

Answer Set Programming and Large Language Models interaction with YAML: Second Report

Mario Alviano, Lorenzo Grillo, Fabrizio Lo Scudo and Luis Angel Rodriguez Reiners*

DeMaCS, University of Calabria, 87036 Rende (CS), Italy

Abstract

Answer Set Programming (ASP) and Large Language Models (LLMs) have emerged as powerful tools in Artificial Intelligence, each offering unique capabilities in knowledge representation and natural language understanding, respectively. In this paper, we combine the strengths of the two paradigms to couple the reasoning capabilities of ASP with the attractive natural language processing capability of LLMs. We introduce a YAML-based format for specifying prompts, allowing users to configure the behavior of the system and to encode domain-specific background knowledge. Input prompts are processed by LLMs to generate relational facts, which are then processed by ASP rules for knowledge reasoning, and finally the ASP output is mapped back to natural language by LLMs, so to provide a captivating user experience.

Keywords

Answer Set Programming, Large Language Models, Knowledge Representation, Natural Language Generation

1. Introduction

Large Language Models (LLMs) and Answer Set Programming (ASP) represent two distinct but complementary paradigms in Artificial Intelligence (AI). LLMs, such as GPT [1], PaLM [2], and LLaMa [3], have transformed natural language processing (NLP) by achieving unprecedented levels of fluency and understanding in textual data. In contrast, ASP [4, 5], a declarative programming paradigm rooted in logic programming under answer set semantics [6], excels in knowledge representation and logical reasoning, making it fundamental for AI systems that require robust inference capabilities. Individually, LLMs and ASP offer unique advantages within their respective domains. LLMs excel at various NLP tasks [7, 8], such as language generation, summarization, and sentiment analysis, utilizing deep learning and extensive pre-trained language models. In contrast, ASP equips AI systems with robust reasoning capabilities, enabling them to process complex knowledge bases, draw logical conclusions, and tackle intricate combinatorial problems. This makes ASP particularly effective in decision-making scenarios, including planning and scheduling [9, 10], as well as diagnosis and configuration [11, 12]. Recognizing the complementary strengths of LLMs' linguistic abilities and ASP's reasoning capabilities, this paper proposes an approach that leverages the synergies between these two paradigms, inspired by recent works in the literature [13, 14]. Our objective is to develop a unified system that seamlessly integrates natural language understanding with logical inference, allowing AI applications to adeptly handle the complex interplay between textual data and logical structures.

In this paper, we propose a comprehensive framework that combines LLMs and ASP, exploiting the strengths of both paradigms to mitigate their respective weaknesses. We outline a method for encoding specific domain knowledge into input prompts using a YAML-based format, which allows LLMs to produce relational facts that are utilized by ASP for reasoning. The reasoned outcomes of ASP are then translated back into natural language by LLMs, creating an engaging user experience and improving the clarity of the results. An overview of the main pipeline addressed by our system is shown in Figure 1.

Workshop on Symbolic and Neuro-Symbolic Architectures for Intelligent Robotics Technology (SYNERGY) co-located with the 21st International Conference on Principles of Knowledge Representation and Reasoning (KR2024), November 2–8, 2024, Hanoi, Vietnam.

*Corresponding author.

✉ mario.alviano@unical.it (M. Alviano); luis.reiners@unical.it (L. A. Rodriguez Reiners)

🌐 <https://alviano.net/> (M. Alviano)

🆔 0000-0002-2052-2063 (M. Alviano); 0009-0000-1808-9910 (L. A. Rodriguez Reiners)

© 2024 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).



