Preprint:
Kaur, N., McPheat, L., Russo, A. et al. (2025) An Empirical Study of Conformal Prediction in LLM with ASP Scaffolds for Robust Reasoning. [Preprint - arXiv]

AN EMPIRICAL STUDY OF CONFORMAL PREDICTION IN LLM WITH ASP SCAFFOLDS FOR ROBUST REASONING

Navdeep Kaur & Lachlan McPheat
The Alan Turing Institute
London, UK
{nkaur, lmcpheat}@turing.ac.uk

Alessandra Russo
Imperial College London
The Alan Turing Institute
London, UK
{arusso}@turing.ac.uk

Anthony G. Cohn
The University of Leeds
The Alan Turing Institute
Leeds & London, UK
{acohn}@turing.ac.uk

Pranava Madhyastha
City University of London
The Alan Turing Institute
London, UK
{pmadhyastha}@turing.ac.uk

ABSTRACT

In this paper, we examine the use of Conformal Language Modelling (CLM) alongside Answer Set Programming (ASP) to enhance the performance of standard open-weight LLMs on complex multi-step reasoning tasks. Using the StepGame dataset, which requires spatial reasoning, we apply CLM to generate sets of ASP programs from an LLM, providing statistical guarantees on the correctness of the outputs. Experimental results show that CLM significantly outperforms baseline models that use standard sampling methods, achieving substantial accuracy improvements across different levels of reasoning complexity. Additionally, the *LLM-as-Judge* metric enhances CLM's performance, especially in assessing structurally and logically correct ASP outputs. However, calibrating CLM with diverse calibration sets did not improve generalisability for tasks requiring many more reasoning steps, indicating limitations in handling more complex tasks. [1]

1 INTRODUCTION

Building automated reasoning systems is one of the cornerstone goals of research in artificial intelligence, with wide-reaching applications in a variety of domains. LLMs have demonstrated remarkable capabilities in various natural language processing tasks Min et al. (2021), yet exhibit critical limitations in a variety of reasoning-related tasks, particularly in multistep and spatially complex scenarios Yang et al. (2023); Li et al. (2024). Although these models process and generate natural language with unprecedented fluency, their reasoning capabilities are still limited by several key constraints.

Most LLMs frequently struggle with verifiable, robust and interpretable reasoning. Experimental studies have shown significant variability in LLM responses to identical reasoning related questions which highlights inherent instability Mirzaee & Kordjamshidi (2023). For instance, sampling from an LLM's responses to reasoning-based queries can produce widely varying answers with significantly divergent reasoning traces. This lack of robustness undermines their reliability in critical decision-making contexts Ji et al. (2023). Moreover, tasks that require multiple sequential reasoning steps, such as complex spatial reasoning, often expose fundamental weaknesses in LLM's systematic reasoning capabilities Shi et al. (2022).

[1]We thank Microsoft Research - Accelerating Foundation Models Research program, for the provision of Azure resources to access OpenAI models. AGC also acknowledges partial support from the Economic and Social Research Council (ESRC) under grant ES/W003473/1.

Recent research has increasingly focused on neural-symbolic approaches that combine the strengths of LLMs in learning from data and rigorous reasoning capabilities of symbolic systems Mao et al. (2019); Yu et al. (2024). Specifically, Answer Set Programming (ASP), a type of declarative logic programming, has emerged as a promising candidate for enhancing LLMs' reasoning capabilities (Yang et al., 2023). Recent studies have explored integrating LLMs with ASP to transform natural language into logical representations, particularly for spatial reasoning tasks (Yang et al., 2023; Li et al., 2024).

However, existing approaches face significant challenges in generating reliable intermediate scaffolds: the logical representations that bridge natural language inputs and systematic symbolic reasoning systems. The translation of natural language into formal logic programs introduces substantial uncertainty, as LLMs can produce inconsistent or semantically incorrect translations Li et al. (2024). This variability may become a significant weakness in this reasoning pipeline, potentially compromising the reliability of downstream reasoning tasks Mirzaee & Kordjamshidi (2023).

In this paper, we present an approach that exploits the recently proposed *Conformal Language Modelling* (CLM) Quach et al. (2024) to systematically address these scaffold generation challenges. Conformal prediction (CP), the foundational framework for CLM, offers unique advantages in uncertainty quantification Vovk et al. (2005); Angelopoulos et al. (2022). CLM has been applied to natural language generation tasks like summarisation and question-answering Quach et al. (2024). Although recent advances in conformal prediction for language generation show promise, significant questions remain regarding the scalability, generalisability and robustness of these approaches across reasoning domains Campos et al. (2024). In this paper, we present a thorough analysis of these open challenges through an empirical lens. We assess how calibration in CP affects generalisability on multi-hop reasoning tasks and evaluate the efficacy of adapting metrics for formal language generation (ASP). Additionally, we identify common error types in the ASP scaffolded LLM based reasoning pipelines and examine how these propagate. Our findings demonstrate the reliability and performance of neuro-symbolic reasoning systems, contributing to the development of more robust and interpretable reasoning frameworks.

2 BACKGROUND

In this section we introduce the main components used in our work: conformal prediction, conformal language modelling and answer set programming.

Conformal Prediction is a statistical framework introduced by Vovk et al. (2005) that provides a method for constructing prediction regions with finite-sample guarantees under minimal assumptions. CP can be described as a sampling method which, under certain mild assumptions, guarantees that samples taken from any distribution using this method will be 'correct' with a controlled probability. This theory has been well-received in machine learning, where providing guarantees on the correctness of predictions helps improve performance, in particular since inference using neural networks involves sampling from theoretically unknown distributions Campos et al. (2024). CP has already proven its utility and efficacy in various domains, such as in computer vision tasks Hechtlinger et al. (2018) and drug discovery Cortés-Ciriano & Bender (2019).

Conformal Language Modelling (CLM), developed in Quach et al. (2024), is a recent generalisation of CP which allows conformal sampling of language models. CLM proposes a conformal approach to generating sets of responses with language models, allowing for statistical guarantees in the vast and complex output space of natural language generation. The main contribution of CLM is an algorithm that can control a specific risk: the expected indicator loss that the generated set contains at least one *acceptable* response according to a predefined admission function—while simultaneously maximising other quantities of interest such as quality, diversity, and confidence. However, it remains unclear how CLM affects generalisability, particularly because it has been primarily tested on natural language generation tasks like summarisation, and report generation, where verifying the accuracy of models is not always straightforward Campos et al. (2024). Applying CLM to domains with rigorous notions of correctness, such as formal language generation, can provide better insights into its handling of generalisability and potential limitations.

Answer set programming is a type of declarative programming which is commonly used in formal representations of knowledge, constraint satisfaction tasks, and formal common-sense reasoning.

