).
```

Here CONFER gives us 0.45, which is inconsistent with the result of the previous example. ProbLog2, on the other hand, gives 0.225, which is unintuitive, but consistent with the unintuitive result of ProbLog2 in the previous example. The three algorithms of Alchemy 2 mentioned above give us 0.047, 0 and 0.98. The issue arising in this example is similar to nonmonotonic reasoning like default logic: adding negative evidence to being a bird should block previously derivable facts. We know that since FOL is not decidable, such checks would make derivation steps generally not computable. As a final twist to the example we augment the ruleset by giving more details about the distribution:

```
0.5::bird(a). 0.5::not bird(a). 0.9::flies(X) :- bird(X).
0.1::not flies(X) :- bird(X).
%% 0.1::flies(X) :- not bird(X). %% commented out
0.9::not flies(X) :- not bird(X).
query(flies(_)).
```

Here CONFER gives us an acceptable 0.014 (positive evidence 0.490 and negative evidence 0.476), while ProbLog2 gives 0.2025. The results of Alchemy 2 are 0.047, 0 and 0.976. Adding the rule we have commented out makes CONFER to give -0.008 while ProbLog2 complains that the example is not acceptable. Alchemy 2 gives us 0.056, 0 and 0.509.

4 Implementation and Experimental Results

The first author has implemented the CONFER framework as an extended version of his high-performance open-source automated reasoning system gkc [39] for FOL, performing fairly well in the yearly CASC competition for automated reasoners [36], see <http://www.tptp.org/CASC/>. The implementation is written in C like gkc. The compiled executable can be downloaded from <http://logictools.org/confer/> along with a number of examples.

Several algorithms, strategies and optimizations present in the gkc system are currently switched off, due to the need for additional modifications and testing.

In particular, parallel processing is switched off, as well as the crucial algorithms for selecting a list of suitable search strategies and performing search by batches with iteratively increasing time limits.

Importantly, we have not yet implemented any specialized strategies for using the attached confidences and dependencies for directing and optimizing search. It is clear that the added information gives ample potential opportunities for directing the search.

We will give an overview of the experiments with the implementation in two sections. First we will look at the confidences calculated and compare these, where possible, with the values given by ProbLog2 and Alchemy 2. Next we will look at the performance of the system on nontrivial problems.

The inputs and outputs for the CONFER implementation and the systems compared to are given on the web page <http://logictools.org/confer/>. The set of examples given contains over 30 case studies and can be run using the command-line implementation provided on the same web page as a single executable file. The implementation is self-contained, not dependent on other systems or external libraries. It should run on any 64-bit Linux system.

4.1 Comparing Confidences

We will compare the confidences calculated by CONFER on small selected examples with these of ProbLog2 and Alchemy 2. The first two are presented in the ProbLog2 tutorial. When CONFER can perform neither cumulation nor collection of evidence, the values calculated are the same as of ProbLog2. The cumulation operation of CONFER produces, as expected, slightly different values than ProbLog2 or Alchemy 2. For the following examples the overall independence estimate h is assigned 1 (maximum). Since the principles of handling negative evidence are fundamentally different between the two systems, this operation causes the most significant changes. It is worth noticing that more often than not, the results of ProbLog2 and Alchemy 2 also differ.

First, a simple version of the well-known social networks of smokers example in the ProbLog syntax. CONFER uses a different syntax, but the clauses and confidences given are exactly the same. We have also built the corresponding data- and rulesets for Alchemy 2, which uses a fairly different input method than CONFER or ProbLog.

```
0.8::stress(ann).          0.4::stress(bob).
0.6::influences(ann,bob).  0.2::influences(bob,carl).
smokes(X) :- stress(X).
smokes(X) :- influences(Y,X), smokes(Y).
query(smokes(carl)).
```

For this example, ProbLog2 gives an answer 0.1376 and CONFER gives 0.1201, cumulating values 0.096 and 0.08. The three different algorithms of Alchemy 2 – MC-SAT inference (see [12]), exact and approximate lifted inference explained in [11] – give 0.135, 0 and 0.741, respectively. In the following tables we

will refer to these three as *Alch i*, *Alch e* and *Alch a*. Removing the input clause `0.4::stress(bob)` also removes the cumulation possibility and both CONFER and ProbLog2 give 0.096 as an answer.