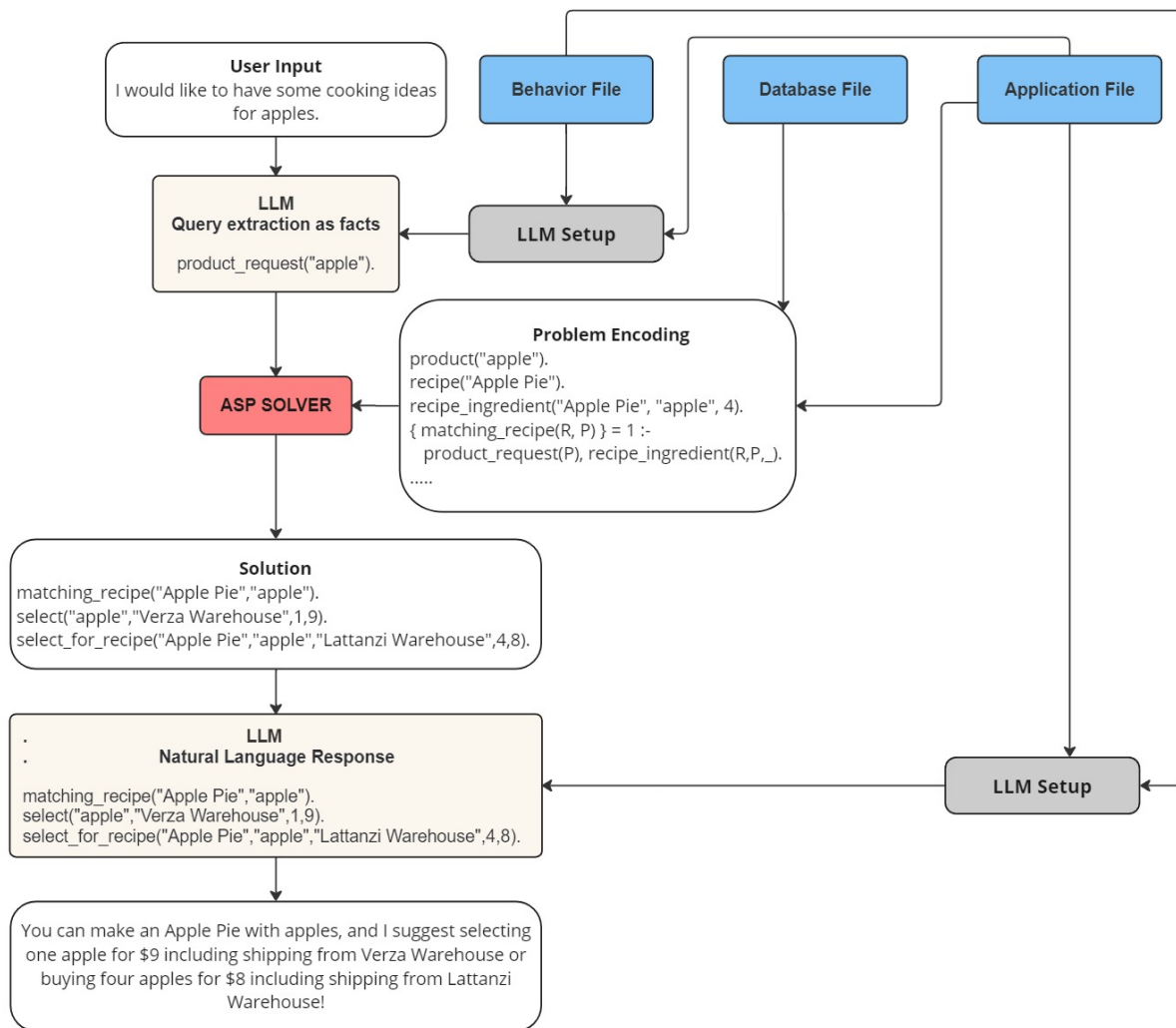


Figure 1: Graphical representation of the LLMASP pipeline. The system receives two YAML files—one describing system behavior and the other related to the specific application being developed—together with a relational database and a user query in natural language. In this setup, the behavior YAML file offers initial prompts for the LLM, which are then filled in with details from the application YAML file and the user’s input. The LLM uses these finalized prompts to extract information from the user input, converting it into factual representations. These facts are integrated with an ASP program (knowledge base) and processed by an ASP solver to produce an answer set. Finally, the initial prompts are once again utilized to translate the returned answer set back into natural language, delivering a coherent (and easier to read) response to the user.

The system takes in input two YAML files, one defining the behavior of the system and one specific of the application under development, a database (relational facts), and a request from the user expressed in natural language. The behavior file contains partial prompts for the LLM, which are completed by the data stored in the application file and the user input. The complete prompts ask to extract data from the user input and represent them in the form of facts. Facts are then combined with a knowledge base (an ASP program) and processed by an ASP solver to obtain an answer set. The answer set is then combined with the prompts obtained by the two YAML files to produce an answer in natural language for the user.

As a possible application of the technology we want to develop, imagine a bustling online marketplace that introduces a new, innovative way for users to interact with its platform: a chatbot. Traditionally, users navigate the website using a mouse and keyboard to add or remove items from their shopping basket. However, this marketplace now offers an alternative that caters to the growing number of mobile phone users who prefer a more intuitive interaction. Instead of manually clicking through

menus, users can simply converse with the chatbot using natural language to achieve the same results. For instance, a user might say, “Add two red t-shirts to my cart” or “Remove the coffee maker from my basket,” and the chatbot seamlessly executes these commands. This voice-to-text interaction is particularly convenient for mobile users who find typing cumbersome or impractical on smaller screens. By leveraging the chatbot, the marketplace not only enhances the shopping experience but also makes it more accessible and user-friendly for everyone.

In summary, our prototype system facilitates a smooth interaction between an LLM and an ASP solver. When an input text is received, the system interacts with the LLM using a set of predefined prompts whose overall structure is enhanced with additional sentences, as described in the YAML format. The LLM-generated responses are then processed to extract factual data, which is fed into the ASP solver. The ASP solver performs logical reasoning on the combination of input data and the specified knowledge base, producing an answer set that encapsulates reasoned conclusions and insights. To translate the logical output into comprehensible natural language, the system re-engages the LLM, using predefined prompts and enriched specifications to convert the answer set into coherent sentences. Lastly, the sentences are summarized by the LLM, offering users a concise and insightful overview of the derived conclusions.

2. Background

2.1. Large Language Models

Large Language Models (LLMs) are sophisticated artificial intelligence systems designed to understand and generate human-like text. These models are typically based on deep learning architectures, such as Transformers, and are trained on vast amounts of text data to learn complex patterns and structures of language. In this article, LLMs are used as black box operators on text (functions that take text in input and produce text in output). At each interaction with a LLM, the generated text is influenced by all previously processed text, and randomness is involved in the process. The text in input is called *prompt*, and the text in output is called *generated text* or *response*.

Example 1. Let us consider the following prompt:

Encode as Datalog facts the following sentences: I would like some cooking ideas with apples as dessert, and something with meat as a main plate.

A response produced by Meta Llama 3 is reported below.

```
want(dessert, apples). want(main_plate, meat).
dish_type(apples, dessert). dish_type(meat, main_plate).
```

It is a very good starting point, but the LLM must be instructed on a specific format to use in encoding facts. We aim at gaining more control on the output produced by the LLM. ■

2.2. Answer Set Programming

All sets and sequences considered in this paper are finite. Let \mathbf{P} , \mathbf{C} , \mathbf{V} be fixed nonempty sets of *predicate names*, *constants* and *variables*. Predicates are associated with an *arity*, a non-negative integer. A *term* is any element in $\mathbf{C} \cup \mathbf{V}$. An *atom* is of the form $p(\bar{t})$, where $p \in \mathbf{P}$, and \bar{t} is a possibly empty sequence of terms. A *literal* is an atom possibly preceded by the default negation symbol not; they are referred to as positive and negative literals. An *aggregate* is of the form

$$\#\text{SUM}\{t_a, \bar{t}' : p(\bar{t})\} \odot t_g \quad (1)$$

where $\odot \in \{<, \leq, \geq, >, =, \neq\}$ is a binary comparison operator, $p \in \mathbf{P}$, \bar{t} and \bar{t}' are possibly empty sequences of terms, and t_a and t_g are terms. Let $\#\text{COUNT}\{\bar{t}' : p(\bar{t})\} \odot t_g$ be syntactic sugar for $\#\text{SUM}\{1, \bar{t}' : p(\bar{t})\} \odot t_g$. A *choice* is of the form

$$t_1 \leq \{atoms\} \leq t_2 \quad (2)$$

where $atoms$ is a possibly empty sequence of atoms, and t_1, t_2 are terms. Let \perp be syntactic sugar for $1 \leq \{\} \leq 1$. A *rule* is of the form

$$head \text{ :- } body. \quad (3)$$

where $head$ is an atom or a choice, and $body$ is a possibly empty sequence of literals and aggregates. (Symbol :- is omitted if $body$ is empty. The head is usually omitted if it is \perp , and the rule is called *constraint*.) For a rule r , let $H(r)$ denote the atom or choice in the head of r ; let $B^\Sigma(r)$, $B^+(r)$ and $B^-(r)$ denote the sets of aggregates, positive and negative literals in the body of r ; let $B(r)$ denote the set $B^\Sigma(r) \cup B^+(r) \cup B^-(r)$.