Answer set programs consist of finite sets of facts and rules that are used to generate *answer sets* under *stable model semantics*, which allows for efficient encoding of complex problems, notably in domains like graph search and combinatorial optimization Lifschitz (2019). ASP's utility extends to various industrial applications due to its expressive power and computational efficiency, for instance, in e-tourism Ricca et al. (2010), high-performance computing scheduling Gamblin et al. (2022), configuration management, diagnosis, and planning Erdem et al. (2016). Further, ASP has been used to bolster spatial reasoning in LLMs through neural-symbolic integration Yang et al. (2023).

In the following section, we present a more formal introduction to Conformal Language Modelling (Section 3), which we then adapt to our purposes in Section 4. With the adapted CLM, we set up an experiment in Section 5 executed on a reasoning task in Section 6 where we present and analyse the results. The main conclusions are presented in section 7 where we also discuss future directions.

3 Formal introduction to CLM

As mentioned in Section 2, CLM extends the statistical guarantees of CP to the domain of language modelling Quach et al. (2024). CP provides statistical guarantees of sampling procedures while being agnostic to the underlying distribution, making its adaptation to LLMs a significant feat. We briefly summarise the mechanics of CLM below, setting the stage for the adaptations we make for *formal* language modelling in our experiments.

At its core, CLM functions as a calibrated rejection sampling algorithm, allowing one to form **conformal sets of samples**, $\mathcal{C}(X)$, from an LLM, given a prompt X, such that the expected loss of $\mathcal{C}(X)$ is guaranteed to be within a chosen bound. Concretely, for a given loss function L and error tolerance $\delta \in (0, 1)$, CLM ensures there exists a risk tolerance $\varepsilon \in (0, 1)$ such that for any prompt X, the conformal set $\mathcal{C}(X)$ satisfies the Equation 1 with probability at least $1 - \delta$:

$$\mathbb{P}(\mathbb{E}[L(\mathcal{C}(X))] \leq \varepsilon) \geq 1 - \delta. \tag{1}$$

This leads to the important question: How is the conformal set $\mathcal{C}(X)$ defined? In Quach et al. (2024), the authors define $\mathcal{C}(X)$ using a rejection sampling algorithm that incorporates three key metrics: a) **quality of samples**: $\mathcal{Q}$, which measures how good each sample is; b) **diversity of samples**: $\mathcal{S}$, which ensures that the samples are sufficiently diverse; and c) **confidence of the set of samples**: $\mathcal{F}$, which evaluates the overall confidence in the correctness of the sample set.

Before considering these metrics however, one requires a method to filter out 'unacceptable' samples. This is achieved using an **admission function**, A, which maps samples to $\{0, 1\}$, effectively discarding samples that do not meet certain criteria. The choice of admission function can be tailored to specific tasks while still maintaining the coverage from Equation 1. Indeed, in Quach et al. (2024) we are presented with a range of instances of A. For instance,for the summarisation task ROUGE_L Lin (2004) scores were used, which focuses on measuring the longest common subsequence between two samples. This is used to compare sample summaries y to the references X being summarised (concretely, $A(y) = 1$ if $\mathrm{ROUGE}_L(X, y) > 0.35$ and 0 otherwise).

The construction of a conformal set $\mathcal{C}(X)$ in the context of language modelling given a prompt X is done recursively using the chosen metrics, $\mathcal{Q}, \mathcal{S}, \mathcal{F}$ along with associated constraints $\lambda = (\lambda_1, \lambda_2, \lambda_3) \in \mathbb{R}^3$, which we learn in advance from the calibration process, explained in the following section.

The recursive construction of $\mathcal{C}(X)$ involves repeatedly sampling the LLM and rejecting samples which do not meet the constraints established by the admission function and the quality, diversity and confidence measurements $\mathcal{Q}, \mathcal{S}, \mathcal{F}$ introduced above. Concretely, we define a chain of sets $\mathcal{C}_0(X) \subseteq \mathcal{C}_1(X) \subseteq \cdots \subseteq \mathcal{C}_k(X)$, where k is the sampling budget. For each $i = 1, \ldots, k$ the set $\mathcal{C}_i(X)$ is defined according to Algorithm 1 in Quach et al. (2024), which we simplify below. We will take y_i to denote the i^{th} sample of the LLM with prompt X, given that y_i is admissible, i.e. $A(y_i) = 1$[2]. The initial set, assuming $A(y_0) = 1$, is $\mathcal{C}_0(X) = \{y_0\}$. For each $i = 1, \ldots, k$ the set $\mathcal{C}_i(X)$ is defined as $\mathcal{C}_{i-1} \cup \{y_i\}$ if the two conditions below are met:

1. y_i is of high enough quality, i.e. $\mathcal{Q}(y_i|X) \geq \lambda_1$

[2] Note that it is indeed possible for none of the samples to be admissible, i.e. $A(y_i) = 0$ for all samples y_i. This simply means the conformal set is empty.

2. y_i is diverse enough, i.e. $\mathcal{S}(y_i, y_j) \leq \lambda_2$ for $0 \leq j < i$.

Finally, after testing quality and diversity, we test whether the confidence of the set is sufficient: $\mathcal{F}(\mathcal{C}_i(X)) \geq \lambda_3$. If this is true, we break the loop and return $\mathcal{C}_i(X)$ as the conformal set. If not, we repeat the loop for the next set $\mathcal{C}_{i+1}(X)$. Next, we present the calibration procedure of Quach et al. (2024) to explain how to find appropriate constraints λ in the first place.

3.1 Calibration

First of all one identifies a calibration set $\mathcal{D}_{\text{cal}}$, often a subset of validation or training data. This data is used to calibrate the values of λ to ensure one's samples conform to a desired standard. The calibration procedure in Quach et al. (2024) is an adaptation of the Learn Then Test (LTT) procedure of Angelopoulos et al. (2022). The calibration procedure is basically a search over a space of weighted configurations $\lambda = (\lambda_1, \lambda_2, \lambda_3)^3$. Given the set of all possible configurations $\Lambda = \{0, 0.01, \ldots, 0.99, 1\}^3$, we consider for each $\lambda \in \Lambda$ and each possible risk-tolerance $\varepsilon \in \{0, 0.01, 0.02, \ldots, 0.99, 1\}$ one calculates the binomial tail bound p-value under the null-hypothesis $\mathcal{H}_\lambda : \mathbb{E}[L(\lambda)] > \varepsilon$. This gives us a p-value $p_{\lambda,\varepsilon} = p_\lambda$:

$$p_\lambda := \mathbb{P}(\text{Bin}(n, \varepsilon) \leq n\hat{R}_n(\lambda)),$$

where $\hat{R}_n(\lambda)$ is the *empirical risk*, taken to be:

$$\hat{R}_n(\lambda) := \frac{1}{n} \sum_{i=1}^{n} L_i(\lambda)$$

and $L_i(\lambda)$ is the *loss*:

$$L_i(\lambda) := \mathbf{1}\{ \nexists y \in \mathcal{C}_\lambda(X_i) \mid A(y) = 1\}.$$