Next, the well-known earthquake example. CONFER performs both cumulation and collecting negative evidence.

```

person(john).      person(mary).
0.7::burglary.    0.2::earthquake.
0.9::alarm :- burglary, earthquake.
0.8::alarm :- burglary, \+earthquake.
0.1::alarm :- \+burglary, earthquake.
0.8::calls(X) :- alarm, person(X).
0.1::calls(X) :- \+alarm, person(X).
evidence(calls(john),true).
evidence(calls(mary),true).
query(burglary).
query(earthquake).

```

We will present the ProbLog2 and CONFER results with both the positive and negative evidence components (columns CONFER + and CONFER -) given by CONFER. Importantly, by default CONFER will try to find up to 10 different proofs: increasing or decreasing these limits has a noticeable effect on the results as well as running time.

query	CONFER	CONFER +	CONFER -	Problog	Alch i	Alch e	Alch a
burglary	0.8713	0.97650	0.1051	0.9819	0.709	0	0.905095
earthquake	0.1648	0.8854	0.7206	0.2268	0.204	0	0.888

Finally we bring the famous penguin example from default logic. We will formulate it using confidences instead of defaults. We state that penguins form a tiny subset of birds. The CONFER implementation collects both positive and negative evidence, but there are no cumulation possibilities.

```

1.0::bird(tweety).    1.0::penguin(pennie).
1.0:: bird(X) :- penguin(X).
0.001:: penguin(X) :- bird(X).
0.9:: flies(X) :- bird(X).
1.0:: not flies(X) :- penguin(X).
query(flies(_)).

```

query	CONFER	CONFER +	CONFER -	Problog	Alch i	Alch e	Alch a
flies(pennie)	-0.1	0.9	1.0	0	0.00001	0	0
flies(tweety)	0.899	0.9	0.001	0.8991	0.064	0	0.873

4.2 Performance

We will investigate the performance of our CONFER implementation on the following nontrivial example FOL problems from the TPTP collection [40]. Due to restrictions in the language or the principles of the search algorithm, ProbLog2 cannot handle any of these examples even if they are converted to clauses in ProbLog syntax. Thus we will compare the performance of the CONFER system on several modifications of the problems against the conventional FOL prover gkc used as a base for building the CONFER system.

The results are given for the following problems with the TPTP identifier and ratings: 0 means all the provers tested by the TPTP maintainers find a proof, 1 means no prover manages to find a proof. *Steamroller* (PUZ031+1, rating 0) is a puzzle without equality. *Dreadbury* (PUZ001+2.p, rating 0.23) is a puzzle using also equality. *Lukasiewicz* (LCL047-1.p, rating 0) is an example in logical calculi. Commonsense reasoning problems from CYC are taken from the largest consistent CYC version in TPTP: *CSR025+5*, *CSR035+5*, *CSR045+5*, *CSR055+5* (ratings 0.67, 0.83, 0.97, 0.87).

The CYC problems *CSR025+5* ... *CSR055+5* contain ca half a million formulae, but the proofs are relatively short. The first three problems are relatively small, but their proofs are significantly longer. The *Steamroller*, *Dreadbury* and the CYC *CSR035+5* problems have been augmented with a question asking for answer substitutions, while for the other CYC problems and the *Lukasiewicz* problems the conjectures do not contain the existence quantifier, thus we just try to prove these. For comparison purposes the CONFER proof searches are restricted to finding only the first answer (thus no cumulation is possible) and not collecting negative evidence.

We consider both the versions of problems with all clauses assigned a confidence between 0.6 ... 0.99 cyclically with a step 0.01 (column CONFER in the following table) and all the confidences assigned 1.0 (column CONFER 1.0). It is important to note that the CONFER system uses conventional subsumption and simplification for clauses with the confidence 1.0, i.e. in the “CONFER 1” column proof search is reduced to the ordinary resolution search. The gkc column gives the pure search time of the gkc prover used as a base for building the CONFER system, for the original TPTP versions (without a question of substitutions being asked). As a special case, variations 0 ... 4 of the *Lukasiewicz* problem are formed by attaching confidences below 1 to respectively 1 ... 4 input clauses and letting other confidences have value 1.0. (the *Lukasiewicz* problem consists of five clauses, one of these being the clause to be proved).