Example 2. Let us consider the following rules:

```
index(1). index(2). index(3). succ(1,2). succ(2,3). empty(2,2).
grid(X,Y) :- index(X), index(Y).
0 <= {assign(X,Y)} <= 1 :- grid(X,Y), not empty(X,Y).
:- #count{A,B : assign(A,B)} != 5.
:- assign(X,Y), succ(X,X'), assign(X',Y').
:- assign(X,Y), succ(Y,Y'), assign(X',Y').
```

The first line above comprises rules with atomic heads and empty bodies (also called *facts*). After that there is a rule with atomic head and nonempty body (also called a *definition*), followed by a choice rule, and three constraints. If r is the choice rule, then $H(r) = \{0 <= \{\text{assign}(X, Y)\} <= 1\}$, $B^+(r) = \{\text{grid}(X, Y)\}$, $B^-(r) = \{\text{not empty}(X, Y)\}$, and $B^\Sigma(r) = \emptyset$. ■

A variable X occurring in $B^+(r)$ is a *global variable*. Other variables occurring among the terms \bar{t} of some aggregate in $B^\Sigma(r)$ of the form (1) are *local variables*. And any other variable occurring in r is an *unsafe variable*. A *safe rule* is a rule with no *unsafe variables*. A *program* Π is a set of safe rules. A substitution σ is a partial function from variables to constants; the application of σ to an expression E is denoted by $E\sigma$. Let $instantiate(\Pi)$ be the program obtained from rules of Π by substituting global variables with constants in \mathbf{C} , in all possible ways; note that local variables are still present in $instantiate(\Pi)$. The Herbrand base of Π , denoted $base(\Pi)$, is the set of ground atoms (i.e., atoms with no variables) occurring in $instantiate(\Pi)$.

Example 3. Variables A, B are local, and all other variables are global. Let Π be the program comprising all rules in Example 2 (which are safe). If $\mathbf{C} = \{1, 2, 3\}$, then $instantiate(\Pi)$ comprises, among others, the following rules:

```
grid(1,1) :- index(1), index(1).
0 <= {assign(1,1)} <= 1 :- grid(1,1), not empty(1,1).
:- #count{A,B : assign(A,B)} != 5.
```

Note that the local variables A, B are still present in the last rule above. ■

An *interpretation* is a set of ground atoms. For an interpretation I , relation $I \models \cdot$ is defined as follows: for a ground atom $p(\bar{c})$, $I \models p(\bar{c})$ if $p(\bar{c}) \in I$, and $I \models \text{not } p(\bar{c})$ if $p(\bar{c}) \notin I$; for an aggregate α of the form (1), the aggregate set of α w.r.t. I , denoted $aggset(\alpha, I)$, is $\{\langle t_a, \bar{t}' \rangle \sigma \mid p(\bar{t})\sigma \in I, \text{ for some substitution } \sigma\}$, and $I \models \alpha$ if $(\sum_{\langle c_a, \bar{c}' \rangle \in aggset(\alpha, I)} c_a) \odot t_g$ is a true expression over integers; for a choice α of the form (2), $I \models \alpha$ if $t_1 \leq |I \cap atoms| \leq t_2$ is a true expression over integers; for a rule r with no global variables, $I \models B(r)$ if $I \models \alpha$ for all $\alpha \in B(r)$, and $I \models r$ if $I \models H(r)$ whenever $I \models B(r)$; for a program Π , $I \models \Pi$ if $I \models r$ for all $r \in instantiate(\Pi)$.

For a rule r of the form (3) and an interpretation I , let $expand(r, I)$ be the set $\{p(\bar{c}) \text{ :- } body. \mid p(\bar{c}) \in I \text{ occurs in } H(r)\}$. The *reduct* of Π w.r.t. I is the program comprising the expanded rules of $instantiate(\Pi)$ whose body is true w.r.t. I , that is, $reduct(\Pi, I) := \bigcup_{r \in instantiate(\Pi), I \models B(r)} expand(r, I)$. An *answer set* of Π is an interpretation I such that $I \models \Pi$ and no $J \subset I$ satisfies $J \models reduct(\Pi, I)$.

Example 4 (Continuing Example 3). Program Π has two answer sets, which comprise the following instances of predicate `assign/2`:

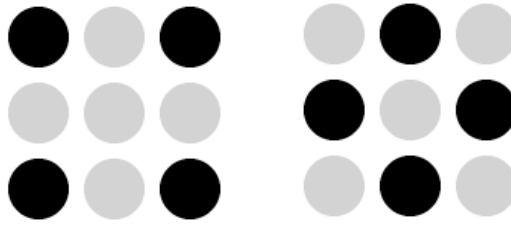


Figure 2: The solutions encoded by the two answer sets of program II in Example 2.

1. `assign(1,2), assign(2,1), assign(2,3), assign(3,2);`
2. `assign(1,1), assign(1,3), assign(3,1), assign(3,3).`

Figure 2 shows the solutions encoded by the two answer sets. ■

The language of ASP supports several other constructs, among them binary built-in relations (i.e., `<`, `<=`, `>=`, `>`, `=`, `!=`) which are interpreted naturally.

2.3. YAML

YAML (YAML Ain't Markup Language; <https://yaml.org/spec/1.2.2/>) is a human-readable data serialization format commonly used for configuration files, data exchange, and representation of structured data. YAML is designed to be easily readable by humans and is commonly used in software development for configuration files, data storage, and data interchange between different programming languages. YAML uses indentation to denote nesting and relies on simple syntax rules, such as key-value pairs and lists, to represent structured data. YAML is often preferred for its simplicity, readability, and flexibility compared to other serialization formats like JSON and XML. In this article, we focus on the following restricted fragment: A *scalar* is any number or string (possibly quoted). A *block sequence* is a sequence of entries, each one starting by a dash followed by a space. A *mapping* is a sequence of key-value pairs, each pair using a colon and space as separator, where keys and values are scalars. A scalar can be written in *block notation* using the prefix `|` (if not a key). Lines starting with `#` are *comments*.

Example 5. Below is a YAML document:

```
name: LLMASP
papers:
- CILC 2024
- SYNERGY 2024
description: |
  LLMASP combines ASP and LLMs...
  LLMs are used to extract data...
```

It encodes a mapping with keys `name`, `papers` and `description`. Key `name` is associated with the scalar `LLMASP`. Key `papers` is associated with the list `[CILC 2024, SYNERGY 2024]`. Key `description` is associated with a scalar in block notation. ■

3. LLMASP Configuration

The interaction between LLMs and ASP is achieved by means of two YAML specification files defined in this section. The first YAML file specifies global behavior settings for LLMASP, as tone, style and general instructions for the LLM, while the second YAML file contains domain-specific guidelines, as a description of the context and mappings between each ASP fact and its corresponding natural language translation.

Behavior File. The behavior file comprises two keys, namely preprocessing and postprocessing. The preprocessing section contains the following properties:

- `init`, whose value is used to provide general instructions to the LLM;
- `context`, whose value must include the string `§context§`, to be combined with contextual information regarding an application of interest;
- `mapping`, whose value must include the strings `§input§`, `§instructions§` and `§atom§`, to be combined with the instructions on how to extract specific atoms from the user input.

The postprocessing section is similar but the mapping must include the strings `§facts§`, `§instructions§` and `§atom§`, to be combined with the instructions on how to map specific atoms (from the facts in the answer set) to natural language. Additionally, the postprocessing section specifies how to summarize the `§responses§` collected by the postprocessing mapping operation.