Following this lengthy sequence of calculations of p-values, one applies a family-wise error rate controlling algorithm to the set of p-values to return the set of valid parameters Λ_{valid}. In Quach et al. (2024) this is implemented using the 'Bonferroni correction':

$$\Lambda_{\text{valid}} = \left\{ \lambda \mid p_\lambda \leq \frac{\delta}{|\Lambda|} \right\}.$$

Finally, we identify the configuration $\lambda \in \Lambda_{\text{valid}}$ that minimises the below combination of conformal set size and number of samples needed to predict the correct label over the calibration set $\mathcal{D}_{\text{cal}}$:

$$\frac{1}{|\mathcal{D}_{\text{cal}}|} \sum_{(X,Y) \in \mathcal{D}_{\text{cal}}} \left(\rho_1 |\mathcal{C}_\lambda(X)| + \rho_2 \frac{[S_\lambda(X) - S^*(X)]^+}{S_\lambda(X)} \right)$$

where $S_\lambda(X_i)$ is the total number of samples made before rejecting (e.g. 20), $S^*(X_i)$ is the index of the first correct sample, and $[\cdot]^+ = \max(\cdot, 0)$. The numerator $[S_\lambda(X) - S^*(X)]^+$ quantifies how quickly the LLM generates a correct sample. Since samples are generated sequentially, we ideally want to minimise superfluous generations which is done by minimising this number. We take ρ_1, ρ_2 to be 0.5, as done by Quach et al. (2024) where the reader can find further detail.

4 Methodology

In this work, we are specifically interested in exploring the application of CLM to improve the performance of mid-range open-weight language models of modest size on complex reasoning tasks. Specifically, we focus on the StepGame dataset of Li et al. (2024), which is a correction of the original dataset developed in Shi et al. (2022), which presents spatial reasoning challenges requiring multiple steps of logical deduction. Unlike previous approaches that leverage larger, more computationally intensive LLMs with extensive prompt engineering, our methodology emphasises improving the capabilities of a mid-range LLM through the integration of ASP and conformal prediction techniques. The goal here is to empirically validate the utility of CLM especially for ensuring that

[3]In general, there is a value of λ for each metric, meaning that the configuration-tuple λ can be of arbitrary length.

the generated ASPs indeed maintain highly useful outputs with highly accurate answer sets and to carefully conduct experiments investigating the generalisability across different calibration settings.

The core construct of ASP is the logic rule, formulated as:

$$a_1 \mid \ldots \mid a_n \;:-\; b_1, \ldots, b_k \;;\; \textbf{not}\; b_{k+1}, \ldots,\; \textbf{not}\; b_m$$

Here, a_i and b_i are *atoms* or positive *literals*, and $\textbf{not}\, b_j$ are negative *literals*. Informally, this rule states that at least one of the a_i must be true if all b_i are true and all b_j are false. A rule with no $b_i, \textbf{not}\, b_j$ is called a *fact*. ASP programs consist of sets of such rules and facts, which collectively define the problem space and constraints. ASP is particularly well-suited for multistep reasoning tasks due to its ability to model complex constraints and perform efficient search procedures to find solutions that satisfy all conditions.

In our approach, CLM is employed to ASP programs sampled from the LLM such that the generated programs are not only syntactically valid but also, the answer sets are likely to contain the correct solution when processed by the `clingo` ASP solver Gebser et al. (2018). We investigate how different types of calibration sets—ranging from those containing single-step reasoning examples to those with multiple-step reasoning examples—affect the performance and generalisability of CLM. This focus allows us to understand the importance of sample diversity in the calibration phase and its impact on the model's ability to generalise across varied reasoning tasks.

Our core methodology involves translating natural language inputs from the StepGame dataset into ASP programs using the LLM. To facilitate the translation of natural language task descriptions into Answer Set Programming (ASP) code, we employ *In-Context Learning (ICL)*. ICL exploits the inherent ability of large language models (LLMs) to discern patterns and structures from provided examples within the input prompt, enabling them to generate contextually appropriate outputs without explicit parameter fine-tuning. The entire ASP program is then sampled using CLM, producing a conformal set of programs. These ASP programs are then processed by `clingo` to derive answer sets.

Note that the CLM process itself is carried out in two main stages with the first stage associated with learning λ using the calibration set (see Calibration in Section 3). After the initial calibration, we employ a separate validation set to fine-tune the parameters further. This validation set consists of additional StepGame tasks not included in the calibration phase, providing an unbiased dataset to evaluate the performance of different λ configurations and optimal ε values. This helps us identify the optimal λ and ε settings that maximise the accuracy and reliability of the generated ASP programs. Once the best λ and ε values are determined, they are fixed and used to evaluate the model's performance on an independent test set, ensuring that the selected parameters generalise well to unseen data. We next provide the exact experimental setup and details of our empirical analysis.

5 EXPERIMENTAL SETUP

5.1 DATASET

All our empirical results are based on the StepGame dataset. The StepGame dataset of Li et al. (2024), consisting of story-query-answer tuples of the form $\langle d, q, a\rangle$ where d is a list of natural language descriptions of edges of a graph. These edges are in one of nine configurations: *right, top-right, top, top-left, left, bottom-left, bottom, bottom-right*, and *overlap* which are present in d using a variety of synonyms. q is a natural language query of the relation between two nodes in the graph, and a is the answer. The entries in d differ in the number descriptions of the graph, between 1 and 24, corresponding to the number of hops.

Stepgame dataset has been used to test the spatial reasoning abilities of LLMs, where the multiple steps of reasoning are necessary to induce the relation between the two given nodes in each query as the answer is not directly retrievable in the description alone. Further variants of this dataset are available in Shi et al. (2022) where noise in several different forms is added to the descriptions in the test set. We do not consider these noisy variations here, but leave it for future work.

For our experiments we generated a dataset of 465,975 datapoints split into 341,284 for training, 11,350 for validation and 113,341 for testing. Within each of these sets there is a distribution of different numbers of hops. The generation was performed using the `asp-solution.py` script[4] written for the data-generation in Li et al. (2024).

The first experiment was performed on $6 \times 200 = 1200$ from the StepGame test set, where we randomly sampled 200 entries of 1, 2, 3, 4, 5 and 15 hops respectively. The second experiment only on the entries with 1, 2 and 3 hops.