The columns CONFER ... “gkc pure” contain the pure proof search time in seconds using negative ordered resolution for all the problems except CYC and the set of support resolution for CYC. The gkc column gives the pure search time for the gkc prover used as a base for building the CONFER system, for the original TPTP versions (without a question of substitutions being asked). Pure search time does not include printing, parsing and clausifying the problem and indexing the formed clauses. The final column “gkc full” gives full wall clock time for gkc.

Problem	CONFER	CONFER 1.0	gkc pure	gkc full
Steamroller	0.0018	0.0015	0.001	0.06
Dreadbury	0.0017	0.0011	0.001	0.06
Lukasz 0		0.0916	0.093	0.22
Lukasz 1	0.913			
Lukasz 2	23			
Lukasz 3	19			
Lukasz 4	16			
CSR025+5	0.0004	0.0001	0.0001	4.5
CSR035+5	0.0001	0.0001	0.07	4.6
CSR045+5	3.418	1.4	1.3	5.8
CSR055+5	0.0001	0.0001	0.0001	4.5

We can observe that the confidence and dependency collecting calculations along with the restricted c-subsumption do not have a noticeable effect on performance for most of these problems. However, adding confidences below 1 to the Lukasziewicz problem do incur a significant penalty, which – surprisingly – diminishes somewhat when all the clauses have such confidences. The confidences incur a noticeable penalty to CSR045+5, which has the longest proof among our CYC examples. Our hypotheses is that for these examples the c-subsumption along with restricted simplification changes the direction of the search significantly.

5 Summary and Future Work

We have presented a novel framework CONFER along with the implementation for reasoning with approximate confidences for full, unrestricted first order logic. The presented examples demonstrate that the confidences found by our implementation are similar to the confidences found by the leading probabilistic Prolog and Markov logic implementations ProbLog2 [18] and Alchemy 2 [1]. CONFER is based on conventional first order theorem proving theory and algorithms not requiring saturation, differently from the systems using weighted ground saturation of FOL formulas like ProbLog2 and Alchemy 2. We have shown that this enables the CONFER implementation to efficiently solve large nontrivial FOL problems with attached confidences.

We plan to continue work on the CONFER implementation in several directions: finding and removing bugs, improving the functionality and devising search strategies specialized for the FOL formulas with associated confidences. We expect to integrate machine learning approaches, in particular using semantic similarities for reasoning with analogies and estimating the relevance of input clauses for proof search guidance. The goal of this work is creating a practically usable component for logic-based question answering from large commonsense knowledge bases.