Example 6. Below is the behavior file encoding the hard-coded system presented at the *39th Italian Conference on Computational Logic* [15].

preprocessing:

```
init: |
  You are a Natural Language to Datalog translator. To translate your
  input to Datalog, you will be asked a sequence of questions. The
  answers are inside the user input provided with
  [USER_INPUT]input[/USER_INPUT] and the format is provided with
  [ANSWER_FORMAT]predicate(terms).[/ANSWER_FORMAT]. Predicate is a
  lowercase string (possibly including underscores). Terms is a
  comma-separated list of either double quoted strings or integers.
  Be sure to control the number of terms in each answer!
  An answer MUST NOT be answered if it is not present in the user input.
  Remember these instructions and don't say anything!
context: |
  Here is some context that you MUST analyze and remember.
  §context§
  Remember this context and don't say anything!
mapping:
  [USER_INPUT]§input§[/USER_INPUT]
  §instructions§
  [ANSWER_FORMAT]§atom§.[/ANSWER_FORMAT]
```

postprocessing:

```
init: |
  You are now a Datalog to Natural Language translator.
  You will be given relational facts and mapping instructions.
  Relational facts are given in the form [FACTS]atoms[/FACTS].
  Remember these instructions and don't say anything!
context: |
  Here is some context that you MUST analyze and remember.
  §context§
  Remember this context and don't say anything!
mapping: |
  [FACTS]§facts§[/FACTS]
  Each fact matching §atom§ must be interpreted as follows:
  §instructions§
summarize: |
  Summarize the following responses:
  §responses§
```

Application File. The second YAML file, the application file, contains three sections, namely preprocessing, knowledge base, and postprocessing. The values associated with preprocessing

and postprocessing are mappings where keys are either atoms or the special value `_` (used for providing a context), and values are scalars. The value associated with `knowledge base` is an ASP program.

Example 7. Below is an example about an assistant for a marketplace aimed at helping customers with fulfilling orders to match a given recipe.

preprocessing:

- `_`: The marketplace offers food products. Products and product preferences will be talked about.
- `product_request("product")`: List all the products mentioned or requested. If a product is named must be listed. Ignore plural, always write the product name in singular.
- `product_request("product", quantity)`: List all the products mentioned or requested if and only if they have a quantity associated. Ignore plural, always write the product name in singular.

knowledge base: |

```
#script(python)
def min(a, b): return a if a < b else b
#end.

% guess selection of products
{select(P,W,Q',T) : Q' = 1..@min(Q,R), T=Q'*PP+C} <= 1 :-
    product_request(P,R), product_price(P,W,PP), warehouse(W),
    warehouse_shipping_cost(W,C), product_in_warehouse(P,W,Q).
:- product_request(P,R), #sum{Q,W : select(P,W,Q,_)} != R.

% minimize total cost considering price and shipping
:~ select(P, W, Q, T).[T@1]

% minimize shipping cost
:~ warehouse_shipping_cost(W,C), warehouse_free_shipping(W,T),
    select(_,W,Q,_), Q > 0,
    #sum{Q' * Price,P : select(P,W,Q',_), product_price(P,W,Price)} < T.
[C@3, W]

%guess selection of products for a recipe
{select_for_recipe(R,P,W,Q',PP') : Q' = 1..@min(Q,A), PP'=PP*Q'} <= 1 :-
    product_request(P), recipe(R), recipe_ingredient(R,P,A),
    product_in_warehouse(P,W,Q), product_price(P, W, PP), warehouse(W).
:- product_request(P), #sum{Q,R,W : select_for_recipe(R,P,W,Q,_);
    -A,R : recipe_ingredient(R, P, A) } != 0.

{matching_recipe(R,P)}=1 :- product_request(P), recipe_ingredient(R,P,_).

#show select/4.
#show select_for_recipe/5.
#show matching_recipe/2.
```

postprocessing:

- `_`: You are an assistant in an online marketplace, which is talking directly to a customer. Your priority is to make sales, so the main goal of all responses must be to make a sale. Your answers must be customer-oriented. Do not mention any product that is not explicitly provided to you before. Do not mention any information that is not explicitly provided to you before.
- `select_for_recipe("recipe", "product", "warehouse", "quantity", "total")`: Say to buy "quantity" of "product" for "total" including shipping costs from "warehouse" if the customer desire to make "recipe". Do not forget the quantity.

- `select("product", "warehouse", "quantity", "total")`: Suggest to select "quantity" of the "product" for a cost of "total" from the "warehouse". This price include the shipping costs.
- `matching_recipe("recipe", "product")`: Suggest that "recipe" can be done with "product".

The preprocessing aims at extracting data or a query about products. The data is combined with the knowledge base and a database containing all information regarding product costs, for example

```
product("apple").
warehouse_shipping_cost("Lattanzi Warehouse", 7).
warehouse_shipping_cost("Verza Warehouse", 5).
warehouse_shipping_fee("Lattanzi Warehouse", 5).
warehouse_shipping_fee("Verza Warehouse", 3).
product_in_warehouse("apple", "Lattanzi Warehouse", 20).
product_in_warehouse("apple", "Verza Warehouse", 3).
product_price("apple", "Lattanzi Warehouse", 2).
product_price("apple", "Verza Warehouse", 4).
recipe_ingredient("Apple Pie", "apple", 4).
```

The combined knowledge is used to identify bests buying offers and products matching a recipe. Finally, the postprocessing prompts aim at producing a paragraph reporting the found results, possible products for adding to a shopping cart with related costs. ■

Notation. For a, b, c being strings, let $a[b \mapsto c]$ denote the string obtained from a by replacing all occurrences of b with c . Given a behavior file B , let $pre_B(\alpha)$ be the value associated with α in the preprocessing mapping of B , where α is among `init`, `context` and `mapping`. Similarly, let $post_B(\alpha)$ be the value associated with α in the postprocessing mapping of B , where α is among `init`, `context` and `mapping`. For a YAML application file A , let $pre_A(\alpha)$ be the value associated with α in the preprocessing mapping, where α is either an atom or `_`. Similarly, let $post_A(\alpha)$ be the value associated with α in the postprocessing mapping, where α is either an atom or `_`. Finally, let kb_A be the ASP program in background knowledge.

4. LLMASP Pipeline

The architecture of LLMASP is shown in Figure 1. In this section, we discuss its core components and illustrate its potential through a practical example.