5.2 ADMISSION

We designed an admission function to filter out non-programs from the samples generated by the LLM, done using an ASP syntax-check. Formally, this admission function, A_{syntax} say, is defined on samples y as:

$$A_{\text{syntax}} = \begin{cases} 1 & \text{if } \texttt{clingo} \text{ parses } y \text{ as a valid program} \\ 0 & \text{if } \texttt{clingo} \text{ cannot parse } y \end{cases}$$

The theory proving the inequality in Equation 1 requires the admission function to return values in $\{0, 1\}$, leaving us the freedom to use A_{syntax}. Enforcing syntactic correctness ensures that only syntactically valid ASPs are admitted into the conformal set. This offers a significant advantage over similarity based admission functions, such as ROUGE scores commonly used in NLG tasks like summarisation (used in Quach et al. (2024)). While ROUGE metrics measure the overlap between generated text and reference summaries, they do not guarantee the functional or structural integrity of the output. In contrast, syntactic correctness provides an objective and reliable estimate of a sample's utility.

5.3 METRICS AND MEASURES

In our first experiment, we adopted the same evaluation metrics used in the text summarisation study by Quach et al. (2024). Specifically, we used the quality metric $\mathcal{Q}$ to be the average transition score of the output and for the confidence function $\mathcal{F}$; we then take the maximum of the transition scores of the output of the LLM. For the diversity metric we used $\mathcal{S} = \text{ROUGE}_L$. However, since ROUGE_L measures sentence-level similarity, it may not be ideal for our formal language-generation, as textually similar strings may still have formal syntactic differences giving rise to unrelated answer sets. Despite this limitation, we justify the adoption of these metrics due to the conceptual similarity between our task and summarisation, as both involve condensing information into a structured format, albeit in different languages.

For the second experiment, we implement an *LLM-as-Judge* metric, inspired by recent methodologies that leverage language models for evaluating and comparing generated outputs Liu et al. (2023); Zheng et al. (2023). This approach uses a fine-tuned language model to assess the quality of generated ASP samples by computing the difference in average log probabilities between pairs of samples. Specifically, we fine-tuned the `llama3.1-8B-Instruct` model on approximately one-third of the training set. This subset includes a representative distribution of reasoning steps ranging from 1 to 24, ensuring that the model is exposed to diverse reasoning complexities during fine-tuning. The fine-tuning process was conducted using the LoRa adapter from the LlamaFactory repository Zheng et al. (2024), allowing efficient adaptation of the model with minimal computational overhead. Notably, the fine-tuned model achieves 100% accuracy on the task, ensuring reliable evaluation performance.

The implementation of the *LLM-as-Judge* metric involves calculating the difference in average log probabilities assigned by the fine-tuned model to two ASP samples. Formally, taking the samples of the fine-tuned language model to be $(y_i)_{i=1,\dots,n}$ and $(z_j)_{j=1,\dots,m}$, the *LLM-as-judge* metric is evaluated on the pair $((y_i), (z_j))$ as:

$$\frac{1}{n}\sum_{i=1}^{n} \log p(y_i|LLM) - \frac{1}{m}\sum_{j=1}^{m} \log p(z_j|LLM).$$

[4] https://github.com/Fangjun-Li/SpatialLM-StepGame/blob/main/asp_solution.py

5.4 Calibration

To evaluate the impact of calibration set composition on the generalisability of Conformal Language Modelling (CLM), we used two distinct calibration sets. The first calibration set comprises StepGame entries where the number of reasoning steps ranges from 1 to 5, encompassing a diverse array of reasoning complexities. The second calibration set is restricted to entries with a only 1 reasoning step. Both calibration sets were extracted from the validation subset and consist of 500 entries each, ensuring a balanced and sufficient sample size for reliable calibration.

The mixed calibration set (1-5 hops) mirrors the distribution of number of reasoning steps present in the test set, thereby maintaining consistency between calibration and evaluation phases. We are interested here in assessing the constraints CLM has on its generalisation capabilities in different constrained scenarios. We are interested in determining the extent to which the diversity of calibration data influences the model's ability to generalise to more complex and varied reasoning tasks.

5.5 In context learning

We designed two ICL prompts, each containing two exemplar pairs from the StepGame dataset. These prompts are comprehensively detailed in Appendix B. The first ICL prompt includes examples with 2 and 4 reasoning steps respectively, representing intermediate levels of reasoning complexity. The second ICL prompt consists of two 1-hop examples, focusing on single-step reasoning tasks. The selection of prompts is directly aligned with the calibration sets: the prompt containing 2 and 4-hop examples is used when calibrating with the mixed set (1-5 hops), while the prompt with 1 reasoning step is employed when calibrating with the single-length set. This distinction is repeated in the baseline experiments too, to give a fairer comparison to the CLM results.

5.6 Language models

For our experiments, we used Meta's `llama3.1-8b-Instruct` model Llama (2024) as the foundational language model around which our CLM framework is constructed. This model represents the smallest yet highly performant open-weight language model available at the time of our study, achieving competitive results on standard language model benchmarks Srivastava et al. (2022). The selection of `llama3.1-8b-Instruct` is primarily motivated by our focus on enhancing the capabilities of less complex models through CLM. Using a smaller model will help us test rigourously the capability of CLM in enhancing the reasoning capabilities of LLMs without relying on extensive parametrisation or specialised training procedures.

6 Experiments and results

6.1 Experiment 1

In this experiment, we evaluated the effectiveness of CLM in generating accurate ASPs for StepGame dataset. Specifically we employ the following list of metrics and functions througout this experiment. We use confidence $\mathcal{Q}$ as in Quach et al. (2024), ROUGE_L Lin (2004) as our diversity-metric $\mathcal{S}$, and the maximum logprob over the set as our confidence function $\mathcal{F}$[5]. We also instantiate the admission function A using `clingo` to test the syntactic correctness of the sample[6].

We tested the two different calibration settings on six portions of the test-set, split according to multi-step reasoning including 1, 2, 3, 4, 5 and 15-steps to test the efficacy of calibration on both in and out-of-distribution data.

The baseline experiments were instantiated using the same LLM, where we provided the same ICL prompts as for the CLM sampling but then only sampled once, greedily, maintaining all other hyperparameters. The LLM-output is post-processed and then fed to `clingo` to generate answer sets.

[5]That is $\mathcal{F}(\mathcal{C}(X)) = \max\{-log(p(y|X)) \mid y \in \mathcal{C}(X)\}$

[6]We employ some minor post-processing to the samples. We found the LLM tended to repeat lines of code which would be cut off by the max-token limit, causing syntactic errors. Hence we remove any incomplete lines of code produced by the LLM. In early experiments we identified a short list of tokens which caused syntactic errors, which we used as stopping-tokens at inference time.