References

1. Alchemy 2 system. <https://code.google.com/archive/p/alchemy-2/>
2. Alchemy 2 system repository. <https://github.com/PhDP/alchemy2>
3. Bachmair, L., Ganzinger, H.: Resolution theorem proving. In: Robinson, A., Voronkov, A. (eds.) *Handbook of Automated Reasoning*, vol. I, ch. 2. pp. 19–99. Elsevier (2001)
4. Baltag, A., Smets, S.: Keep changing your beliefs, aiming for the truth. *Erkenntnis* **75**(2), 255–270 (2011)
5. Beltagy, I., Chau, C., Boleda, G., Garrette, D., Erk, K., Mooney, R.: Montague meets Markov: Deep semantics with probabilistic logical form. In: Diab, M., Baldwin, T., Baroni, M. (eds.) *Proc. of *SEM'12 – the 2nd Joint Conference on Lexical and Computational Semantics*. pp. 11–21. Association for Computational Linguistics (2013)
6. Buchanan, B., Shortliffe, E.: *Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project*. Addison Wesley (1984)
7. Chaliar, Y., Razniewski, S., Weikum, G.: Joint reasoning for multi-faceted common-sense knowledge. *CoRR* abs/2001.04170 (2020), <https://arxiv.org/abs/2001.04170>
8. Conesa, J., Storey, V., Sugumaran, V.: Usability of upper level ontologies: The case of ResearchCyc. *Data and Knowledge Engineering* **69**(4), 343–356 (2010)
9. de Salvo Braz, R., Amir, E., Roth, D.: Lifted first-order probabilistic inference. In: Kaelbling, L.P., Saffioti, A. (eds.) *Proc. of IJCAI'05 – the 19th Intl. Joint Conf. on Artificial intelligence*. pp. 1319–1325. Professional Book Center (2005)
10. Domingos, P.: *The Master Algorithm: How the Quest for the Ultimate Learning Machine Will Remake Our World*. Basic Books (2015)
11. Domingos, P., Gogate, V.: Probabilistic theorem proving. In: Cozman, F., Pfeffer, A. (eds.) *Proc. of UAI'11 – the 27th Conf. on Uncertainty in Artificial Intelligence*. pp. 256–265. AUAI (2011)
12. Domingos, P., Kok, S., Poon, H., Richardson, M., Singla, P.: Unifying logical and statistical AI. In: Cohn, A. (ed.) *Proc. of AAAI'06 – the 21st National Conf. on Artificial Intelligence*. pp. 2–9. AAAI (2006)
13. Domingos, P.M. and Kok, S., Lowd, D., Poon, H., Richardson, M., Singla, P.: Markov Logic. In: De Raedt, L., Frasconi, P., Kersting, K., Muggleton, S. (eds.) *Probabilistic Inductive Logic Programming: Theory and Applications*. LNCS, vol. 4911, pp. 92–117. Springer (2008)
14. Dong, X., Gabrilovich, E., Heitz, G., Horn, W., Lao, N., Murphy, K., Strohmman, T., Sun, S., Zhang, W.: Knowledge vault: a Web-scale approach to probabilistic knowledge fusion. In: Macskassy, S.A., Perlich, C., Leskovec, J., Wang, W., Ghani, R. (eds.) *Proc. of KDD'14 – the 20th ACM SIGKDD Intl. Conf. on Knowledge Discovery and Data Mining*. pp. 601–610. ACM (2014)
15. Draheim, D., Tammet, T.: From sensors to Dempster-Shafer theory and back: The axiom of ambiguous sensor correctness and its applications. In: Hartmann, S., Küng, J., Kotsis, G., Tjoa, A.M., Khalil, I. (eds.) *Proc. of DEXA'2020 – the 31st Intl. Conf. on Database and Expert Systems Applications*. LNCS, vol. 12391, pp. 3–19. Springer (2020)
16. Draheim, D.: *Generalized Jeffrey Conditionalization – A Frequentist Semantics of Partial Conditionalization*. Springer (2017)
17. Ferrucci, D.A.: Introduction to “This is Watson”. *IBM Journal of Research and Development* **56**(3.4), 1–15 (2012)

18. Fierens, D., den Broeck, V., G., Renkens, J., Shterionov, D., Gutmann, D., Thon, I., Janssens, G., De Raedt, L.: Inference and learning in probabilistic logic programs using weighted Boolean formulas. *Theory and Practice of Logic Programming* **15**(3), 358–401 (2015)
19. Formato, F., Gerla, G., Sessa, M.: Similarity-based unification. *Fundamenta Informaticae* **41**(4), 393–414 (2000)