LLMASP takes in input the two YAML file introduced in Section 3, namely the behavior file B and the application file A , together with a database file D (comprising facts) and a request text T (expressed by the user in natural language). The system undergoes a sequence of interactions with an LLM to populate a set F of facts and a set R of responses, which are initially empty. Specifically, LLMASP execute the following procedure:

- P1.** The LLM is invoked with the prompt $pre_B(\text{init})$
- P2.** If $pre_A(_)$ is defined, the LLM is invoked with the following prompt:
 $pre_B(\text{context})[\$context\$ \mapsto pre_A(_)]$.
- P3.** For each atom α such that $pre_A(\alpha)$ is defined, the LLM is invoked with the prompt
 $pre_B(\text{mapping})[\$input\$ \mapsto T][\$atom\$ \mapsto \alpha][\$instructions\$ \mapsto pre_A(\alpha)]$. Facts in the response are collected in F . Everything else is ignored.
- P4.** An answer set of $kb_A \cup \{\alpha. \mid \alpha \in D \cup F\}$ is searched, say I . If an answer set does not exist, the process terminates with a failure.
- P5.** The LLM is reset and invoked with the prompt $post_B(\text{init})$.
- P6.** If $post_A(_)$ is defined, the LLM is invoked with the following prompt:
 $post_B(\text{context})[\$context\$ \mapsto post_A(_)]$.

- P7.** For each atom $p(\bar{t})$ such that $post_A(p(\bar{t}))$ is defined, the LLM is invoked with the prompt $post_B(\text{mapping})[\$facts\$ \mapsto I][\$atom\$ \mapsto \alpha][\$instructions\$ \mapsto post_A(\alpha)]$. Responses are collected in R .
- P8.** The LLM is invoked with the prompt $post_B(\text{summarize})[\$responses\$ \mapsto R]$. The response is provided in output.

In our implementation, the LLM is the latest iteration of the Meta’s open-source project LLama¹, which at the time of writing this paper has reached the third version, and the ASP system used is CLINGO [16].

Example 8. Let A be the application file given in Example 7, and B be the behavior file reported in Example 6. Let T be the following text:

I would like some cooking ideas with apples for dessert and a main plate with meat.

The LLM is invoked with the initial fixed prompt of **P1**, and then with the prompt of **P2**:

Here is some context that you MUST analyze and remember.
The marketplace offers food products. Products and product preferences will be talked about.
Remember this context and don’t say anything!

After that, the LLM is invoked two times with the prompt of **P3** to populate set F . For example,

```
[USER_INPUT]I would like some cooking ideas with apples for dessert and a main plate with meat.
[/USER_INPUT]
List all the products mentioned or requested. If a product is named MUST be listed. Ignore plural,
always write the product name in singular.
[ANSWER_FORMAT]product_request("product").[/ANSWER_FORMAT]
```

The LLM may provide the response

```
[ANSWER_FORMAT]product_request("apple").[/ANSWER_FORMAT]
[ANSWER_FORMAT]product_request("meat").[/ANSWER_FORMAT]
```

from which the following facts are extracted and added to F :

```
product_request("apple"). product_request("meat").
```

Once all facts are collected, the knowledge base is used to search an answer set (**P4**), say one containing the following atoms:

```
matching_recipe("Apple Pie","apple").
matching_recipe("Meat and Onion","meat").
select("apple","Verza Warehouse",1,9).
select("meat","Rellman Warehouse",1,11).
select_for_recipe("Apple Pie","apple","Lattanzi Warehouse",4,8).
select_for_recipe("Meat and Onion","meat","Rellman Warehouse",5,35).
```

The LLM is now invoked with the prompts of **P5–P7**. In particular, the prompt of **P7** is the following:

```
[FACTS]select_for_recipe("Apple Pie","apple","Lattanzi Warehouse",4,8).
select_for_recipe("Meat and Onion","meat","Rellman Warehouse",5,35). [/FACTS]
```

Each fact matching `select_for_recipe("recipe", "warehouse", "quantity", "total")` must be interpreted as follows: Say to buy "quantity" of "product" for "total" including shipping costs from "warehouse" if the customer desire to make "recipe". Do not forget the quantity.

The response, say

Consider buying 4 apples for 8 from Lattanzi Warehouse if you desire to make Apple Pie.
Consider buying 5 meat for 35 from Rellman Warehouse if you desire to make Meat and Onion.

¹<https://llama.meta.com/llama3/>

is added to R . Finally, the LLM is invoked with the prompt of **P8**:

Summarize the following responses: You might enjoy making an Apple Pie with our apples. Consider preparing a Meat and Onion dish using our meat. Suggest to select 1 of the apple for a cost of 9 from the Verza Warehouse. This price includes the shipping costs. Suggest to select 1 of the meat for a cost of 11 from the Rellman Warehouse. This price includes the shipping costs. Consider buying 4 apples for 8 from Lattanzi Warehouse if you desire to make Apple Pie. Consider buying 5 meat for 35 from Rellman Warehouse if you desire to make Meat and Onion.

The response, say

Why not try making an Apple Pie with our delicious apples, or a savory Meat and Onion dish using our fresh meat? You can get 1 apple from Verza Warehouse for \$9 or 4 from Lattanzi Warehouse for \$8, and pair it with 1 meat from Rellman Warehouse for \$11 or 5 for \$35!

is provided in output. ■

The main benefit of the new LLMASP version compared to the initial proposal introduced at CILC 2024 [15] is the ease of testing various behavior files. This feature allows for experimentation with different prompts to manage the interaction and behavior of the LLM.

Example 9. Let us consider again the application file reported in Example 7, and the following behavior file (different from the one reported in Example 6):

preprocessing:

init: |

As an ASP translator, your primary task is to convert natural language descriptions, provided in the format [INPUT]input[/INPUT], into precise ASP code, outputting in the format [OUTPUT]predicate(terms).[/OUTPUT]. Focus on identifying key entities and relationships to create facts (e.g., [INPUT]Alice is happy[/INPUT] becomes [OUTPUT]happy(alice).[/OUTPUT]), [INPUT]Bob owns a car[/INPUT] becomes [OUTPUT]owns(bob, car)[/OUTPUT], [INPUT]The sky is blue[/INPUT] becomes [OUTPUT]color(sky, blue)[/OUTPUT], and [INPUT]Cats are mammals[/INPUT] becomes [OUTPUT]mammal(cat)[/OUTPUT]. Ensure that the natural language intent is accurately and logically reflected in the ASP code. Maintain semantic accuracy by ensuring logical consistency and correctly reflecting the natural language intent in your ASP code. Remember these instructions and don't say anything!

context: |

Here is some context that you MUST analyze and remember.