# hops →	1	2	3	4	5	15
Calib. 1 − 5	80.0	72.5	70.5	70.0	65.0	0.5
Calib. 1	71.5	65.5	55.5	54.0	50.0	-
Basel. 2 + 4	45.0	39.5	36.5	30.0	28.5	0
Basel. 1	42.0	28.5	27.5	19.5	13.5	0

Table 1: CLM results (accuracies in %) on StepGame task, using the ASP-parser as the admission function. **Calib.** refers to *Calibration Length*, where *1-5* refers to calibration on 1 to 5-hop StepGame entries, and 1 to calibration on single-hop entries. **Basel.** refers to *Baseline* and **2 + 4** and **1** refer to the type of ICL prompt used, see Section 5 for further detail.

Note that we consider a sample program correct iff there is only a single answer set generated by `clingo` and it contains only one answer-predicate and that answer predicate is the correct one.

The results of this experiment are presented in Table 1, where we observe firstly that CLM significantly improved accuracy across all test sets by at least 20 percentage points in comparison to the baseline results. The calibration set consisting of 1 to 5-hop examples ('**Calib. 1 - 5**') consistently outperformed the calibration set limited to single hop ('**Calib. 1**'). This trend is evident across all test segments, with differences becoming more pronounced as the number of reasoning steps increased. This highlights the importance of diverse calibration data in enabling the model to generalise effectively to more complex scenarios.

One notable aspect of the experiment is the evaluation on the 15-hop test set, which falls outside the range of both calibration settings. Here, the performance drops significantly and the generated answer sets are either null or mostly incorrect. This result underscores the inherent challenge of generalising to significantly longer sequences without specific calibration data. It suggests that, while CLM with diverse calibration improves generalisation, there are still considerable difficulties in handling sequences that require substantially more reasoning steps than those seen during calibration. Finally, the improvements in performance with CLM underscore the capability of ASP to effectively capture and reason through complex logical relationships.

6.2 EXPERIMENT 2

In our second experiment, we evaluated the effectiveness of the purpose-built *LLM-as-Judge* metric, as introduced in Section 5. This new metric was designed to overcome some of the inherent limitations of conventional metrics, such as the ROUGE-L score, when applied to formal language tasks like ASP generation. The remaining metrics, admission functions, and hyperparameters were kept identical to those in Experiment 1 to allow for a direct comparison of performance outcomes. The results are presented in Table 2.

# hops →		1	2	3
Calib. 1 − 5	LLM as Judge	80.0	72.5	72.0
	ROUGE − L score	80.0	72.5	70.5
Calib. 1	LLM as Judge	76.5	68.5	60.5
	ROUGE − L score	71.5	65.5	55.5

Table 2: Results (accuracies in %) from using LLM-as-judge as a similarity metric in the CLM and calibration procedures.

We observe that the *LLM-as-Judge* metric provided a consistent improvement over the ROUGE-L score across the calibration settings, especially for the more complex descriptions (lengths 2 and 3). This performance closely matches or slightly outperforms the ROUGE_L metric for each corresponding number of reasoning steps. This improvement suggests that *LLM-as-Judge* is better at assessing the quality of complex reasoning samples by capturing the nuances of logical structure that are essential for ASP correctness. For single-length calibration set, *LLM-as-Judge* metric demonstrated a marked improvement over the ROUGE-L score across all numbers of reasoning steps.

This experiment demonstrates the primary limitation of the ROUGE-L metric, as observed in these experiments, lies in its focus on n-gram similarity, which is less suitable for evaluating formal languages like ASP. ROUGE-L does not adequately capture structural or functional correctness, leading to the acceptance of syntactically incorrect or logically inconsistent samples.

6.3 Error analysis

We found there are five type of errors that are made by the LLM while generating answer set programs. We list them below in decreasing order of prevalence. The most prominent error is when `clingo` returns '**`empty`**'. This is because `clingo` failed to execute the ASP program successfully. This might be because: (i) The LLM generated a wrong fact corresponding to an instruction; for example, for a given instruction 'L is at F's 9 o' clock', the LLM may generate at("L", "F", 9) which is none of the nine possible configurations in Section 5. (ii) The rules have typos in them, for example: is(A, down_right, B):− down_right(B) (iii) The LLM generates gibberish at the end of a correct ASP program This makes `clingo` raise a parse error (iv) Instead of generating a valid ASP program, the LLM generates natural language text which is read as gibberish by `clingo`.

The second most prominent kind of error is when `clingo` returns one answer set missing any answer[7]. Such answer sets are not empty, as answer may contain facts it has been given or other inferred facts. This could be due to at least one of the ASP facts corresponding to an instruction being wrong. The third most common type of error is when `clingo` returns one answer set containing multiple answers. This could be due to wrong rule being generated by LLM such as:

```
is(A, down_left, B):- down_left(A,B).
is(A, down_right, B):- down_left(A,B).
```

When this rule is fired, it would lead to two answers for a given query.

The fourth most common type of error is when `clingo` returns one answer set containing one answer but the answer is incorrect. The most common reason is that translation done by the LLM from NLP to ASP facts is wrong. For example:

```
% U is positioned above N and to the right.
top("U", "N").
```

In the above instruction, the correct answer generated by the LLM should have been *top-right("U", "N")*.

The final type of error is when `clingo` returns multiple answer sets for a given input instruction. In principle, there are plenty of reasons for ASP programs to have such answer sets, but in this case this is due to the LLM generating multiple facts corresponding to one NLP instruction in a single row separated by commas, for example:

```
% F is on the right side and below X.
right("F", "X"), down("F", "X").
```

The comma in the two facts acts as disjunction, meaning that `clingo` may use either of the two facts, resulting in two answer sets for the same program.

7 Conclusions

In this study, we explored the application of CLM to enhance the performance of a mid-range open-weight language model on complex reasoning tasks, specifically using the StepGame dataset. The CLM integration significantly improved accuracy on the StepGame task. When compared to baseline models that relied on standard sampling methods, CLM consistently outperformed across all tested segments. We also observe that the *LLM-as-Judge* metric further improved the performance of CLM, in particular on multi-step reasoning. We observed that the diversity of samples is an important consideration for the calibration set. However, we also observed that calibrating CLM on a diverse set of examples (1 to 5 reasoning steps) did not enhance generalisability to tasks requiring

[7]The answer in this case is a fact of the form **`answer`**(R), where R is one of the nine relations.

significantly more reasoning steps (e.g. 15-hop). These results underscore the potential of CLM in augmenting the reasoning capabilities of standard language models, particularly when complemented by advanced evaluation metrics like *LLM-as-Judge*. However, the diminished performance on higher-complexity tasks shows the potential limitations of CLM. Future work could focus on a thorough examination of the statistics related to conformal set sizes, such as their distribution and relationship with task complexity. This could inform more effective sampling strategies and constraint settings within the CLM framework. Repeating the experiments multiple times to enable the calculation of error bars would also be desirable. Finally, a further baseline could be provided consisting of the LLM alone, without recourse to the external ASP reasoner.