20. Furbach, U., Krämer, T., Schon, C.: Names are not just sound and smoke: Word embeddings for axiom selection. In: Fontaine, P. (ed.) *Proc. of CADE'2019 – the 27th Intl. Conf. on Automated Deduction*. LNCS, vol. 11716, pp. 250–268. Springer (2019)
21. Green, C.: Theorem proving as a basis for question-answering systems. *Machine Intelligence* **4**, 183–205 (1969)
22. Hájek, A.: Interpretations of Probability. In: *Stanford Encyclopedia of Philosophy* (2019), <https://plato.stanford.edu/entries/probability-interpret/>
23. Kalyanpur, A., Breloff, T., Ferrucci, D.A., Lally, A., Jantos, J.: Braid: Weaving symbolic and statistical knowledge into coherent logical explanations. *CoRR abs/2011.13354* (2020), <https://arxiv.org/abs/2011.13354>
24. Khot, T., Balasubramanian, N., Gribkoff, E., Sabharwal, E., Clark, P., Etzioni, O.: Exploring Markov Logic Networks for question answering. In: Màrquez, L., Callison-Burch, C., Su, J. (eds.) *Proc. of EMNLP'2015 – the 2015 Conference on Empirical Methods in Natural Language Processing*. pp. 685–694. Association for Computational Linguistics (2015)
25. Lenat, D., Prakash, M., Shepherd, M.: Using common sense knowledge to overcome brittleness and knowledge acquisition bottlenecks. *AI Magazine* **6**(4), 65–85 (1985)
26. Marcus, G.: The next decade in AI: four steps towards robust artificial intelligence. *CoRR abs/2002.06177* (2020), <https://arxiv.org/abs/2002.06177>
27. Mitchell, T., Cohen, W., Hruschka, E., Talukdar, P., Yang, B., Betteridge, J., Carlson, A., Dalvi, B., Gardner, M., Kisiel, B.: Never-ending learning. *Communications of the ACM* **61**(5), 103–115 (2018)
28. Pileggi, S.F.: Web of similarity. *Journal of Computational Science* **36**(100578), 1–7 (2019)
29. Problog2. <https://dtai.cs.kuleuven.be/problog/>
30. Ramachandran, D., Reagan, P., Goolsbey, K.: First-orderized researchcyc: Expressivity and efficiency in a common-sense ontology. In: *AAAI workshop on contexts and ontologies: theory, practice and applications*. pp. 33–40 (2005)
31. Reiter, R.: A logic for default reasoning. *Artificial Intelligence* **13**(1–2), 81–132 (1980)
32. Richardson, M., Domingos, P.: Markov Logic Networks. *Machine Learning* **62**(1–2), 107–136 (2006)
33. Romero, J., Razniewski, S., Pal, K., Pan, J.Z., Sakhadeo, A., Weikum, G.: Commonsense properties from query logs and question answering forums. In: Zhu, W., Tao, D., Cheng, X., Cui, P., Rundensteiner, E.A., Carmel, D., He, Q., Yu, J.X. (eds.) *Proc. of CIKM'19 – the 28th ACM Intl. Conf. on Information and Knowledge Management*. pp. 1411–1420. ACM (2019)
34. Sato, T.: Generative modeling by PRISM. In: Hill, P.M., Warren, D.S. (eds.) *Proc. ICLP'2009 – the 25th Intl. Conf. on Logic Programming*. LNCS, vol. 5649, pp. 24–35. Springer (2009)
35. Speer, R., Chin, J., Havasi, C.: ConceptNet 5.5: An open multilingual graph of general knowledge. In: Singh, S.P., Markovitch, S. (eds.) *Proc. of AAAI'2017 – the 31st AAAI Conf. on Artificial Intelligence*. pp. 4444–4451. AAAI (2017)

36. Sutcliffe, G.: The CADE ATP system competition – CASC. *AI Magazine* **37**(2), 99–101 (2016)
37. Sutcliffe, G., Yeralapudi, A., Trac, S.: Multiple answer extraction for question answering with automated theorem proving systems. In: Lane, H.C., Guesgen, H.W. (eds.) *Proc. of FLAIRS'22 – the 22nd Intl. Florida Artificial Intelligence Research Society Conference. AAAI* (2009)
38. Tammet, T.: Completeness of resolution for definite answers. *Journal of Logic and Computation* **5**(4), 449–71 (1995)
39. Tammet, T.: GKC: A reasoning system for large knowledge bases. In: Fontaine, P. (ed.) *Proc. of CADE'2019 – the 27th Intl. Conf. on Automated Deduction. LNCS*, vol. 11716, pp. 538–549. Springer (2019)
40. TPTP homepage. <http://www.tptp.org>
41. Zadeh, L.: Fuzzy logic. *Computer* **21**(4), 94–102 (1988)

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.