§context§

Remember this context and don't say anything!

mapping:

[INPUT]§input§[/INPUT]

§instructions§

[OUTPUT]§atom§.[/OUTPUT]

postprocessing:

init: |

As an ASP to natural language translator, you will convert ASP facts provided in the format [FACTS]atoms[/FACTS] into clear natural language statements using predefined mapping instructions. For example, [FACTS]happy(alice)[/FACTS] should be translated to "Alice is happy," [FACTS]friend(alice, bob)[/FACTS] to "Alice is friends with Bob," and [FACTS]owns(bob, car)[/FACTS] to "Bob owns a car." Ensure each fact is accurately and clearly represented in natural language, maintaining the integrity of the original information.

```

Remember these instructions and don't say anything!
context: |
Here is some context that you MUST analyze and remember.
$context$
Remember this context and don't say anything!
mapping: |
[FACTS]$facts$[/FACTS]
Each fact matching $atom$ must be interpreted as follows:
$instructions$
summarize: |
Summarize the following responses:
$responses$

```

In this case the prompt used in $pre_B(\text{init})$ was enriched using *in-context learning* (ICL), a specific method of prompt engineering where demonstrations of the task are provided to the model as part of the prompt (in natural language) [17]. Considering the same input text as Example 8, after invoking the LLM with the prompt of **P3**, the provided response is

```

[OUTPUT]product_request("apple").[/OUTPUT]
[OUTPUT]product_request("apple", 0).[/OUTPUT]
[OUTPUT]product_request("meat").[/OUTPUT]
[OUTPUT]product_request("meat", 0).[/OUTPUT]

```

In this case no quantities were specify by the user input, however the model still generate a 0 quantity request for each product. ■

Continuing with the behavior file discussed in Example 9, let's now consider a different application: an intelligent assistant. Still within the context of a marketplace, this assistant will aid in managing inventory efficiently and effectively.

Example 10. Below is an assistant for monitoring inventory levels and to provide notifications when it is necessary to replenish products in the warehouse.

```

preprocessing:
- _: You are an assistant for logistics management. Products, warehouses and
  products stocks will be talked about.
- product_request("product", quantity):: List all the products mentioned or
  requested with a quantity associated. If no quantity is mentioned, assume 1.
  Ignore plural, always write the product name in singular.

```

```

knowledge_base: |
#script(python)
def min(a, b): return a if a < b else b
#end.

% guess selection of products
{select(P,W,Q',S) : Q' = 1..@min(Q,R), S = Q-Q'} <= 1 :-
  product_request(P,R),
  product_price(P,W,PP),
  warehouse(W),
  warehouse_shipping_cost(W,C),
  product_in_warehouse(P,W,Q).

% select the correct amount of products
:- product_request(P,R), #sum{Q,W : select(P,W,Q,_)} != R.

% minimize shipping cost
:~ warehouse_shipping_cost(W,C),
  warehouse_free_shipping(W,T),

```

```
select(_,W,Q,_), Q > 0,  
#sum{Q' * Price,P : select(P,W,Q',_), product_price(P,W,Price)} < T.  
[C@3, W]
```

```
#show select/4.
```

```
postprocessing:
```

```
- _: |  
    You are an assistant for logistics management in an online marketplace, which  
    is talking to a manager. Your priority is to keep track of product stocks and  
    inventory. Do not mention any product that is not explicitly provided to you  
    before. Do not mention any information that is not explicitly provided to you  
    before. If there is a 0 quantity associated with a product say that is out of  
    stock. Your answers should be suggestions for the manager to keep the  
    warehouses full of products. It must guide the manager to place the products  
    in the warehouses. Always limit your responses to a maximum of 100 characters.  
- select("product", "warehouse", "quantity", "stock").: |  
    Suggest that if "product" with "quantity" is selected from "warehouse", the  
    remaining "stock" quantity in the warehouse should be tracked.  
    Suggest to consider placing more products.
```

Given the following configuration, *A* be application file from Example 10 and *B* the behavior file from Example 9. Let *T* be the following input text:

What happens if we get and order of 22 apples?

Following the procedure from **P1-P8** the final output of the system will be

Restock apples in Verza Warehouse (out of stock) and consider restocking in Lattanzi Warehouse (only 1 left).

Using YAML configuration files allows for easy and human-readable editing of the system settings, enabling quick modifications without altering the underlying code. This flexibility facilitates testing different scenarios and generating varied outputs by simply updating the configuration files.

5. Related Work

In general, LLMs, including ChatGPT [1], PaLM [2], and LLaMa [3], demonstrate an impressive ability to utilize semantic relationships among tokens within natural language sequences of various lengths. They excel particularly in sequence-to-sequence (seq2seq) tasks, where a text input triggers a text output generated by the model. The applications of seq2seq models are diverse, covering areas such as machine translation [18], answering factual questions [19], executing basic arithmetic operations [20], text summarization [21], and chatbot functionalities [22]. It is reasonable to expect that these models could also excel in information extraction tasks.

The extraction of relational facts from text has been a continuous effort in Natural Language Processing. The capability to identify semantic relationships between entities in text allows one to convert unstructured raw text into structured data, which can then be utilized in various downstream tasks and applications, including the development of Knowledge Bases. In [23], the researchers introduce a sequence-to-sequence model built on BART [24] capable of executing end-to-end extraction for over 200 distinct relation categories. Their approach aligns with a growing trend where seq2seq Transformer models, such as BART or T5 [25], are employed in NLU tasks, including Entity Linking [26] and Semantic Role Labeling [27], by redefining them as seq2seq problems. Nowadays, although the most advanced Transformer-based models, such as GPT-3 and 4, have increased in size and demonstrate outstanding performance in numerous natural language processing tasks, these systems still exhibit limited reasoning capabilities despite the use of various prompting methods [28]. An alternative perspective, highlighted by [29], suggests that LLMs are suitable for what Kahneman refers to as System-1 thinking.

This is because they are designed to predict the next word in a sequence without deep comprehension of crucial reasoning concepts like causality, logic, and probability. Merging LLMs with logical reasoning into a neurosymbolic framework is an active research field. For example, considering ASP, a recent study [30] uses a dual phase architecture, incorporated in the NL2ASP tool, to produce ASP programs from natural language descriptions. NL2ASP utilizes neural machine translation to convert natural language into Controlled Natural Language (CNL) statements, which are then translated into ASP code via the CNL2ASP tool. Despite being a well-established approach to bridge the difference between the flexibility of natural language (NL) and the strictness of formal languages, the application of CNL [31] still faces limitations regarding its expressiveness [32]. In our approach, we bypass the use of intermediate languages like CNL. Instead, we implement a method where the users articulate their intents in NL, and arrange the information more systematically to align with LLM input preferences in order to minimize irrelevant responses. It is important to highlight that our task is less complex than the one tackled by [30]. Unlike [30], which aims to convert textual information into a formal knowledge base, our focus is directed toward a more straightforward task that involves the input and output of the ASP solver, without diving into deep semantic-demanding translation of the application domain. In addition, combining both methods could lessen the effort required from the end user.