REFERENCES

Anastasios N. Angelopoulos, Stephen Bates, Emmanuel J. Candès, Michael I. Jordan, and Lihua Lei. Learn then Test: Calibrating Predictive Algorithms to Achieve Risk Control, September 2022. URL http://arxiv.org/abs/2110.01052. arXiv:2110.01052 [cs, stat].

Margarida Campos, António Farinhas, Chrysoula Zerva, Mário AT Figueiredo, and André FT Martins. Conformal prediction for natural language processing: A survey. *Transactions of the Association for Computational Linguistics*, 12:1497–1516, 2024.

Isidro Cortés-Ciriano and Andreas Bender. Concepts and Applications of Conformal Prediction in Computational Drug Discovery, August 2019. URL http://arxiv.org/abs/1908.03569. arXiv:1908.03569.

Esra Erdem, Michael Gelfond, and Nicola Leone. Applications of answer set programming. *AI Magazine*, 37(3):53–68, 2016.

Todd Gamblin, Massimiliano Culpo, Gregory Becker, and Sergei Shudler. Using Answer Set Programming for HPC Dependency Solving, October 2022. URL http://arxiv.org/abs/2210.08404. arXiv:2210.08404.

Martin Gebser, Roland Kaminski, Benjamin Kaufmann, and Torsten Schaub. Multi-shot asp solving with clingo, 2018. URL https://arxiv.org/abs/1705.09811.

Yotam Hechtlinger, Barnabás Póczos, and Larry Wasserman. Cautious deep learning. *arXiv preprint arXiv:1805.09460*, 2018.

Ziwei Ji, Nayeon Lee, Rita Frieske, Tiezheng Yu, Dan Su, Yan Xu, Etsuko Ishii, Ye Jin Bang, Andrea Madotto, and Pascale Fung. Survey of hallucination in natural language generation. *ACM Comput. Surv.*, 55(12), mar 2023. ISSN 0360-0300. doi: 10.1145/3571730. URL https://doi.org/10.1145/3571730.

Fangjun Li, David C. Hogg, and Anthony G. Cohn. Advancing Spatial Reasoning in Large Language Models: An In-Depth Evaluation and Enhancement Using the StepGame Benchmark, January 2024. URL http://arxiv.org/abs/2401.03991. arXiv:2401.03991 [cs].

Vladimir Lifschitz. *Answer Set Programming*. Springer, 2019.

Chin-Yew Lin. ROUGE: A Package for Automatic Evaluation of Summaries. In *Text Summarization Branches Out*, pp. 74–81, Barcelona, Spain, July 2004. Association for Computational Linguistics. URL https://aclanthology.org/W04-1013.

Yang Liu, Dan Iter, Yichong Xu, Shuohang Wang, Ruochen Xu, and Chenguang Zhu. G-eval: Nlg evaluation using gpt-4 with better human alignment. *arXiv preprint arXiv:2303.16634*, 2023.

Team Llama. The Llama 3 Herd of Models, November 2024. URL http://arxiv.org/abs/2407.21783. arXiv:2407.21783.

Jiayuan Mao, Chuang Gan, Pushmeet Kohli, Joshua B Tenenbaum, and Jiajun Wu. The neuro-symbolic concept learner: Interpreting scenes, words, and sentences from natural supervision. *arXiv preprint arXiv:1904.12584*, 2019.

Bonan Min, Hayley Ross, Elior Sulem, Amir Pouran Ben Veyseh, Thien Huu Nguyen, Oscar Sainz, Eneko Agirre, Ilana Heintz, and Dan Roth. Recent advances in natural language processing via large pre-trained language models: A survey. *CoRR*, abs/2111.01243, 2021. URL https://arxiv.org/abs/2111.01243.

Roshanak Mirzaee and Parisa Kordjamshidi. Disentangling extraction and reasoning in multi-hop spatial reasoning. *arXiv preprint arXiv:2310.16731*, 2023.

Victor Quach, Adam Fisch, Tal Schuster, Adam Yala, Jae Ho Sohn, Tommi S. Jaakkola, and Regina Barzilay. Conformal Language Modeling, June 2024. URL http://arxiv.org/abs/2306.10193. arXiv:2306.10193 [cs].

Francesco Ricca, Antonella Dimasi, Giovanni Grasso, Salvatore Ielpa, Salvatore Iiritano, Marco Manna, and Nicola Leone. A Logic-Based System for e-Tourism. *Fundam. Inform.*, 105:35–55, January 2010. doi: 10.3233/FI-2010-357.

Zhengxiang Shi, Qiang Zhang, and Aldo Lipani. StepGame: A New Benchmark for Robust Multi-Hop Spatial Reasoning in Texts, April 2022. URL http://arxiv.org/abs/2204.08292. arXiv:2204.08292 [cs].

Aarohi Srivastava, Abhinav Rastogi, Abhishek Rao, Abu Awal Md Shoeb, Abubakar Abid, Adam Fisch, Adam R Brown, Adam Santoro, Aditya Gupta, Adrià Garriga-Alonso, et al. Beyond the imitation game: Quantifying and extrapolating the capabilities of language models. *arXiv preprint arXiv:2206.04615*, 2022.

Vladimir Vovk, Alex Gammerman, and Glenn Shafer. *Algorithmic Learning in a Random World*. Springer-Verlag, Berlin, Heidelberg, 2005. ISBN 0387001522.

Zhun Yang, Adam Ishay, and Joohyung Lee. Coupling Large Language Models with Logic Programming for Robust and General Reasoning from Text. In Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki (eds.), *Findings of the Association for Computational Linguistics: ACL 2023*, pp. 5186–5219, Toronto, Canada, July 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.findings-acl.321. URL https://aclanthology.org/2023.findings-acl.321.

Fei Yu, Hongbo Zhang, Prayag Tiwari, and Benyou Wang. Natural language reasoning, a survey. *ACM Computing Surveys*, 56(12):1–39, 2024.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in Neural Information Processing Systems*, 36:46595–46623, 2023.

Yaowei Zheng, Richong Zhang, Junhao Zhang, Yanhan Ye, Zheyan Luo, Zhangchi Feng, and Yongqiang Ma. LlamaFactory: Unified Efficient Fine-Tuning of 100+ Language Models, June 2024. URL http://arxiv.org/abs/2403.13372. arXiv:2403.13372.

A Appendix

B In-context learning prompts

We prepend the following ICL prompts to the test-entries in the experiments in section 6. The %-character is a comment-character in `clingo`, and the #-character denotes the beginning of a new example.