Attempting a similar objective as [30], but using a prompt engineering strategy, [33] suggests carefully crafting prompts for an LLM to transform natural language descriptions into ASP incrementally. They demonstrate that, with just a handful of in-context learning instances, LLMs are capable of producing fairly complex answer set programs. The proposed pipeline initially identifies the relevant objects and their categories. Subsequently, it forms a predicate that delineates the relationships between objects from various categories. Using these derived data, the pipeline proceeds to build an ASP program following the Generate-Define-Test paradigm.

The main idea of [33] involves using LLM as an interface for answer set programming, thereby leveraging their language processing capabilities to convert NL descriptions into the declarative syntax of answer set programs. In contrast to approaches that rely on algorithms or machine learning methods, the authors observe that LLMs, when provided with an effective prompt, can produce fairly accurate answer set programs. Although we share with [33] the insight that effective prompt engineering is adequate for initial entity and relation extraction, we depart from their findings by asserting that the complexity of real-world applications necessitates a more structured approach to ASP than the pipeline proposed in their work. Additionally, because the authors appear not to make any specific assumptions about the problem description required to generate the final program with LLM, we aim to explore in future research the induction of logic programs from unstructured text using LLMs.

The closest related work to our goal that we found in the literature is [34]. In this work, the authors focus primarily on the Question Answering (QA) task, using an LLM to transform a problem description (comprising context S and query q) into atomic facts. Subsequently, the ASP solver processes these facts along with background knowledge represented as ASP rules to derive an answer a . This method does not rely on training datasets². Instead, only a small number of examples were used as *few-shot* contextual data for the LLM input. Supplying this meta-information about the problem context consequently facilitates the translation of natural language sentences into atomic facts, thereby enhancing the LLMs' semantic parsing abilities. We set our work apart by proposing a more organized method for the prompt engineering task that does not rely on few-shot examples. However, integrating contextual few-shot prompting into our framework is feasible at a low cost; users simply need to create a small number of examples based on the complexity of the problem to guide the LLM. We will explore this possibility in future work.

Finally, despite their individual successes, the integration of LLMs and ASP remains relatively unexplored. This synergy presents a unique opportunity to leverage the strengths of both paradigms: LLMs for their linguistic and contextual capabilities, and ASP for its precise, logic-based reasoning. By combining these technologies, we can create powerful AI systems capable of both understanding complex natural language inputs and performing sophisticated logical inference [35, 36].

²Although, the authors still have access to labeled QA datasets in which tuple $\langle S, q, a \rangle$ is given.

6. Conclusion

In this paper, we have introduced an approach for combining Large Language Models (LLMs) and Answer Set Programming (ASP) to harness their complementary strengths in natural language understanding and logical reasoning. Our prototype system (<https://github.com/lewashby/llmasp>) is written in Python, and it is powered by the LLM llama3:70b from ollama³ and the CLINGO Python API [37]. By providing predefined prompts and enriching specifications with domain-specific knowledge, our approach enables users to tailor the system to diverse problem domains and applications, enhancing its adaptability and versatility. With respect to the previous version of the system [15], the predefined prompts can be easily modified as they are stored in a separate YAML file. Such a separation is a first step toward our future research directions, which encompass the evaluation of the quality of the answers provided by our integrated LLMASP system, and the exploration with different prompts to improve the overall quality of the system.

Acknowledgments

This work was supported by Italian Ministry of University and Research (MUR) under PRIN project PRODE “Probabilistic declarative process mining”, CUP H53D23003420006, under PNRR project FAIR “Future AI Research”, CUP H23C22000860006, under PNRR project Tech4You “Technologies for climate change adaptation and quality of life improvement”, CUP H23C22000370006, and under PNRR project SERICS “SEcurity and RIghts in the CyberSpace”, CUP H73C22000880001; by Italian Ministry of Health (MSAL) under POS projects CAL.HUB.RIA (CUP H53C22000800006) and RADIOAMICA (CUP H53C22000650006); by Italian Ministry of Enterprises and Made in Italy under project STROKE 5.0 (CUP B29J23000430005); and by the LAIA lab (part of the SILA labs). Alviano is member of Gruppo Nazionale Calcolo Scientifico-Istituto Nazionale di Alta Matematica (GNCS-INdAM).

References

- [1] T. B. Brown, et al., Language models are few-shot learners, CoRR abs/2005.14165 (2020). URL: <https://arxiv.org/abs/2005.14165>. arXiv:2005.14165.
- [2] A. Chowdhery, et al., Palm: Scaling language modeling with pathways, J. Mach. Learn. Res. 24 (2023) 240:1–240:113. URL: <http://jmlr.org/papers/v24/22-1144.html>.
- [3] H. Touvron, et al., Llama: Open and efficient foundation language models, CoRR abs/2302.13971 (2023). doi:10.48550/ARXIV.2302.13971. arXiv:2302.13971.
- [4] V. Marek, M. Truszczyński, Stable models and an alternative logic programming paradigm, in: The Logic Programming Paradigm: a 25-year Perspective, 1999, pp. 375–398. doi:10.1007/978-3-642-60085-2_17.
- [5] I. Niemelä, Logic programming with stable model semantics as a constraint programming paradigm, Annals of Mathematics and Artificial Intelligence 25 (1999) 241–273. doi:10.1023/A:1018930122475.
- [6] M. Gelfond, V. Lifschitz, Logic programs with classical negation, in: D. Warren, P. Szeredi (Eds.), Logic Programming: Proc. of the Seventh International Conference, 1990, pp. 579–597.
- [7] H. Jin, Y. Zhang, D. Meng, J. Wang, J. Tan, A comprehensive survey on process-oriented automatic text summarization with exploration of llm-based methods, CoRR abs/2403.02901 (2024). doi:10.48550/ARXIV.2403.02901. arXiv:2403.02901.
- [8] W. Zhang, X. Li, Y. Deng, L. Bing, W. Lam, A survey on aspect-based sentiment analysis: Tasks, methods, and challenges, IEEE Trans. Knowl. Data Eng. 35 (2023) 11019–11038. doi:10.1109/TKDE.2022.3230975.
- [9] P. Cappanera, M. Gavanelli, M. Nonato, M. Roma, Logic-based benders decomposition in answer set programming for chronic outpatients scheduling, Theory Pract. Log. Program. 23 (2023) 848–864. doi:10.1017/S147106842300025X.

³<https://ollama.com/>

- [10] M. Cardellini, P. D. Nardi, C. Dodaro, G. Galatà, A. Giardini, M. Maratea, I. Porro, Solving rehabilitation scheduling problems via a two-phase ASP approach, *Theory Pract. Log. Program.* 24 (2024) 344–367. doi:10.1017/S1471068423000030.
- [11] F. Wotawa, On the use of answer set programming for model-based diagnosis, in: H. Fujita, P. Fournier-Viger, M. Ali, J. Sasaki (Eds.), *Trends in Artificial Intelligence Theory and Applications. Artificial Intelligence Practices - 33rd International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems, IEA/AIE 2020, Kitakyushu, Japan, September 22-25, 2020*, Proceedings, volume 12144 of *Lecture Notes in Computer Science*, Springer, 2020, pp. 518–529. doi:10.1007/978-3-030-55789-8_45.
- [12] R. Taupe, G. Friedrich, K. Schekotihin, A. Weinzierl, Solving configuration problems with ASP and declarative domain specific heuristics, in: M. Aldanondo, A. A. Falkner, A. Felfernig, M. Stettinger (Eds.), *Proceedings of the 23rd International Configuration Workshop (CWS/ConfWS 2021)*, Vienna, Austria, 16-17 September, 2021, volume 2945 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2021, pp. 13–20. URL: https://ceur-ws.org/Vol-2945/21-RT-ConfWS21_paper_4.pdf.
- [13] K. Basu, S. C. Varanasi, F. Shakerin, J. Arias, G. Gupta, Knowledge-driven natural language understanding of english text and its applications, in: *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021*, AAAI Press, 2021, pp. 12554–12563. doi:10.1609/AAAI.V35I14.17488.
- [14] Y. Zeng, A. Rajasekharan, P. Padalkar, K. Basu, J. Arias, G. Gupta, Automated interactive domain-specific conversational agents that understand human dialogs, in: M. Gebser, I. Sergey (Eds.), *Practical Aspects of Declarative Languages - 26th International Symposium, PADL 2024, London, UK, January 15-16, 2024*, Proceedings, volume 14512 of *Lecture Notes in Computer Science*, Springer, 2024, pp. 204–222. doi:10.1007/978-3-031-52038-9_13.
- [15] M. Alviano, L. Grillo, Answer set programming and large language models interaction with yaml: Preliminary report, in: *CILC, CEUR Workshop Proceedings*, CEUR-WS.org, 2024.
- [16] M. Gebser, R. Kaminski, T. Schaub, Complex optimization in answer set programming, *Theory Pract. Log. Program.* 11 (2011) 821–839.
- [17] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, D. Amodei, Language models are few-shot learners, in: H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, H. Lin (Eds.), *Advances in Neural Information Processing Systems*, volume 33, Curran Associates, Inc., 2020, pp. 1877–1901. URL: https://proceedings.neurips.cc/paper_files/paper/2020/file/1457c0d6bfc4967418bfb8ac142f64a-Paper.pdf.
- [18] R. Dabre, C. Chu, A. Kunchukuttan, A survey of multilingual neural machine translation, *ACM Computing Surveys (CSUR)* 53 (2020) 1–38.
- [19] F. Petroni, T. Rocktäschel, P. Lewis, A. Bakhtin, Y. Wu, A. H. Miller, S. Riedel, Language models as knowledge bases?, *arXiv preprint arXiv:1909.01066* (2019).
- [20] J. Hoffmann, S. Borgeaud, A. Mensch, E. Buchatskaya, T. Cai, E. Rutherford, D. d. L. Casas, L. A. Hendricks, J. Welbl, A. Clark, et al., Training compute-optimal large language models, *arXiv preprint arXiv:2203.15556* (2022).
- [21] H. Zhang, J. Xu, J. Wang, Pretraining-based natural language generation for text summarization, *arXiv preprint arXiv:1902.09243* (2019).
- [22] Z. Liu, M. Patwary, R. Prenger, S. Prabhunoye, W. Ping, M. Shoenybi, B. Catanzaro, Multi-stage prompting for knowledgeable dialogue generation, in: *Findings of the Association for Computational Linguistics: ACL 2022*, 2022, pp. 1317–1337.
- [23] P.-L. H. Cabot, R. Navigli, Rebel: Relation extraction by end-to-end language generation, in: *Findings of the Association for Computational Linguistics: EMNLP 2021*, 2021, pp. 2370–2381.
- [24] M. Lewis, Y. Liu, N. Goyal, M. Ghazvininejad, A. Mohamed, O. Levy, V. Stoyanov, L. Zettlemoyer,

- Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension, in: Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, 2020, pp. 7871–7880.
- [25] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, P. J. Liu, Exploring the limits of transfer learning with a unified text-to-text transformer, *Journal of machine learning research* 21 (2020) 1–67.
- [26] N. De Cao, G. Izacard, S. Riedel, F. Petroni, Autoregressive entity retrieval, in: ICLR 2021-9th International Conference on Learning Representations, volume 2021, ICLR, 2020.
- [27] R. Blloshmi, S. Conia, R. Tripodi, R. Navigli, et al., Generating senses and roles: An end-to-end model for dependency-and span-based semantic role labeling, in: Proc. of 30th International Joint Conference on Artificial Intelligence, IJCAI 2021, 2021, pp. 3786–3793.
- [28] J. Wei, X. Wang, D. Schuurmans, M. Bosma, F. Xia, E. Chi, Q. V. Le, D. Zhou, et al., Chain-of-thought prompting elicits reasoning in large language models, *Advances in neural information processing systems* 35 (2022) 24824–24837.
- [29] M. Nye, M. Tessler, J. Tenenbaum, B. M. Lake, Improving coherence and consistency in neural sequence models with dual-system, neuro-symbolic reasoning, *Advances in Neural Information Processing Systems* 34 (2021) 25192–25204.
- [30] M. Borroto, I. Kareem, F. Ricca, Towards automatic composition of asp programs from natural language specifications, *arXiv preprint arXiv:2403.04541* (2024).
- [31] T. Kuhn, A survey and classification of controlled natural languages, *Computational linguistics* 40 (2014) 121–170.
- [32] R. Schwitter, Controlled natural languages for knowledge representation, in: *Coling 2010: Posters*, 2010, pp. 1113–1121.
- [33] A. Ishay, Z. Yang, J. Lee, Leveraging large language models to generate answer set programs, in: *Proceedings of the 20th International Conference on Principles of Knowledge Representation and Reasoning*, 2023, pp. 374–383.
- [34] Z. Yang, A. Ishay, J. Lee, Coupling large language models with logic programming for robust and general reasoning from text, in: *Findings of the Association for Computational Linguistics: ACL 2023*, 2023, pp. 5186–5219.
- [35] A. Ishay, Z. Yang, J. Lee, Leveraging Large Language Models to Generate Answer Set Programs, in: *Proceedings of the 20th International Conference on Principles of Knowledge Representation and Reasoning*, 2023, pp. 374–383. URL: <https://doi.org/10.24963/kr.2023/37>. doi:10.24963/kr.2023/37.
- [36] A. Rajasekharan, Y. Zeng, G. Gupta, Argument analysis using answer set programming and semantics-guided large language models., in: *ICLP Workshops*, 2023.
- [37] R. Kaminski, J. Romero, T. Schaub, P. Wanko, How to build your own asp-based system?!, *Theory and Practice of Logic Programming* 23 (2023) 299–361. doi:10.1017/S1471068421000508.