B.1 Instruction length 1

```
You are an text to Answer Set Program (ASP) translator.
Users will ask you questions in English and you will generate a
ASP program based on the provided input.
Example 1:
### Instruction:
```

```
Generate the Answer Set Program(ASP) for the given input problem.
Each numbered point in the input will have one corresponding ASP
fact as output. Additionally, include some generic ASP rules that
will help the user in solving the problem. 1 X is to the left of K
and is on the same horizontal plane.
2 What is the relation of the agent X to the agent K?

### Response:
 % we generate one fact for each input sentence:
% 1 X is to the left of K and is on the same horizontal plane.
left(”X”, ”K”).

% 2 What is the relation of the agent X to the agent K?
query(”X”, ”K”).

% Now we consider the answer set rules that will help us arrive
at the final answer:
% assume the 2nd queried object is at location (0,0)
location(Q2, 0, 0) :- query(_, Q2).

% extract answer relation R such that the offset (Ox,Oy) of R is in
the same direction of (X,Y)
answer(R) :- query(Q1, _), location(Q1, X, Y), offset(R, Ox, Oy),
Ox=-1: X$<$0; Ox=0: X=0; Ox=1: X$>$0; Oy=-1: Y$<$0; Oy=0: Y=0;
Oy=1: Y$>$0.

% general format translation, which can also be easily done in
% python script (this is not needed if we directly extract the
%  general form in the beginning as in bAbI task4)
is(A, top, B) :- top(A, B).
is(A, top, B) :- up(A, B).
is(A, down, B) :- down(A, B).
is(A, left, B) :- left(A, B).
is(A, right, B) :- right(A, B).
is(A, top_left, B) :- top_left(A, B).
is(A, top_right, B) :- top_right(A, B).
is(A, down_left, B) :- down_left(A, B).
is(A, down_right, B) :- down_right(A, B).
is(A, east, B) :- east(A, B).
is(A, west, B) :- west(A, B).
is(A, south, B) :- south(A, B).
is(A, north, B) :- north(A, B).

% synonyms
synonyms(north, northOf; south, southOf; west, westOf; east,
eastOf; top, northOf; down, southOf; left, westOf; right,
eastOf).
synonyms(A, B) :- synonyms(B, A).
synonyms(A, C) :- synonyms(A, B), synonyms(B, C), A!=C.

% define the offsets of 8 spacial relations
offset(overlap,0,0; top,0,1; down,0,-1; left,-1,0; right,1,0;
top_left,-1,1; top_right,1,1; down_left,-1,-1; down_right,1,-1).

% derive the kind of spacial relation from synonyms and offset
is(A, R1, B) :- is(A, R2, B), synonyms(R1, R2).
is(A, R1, B) :- is(B, R2, A), offset(R2,X,Y), offset(R1,-X,-Y).

% derive the location of every object
```

```
% the search space of X or Y coordinate is within −100 and 100
(to avoid infinite loop in clingo when data has error)
nums(−100..100).

location(A, Xa, Ya) :− location(B, Xb, Yb), nums(Xa), nums(Ya),
is(A, Kind, B), offset(Kind, Dx, Dy), Xa−Xb=Dx, Ya−Yb=Dy.

location(B, Xb, Yb) :− location(A, Xa, Ya), nums(Xb), nums(Yb),
is(A, Kind, B), offset(Kind, Dx, Dy), Xa−Xb=Dx, Ya−Yb=Dy.
Example 2: ### Instruction:
Generate the Answer Set Program(ASP) for the given input problem.
Each numbered point in the input will have one corresponding ASP
fact as output. Additionally, include some generic ASP rules
that will help the user in solving the problem. 1 G is at the 6 o
'clock position relative to R.
2 What is the relation of the agent G to the agent R?

### Response:
% we generate one fact for each input sentence:
% 1 G is at the 6 o'clock position relative to R.
down("G", "R").

% 2 What is the relation of the agent G to the agent R?
query("G", "R").

% Now we consider the answer set rules that will help us arrive
 at the final answer:
% assume the 2nd queried object is at location (0,0)
location(Q2, 0, 0) :− query(_, Q2).

% extract answer relation R such that the offset (Ox,Oy) of R is
% in the same direction of (X,Y)
answer(R) :− query(Q1, _), location(Q1, X, Y), offset(R, Ox, Oy),
Ox=−1: X$<$0; Ox=0: X=0; Ox=1: X$>$0; Oy=−1: Y$<$0; Oy=0: Y=0;
Oy=1: Y$>$0.

% general format translation, which can also be easily done in
python script
% (this is not needed if we directly extract the general form in
the beginning as in bAbI task4)
is(A, top, B) :− top(A, B).
is(A, top, B) :− up(A, B).
is(A, down, B) :− down(A, B).
is(A, left, B) :− left(A, B).
is(A, right:%s/, B) :− right(A, B).
is(A, top_left, B) :− top_left(A, B).
is(A, top_right, B) :− top_right(A, B).
is(A, down_left, B) :− down_left(A, B).
is(A, down_right, B) :− down_right(A, B).
is(A, east, B) :− east(A, B).
is(A, west, B) :− west(A, B).
is(A, south, B) :− south(A, B).
is(A, north, B) :− north(A, B).

% synonyms
synonyms(north, northOf; south, southOf; west, westOf; east,
eastOf; top, northOf; down, southOf; left, westOf; right,
eastOf).
synonyms(A, B) :− synonyms(B, A).
```

```
synonyms(A, C) :- synonyms(A, B), synonyms(B, C), A!=C.

% define the offsets of 8 spacial relations
offset(overlap,0,0; top,0,1; down,0,-1; left,-1,0; right,1,0;
top_left,-1,1; top_right,1,1; down_left,-1,-1; down_right,1,-1).

% derive the kind of spacial relation from synonyms and offset
is(A, R1, B) :- is(A, R2, B), synonyms(R1, R2).
is(A, R1, B) :- is(B, R2, A), offset(R2,X,Y), offset(R1,-X,-Y).

% derive the location of every object
% the search space of X or Y coordinate is within -100 and 100
(to avoid infinite loop in clingo when data has error)
nums(-100..100).

location(A, Xa, Ya) :- location(B, Xb, Yb), nums(Xa), nums(Ya),
is(A, Kind, B), offset(Kind, Dx, Dy), Xa-Xb=Dx, Ya-Yb=Dy.

location(B, Xb, Yb) :- location(A, Xa, Ya), nums(Xb), nums(Yb),
is(A, Kind, B), offset(Kind, Dx, Dy), Xa-Xb=Dx, Ya-Yb=Dy.
Example 3:
### Instruction:
```

B.2 INSTRUCTION LENGTH 2+4

```
You are an text to Answer Set Program (ASP) translator.Users will
ask you questions in English and you will generate a ASP program
based on the provided  input.
Example 1:
### Instruction:
Generate the Answer Set Program(ASP) for the given input problem.
Each numbered point in the input will have one corresponding ASP
fact as output. Additionally, include some generic ASP rules that
will help the user in solving the problem. 1 F and T are next to
each other with F on the left and T on the right.
2 U and F are parallel, and U is on top of F.
3 What is the relation of the agent U to the agent F?

### Response:
% we generate one fact for each input sentence:
% 1 F and T are next to each other with F on the left and T on the
% right.
right("T", "F").

% 2 U and F are parallel, and U is on top of F.
top("U", "F").

% 3 What is the relation of the agent U to the agent F?
query("U", "F").

% Now we consider the answer set rules that will help us arrive at
% the final answer:
% assume the 2nd queried object is at location (0,0)
location(Q2, 0, 0) :- query(_, Q2).

% extract answer relation R such that the offset (Ox,Oy) of R is in
% the same direction of (X,Y)
answer(R) :- query(Q1, _), location(Q1, X, Y), offset(R, Ox, Oy),
```

```
Ox=-1: X$<$0; Ox=0: X=0; Ox=1: X$>$0; Oy=-1: Y$<$0; Oy=0: Y=0;
Oy=1: Y$>$0.

% general format translation, which can also be easily done in
%python script (this is not needed if we directly extract the
% general form in the beginning as in bAbI task4)
is(A, top, B) :- top(A, B).
is(A, top, B) :- up(A, B).
is(A, down, B) :- down(A, B).
is(A, left, B) :- left(A, B).
is(A, right, B) :- right(A, B).
is(A, top_left, B) :- top_left(A, B).
is(A, top_right, B) :- top_right(A, B).
is(A, down_left, B) :- down_left(A, B).
is(A, down_right, B) :- down_right(A, B).
is(A, east, B) :- east(A, B).
is(A, west, B) :- west(A, B).
is(A, south, B) :- south(A, B).
is(A, north, B) :- north(A, B).

% synonyms
synonyms(north, northOf; south, southOf; west, westOf; east,
eastOf; top, northOf; down, southOf; left, westOf; right,
eastOf).
synonyms(A, B) :- synonyms(B, A).
synonyms(A, C) :- synonyms(A, B), synonyms(B, C), A!=C.

% define the offsets of 8 spacial relations
offset(overlap,0,0; top,0,1; down,0,-1; left,-1,0; right,1,0;
top_left,-1,1; top_right,1,1; down_left,-1,-1; down_right,1,-1).

% derive the kind of spacial relation from synonyms and offset
is(A, R1, B) :- is(A, R2, B), synonyms(R1, R2).
is(A, R1, B) :- is(B, R2, A), offset(R2,X,Y), offset(R1,-X,-Y).

% derive the location of every object
% the search space of X or Y coordinate is within -100 and 100
% (to avoid infinite loop in clingo when data has error)
nums(-100..100).

location(A, Xa, Ya) :- location(B, Xb, Yb), nums(Xa), nums(Ya),
is(A, Kind, B), offset(Kind, Dx, Dy), Xa-Xb=Dx, Ya-Yb=Dy.

location(B, Xb, Yb) :- location(A, Xa, Ya), nums(Xb), nums(Yb),
is(A, Kind, B), offset(Kind, Dx, Dy), Xa-Xb=Dx, Ya-Yb=Dy.
Example 2:
### Instruction:
Generate the Answer Set Program(ASP) for the given input problem.
Each numbered point in the input will have one corresponding ASP
fact as output. Additionally, include some generic ASP rules that
will help the user in solving the problem.
1 C and M are both there with the object C above the object M.
2 Z is at the bottom and Y is on the top.
3 Z is at a 45 degree angle to M, in the upper lefthand corner.
4 Y is placed at the lower left of G.
5 What is the relation of the agent Z to the agent C?
### Response:
% we generate one fact for each input sentence:
% 1 C and M are both there with the object C above the object M.
```

```
top("C", "M").

% 2 Z is at the bottom and Y is on the top.
down("Z", "Y").

% 3 Z is at a 45 degree angle to M, in the upper lefthand corner.
top_left("Z", "M").

% 4 Y is placed at the lower left of G.
down_left("Y", "G").

% 5 What is the relation of the agent Z to the agent C?
query("Z", "C").

% Now we consider the answer set rules that will help us arrive
% at the final answer:
% assume the 2nd queried object is at location (0,0)
location(Q2, 0, 0) :- query(_, Q2).

% extract answer relation R such that the offset (Ox,Oy) of R is
% in the same direction of (X,Y)
answer(R) :- query(Q1, _), location(Q1, X, Y), offset(R, Ox, Oy),
Ox=-1: X$<$0; Ox=0: X=0; Ox=1: X$>$0; Oy=-1: Y$<$0; Oy=0: Y=0;
Oy=1: Y$>$0.

% general format translation, which can also be easily done in
% python script (this is not needed if we directly extract the
% general form in the beginning as in bAbI task4)
is(A, top, B) :- top(A, B).
is(A, top, B) :- up(A, B).
is(A, down, B) :- down(A, B).
is(A, left, B) :- left(A, B).
is(A, right, B) :- right(A, B).
is(A, top_left, B) :- top_left(A, B).
is(A, top_right, B) :- top_right(A, B).
is(A, down_left, B) :- down_left(A, B).
is(A, down_right, B) :- down_right(A, B).
is(A, east, B) :- east(A, B).
is(A, west, B) :- west(A, B).
is(A, south, B) :- south(A, B).
is(A, north, B) :- north(A, B).

% synonyms
synonyms(north, northOf; south, southOf; west, westOf; east,
eastOf; top, northOf; down, southOf; left, westOf; right,
eastOf).
synonyms(A, B) :- synonyms(B, A).
synonyms(A, C) :- synonyms(A, B), synonyms(B, C), A!=C.

% define the offsets of 8 spacial relations
offset(overlap,0,0; top,0,1; down,0,-1; left,-1,0; right,1,0;
top_left,-1,1; top_right,1,1; down_left,-1,-1; down_right,1,-1).

% derive the kind of spacial relation from synonyms and offset
is(A, R1, B) :- is(A, R2, B), synonyms(R1, R2).
is(A, R1, B) :- is(B, R2, A), offset(R2,X,Y), offset(R1,-X,-Y).

% derive the location of every object
% the search space of X or Y coordinate is within -100 and 100
```

```
% (to avoid infinite loop in clingo when data has error)
nums(-100..100).

location(A, Xa, Ya) :- location(B, Xb, Yb), nums(Xa), nums(Ya),
is(A, Kind, B), offset(Kind, Dx, Dy), Xa-Xb=Dx, Ya-Yb=Dy.

location(B, Xb, Yb) :- location(A, Xa, Ya), nums(Xb), nums(Yb),
is(A, Kind, B), offset(Kind, Dx, Dy), Xa-Xb=Dx, Ya-Yb=Dy.
Example 3:
### Instruction:
```