Assignment1 – SymNLProj Warmup-Up: *Natural Language Semantics* with *Prolog*

The goal of this assignment will be to *implement* a basic *natural language*-based *reasoning system* in *Prolog*.

Problem files, example solutions, and a code skeleton are provided in the assignment repository[1].

Important

1. This assignment has to be solved individually. If you use someone else's *code* (even if you modify it), you will fail the assignment.
2. This assignment is pre-requisite for the project. You can only participate in the project if you pass this assignment. The *idea* is that this assignment lets you find out if the project works for you.
3. If you are stuck, ask for help.
4. Deadline: Nov. 29, 2024.
5. Early deadline: Nov. 15, 2024. At this early deadline, you have to submit a solution worth at least 10 points. This is a requirement to continue the project.
6. This is a new assignment. If you find any mistakes/inconsistencies/poor descriptions (very likely), please report them.

Background: The Wumpus World You may know the *Wumpus World* from the AI *lectures*. In this assignment, we will do some *reasoning* about the *Wumpus World* based on *scout reports*.

The world is a 4×2 grid, where each cell is labelled with a *lowercase letter*. Here is the map:

a	b	c	d
e	f	g	h

Scouts can explore the world and report back what they see. In particular, they look for *pits* and *echoes*. A *cell* will have an **echo** if and only if there is a *pit* in it or one of the adjacent *cells*.

[1] `https://gitlab.rrze.fau.de/wrv/SymNLProj/ws2425/a1-warmup/assignment`

Overview In this assignment, you will have to implement a Prolog program that parses English scout reports (observations about the world) and ultimately answers questions based on them. The assignment repository contains a lot of example problems and solutions, and some "driver code" that iterates over the problems and calls your predicates.

Task 1.1 (Parse scout reports into PL^{nq})

Scout reports are writting in English and report on the distribution of *pits* and *echoes* on the grid. Here are some examples:

1. *There is no* echo *in f.*
2. *If there is a pit in cell h, then there is no pit in b.*
3. *We discovered echoes in cells d, a, b and f.*

We intentionally do not specify the *scout report* fragment in detail – you can infer it from the problem files.

Write a *Prolog program* that *parses* the English *sentences* into a PL^{nq} *formula*. For example, the *sentence*

If there is a pit *in* g *or there is no* pit *in a, then there is a* pit *in f.*

should be *translated* to the *formula*

```
implies(or(pit(g),not(pit(a))),pit(f))
```

You can use the following *predicate* to *translate* a *sentence* into a *list* of *tokens*.

```
string_to_words(String, Words) :-
   string_lower(String, LowerString),
   split_string(LowerString, " ", ",.?", WordStrs),
   maplist(atom_string, Words, WordStrs).
```

The appendix contains more information on how to use *Prolog* for *parsing*.

Notes

1. It is okay if your grammar overgenerates, i.e. if it accepts sentences that are ungrammatical, like *There is a pits in a*.
2. The driver code expects the *predicate* to be called nl_to_plnq (see the driver code for details).
3. For simplicity, the scout reports are designed to be (semantically) non-ambiguous. The same will be true for the questions in the later tasks. If you do find ambiguities, please report them.

Task 1.2 (Evaluate truth of *scout reports*)

Write a *predicate* report_eval that, given a description of the world and a *scout report*, determines whether the *scout report* is *true* or *false*. The description of the world is a *list* of *variable assignments* asgn(X, Y), where X is an *atomic proposition* and Y is a *truth value*.
Example:

```
?— report_eval([asgn(pit(a), true), asgn(pit(b), false)], "There is a pit in a.", X).
X = true.
?— report_eval([asgn(pit(a), true), asgn(pit(b), false)], "There is a pit in b.", X).
X = false.
```

Hint:

1. Use your solution from the previous task for parsing the scout reports.
2. As inspiration: $X \wedge Y$ should evaluate to true if both X and Y evaluate to true. $X \wedge Y$ should evaluate to false if either X or Y evaluates to false.

Task 1.3 (*Scout Report* Reasoning)

Ultimately, the goal is to answer questions based on knowledge from *scout reports*. Extend your *grammar* to include yes/no questions like

1. *Is there a* pit *in a?*
2. *Is it true that there are no echoes in cells d and h?*

Write a *predicate* that, given *scout reports* and a question, provides an answer.
Example:

```
?— answer(
   ["If there is a pit in a, then there is a pit in b.",
    "There is no pit in b".],
   "Is there a pit in a?", X).
X = "no".
```

You can again use the problem files and example solutions as a guide.

Hint: Basically, you have to *implement* a prover here. You should already have an *evaluator* from the previous task. If you feed an "uninstantiated" *variable assignment* (i.e. [asgn(pit(a), B1), asgn(echo(a), B2), asgn(pit(b), B3), ...]) to the *evaluator*, it will try to find a *variable assignment* that *satisfies* the $\mathrm{PL}^{\mathrm{nq}}$ *formula*. In other words, you have a *satisfiability* checker for free. You can turn that into a prover by exploiting the fact that φ is a *theorem* if and only if $\neg\varphi$ is *unsatisfiable*.

Task 1.4 (World Knowledge)

Include the *world knowledge* that an *echo* is present if and only if there is a *pit* in the cell or in one of the neighboring *cells*.

You can get the *world knowledge* as a *list* of facts from the assignment *repository*. It contains sentences like

If there is a pit in a, then there are echoes in a, e and b.

You should already be able to parse these sentences.

Develop a *predicate* answer2 that anwers questions like in the previous task, but now also takes the *world knowledge* into account:

```
?- answer2(["There is a pit in a."], "Is there an \sn{echo} in b?", X).
X = "yes".
```

Hint: You might notice that the problems are sufficiently complex that a naive solver is slow. A proper solution would be to use a SAT solver, but that is beyond the scope of this assignment. It is possible to solve the problems with a naive solver, but you might have to make sure that you do not backtrack too much. For example, if there are multiple readings, your answer predicate could just take the first one (use the ! operator).

Submission and Points

At the deadline, you have to submit:

1. All your code,
2. solution files for the problems,
3. a README file that explains how to run your code,
4. a short summary of how you solved the problem ($\frac{1}{2} - 1$ page, but the upper limit is optional). The summary should focus on the conceptual aspects of your solution, and not document the code.

For each task, you get 100 problem files. They are split into 4 groups of 25 files each (0–24, 25–49, etc.) of increasing difficulty. For the first two tasks, you get 6 points for each group of 25 files you solve. And for the other two tasks, you get 4 points per group. Note that you only get points for a group if all problems were solved correctly. That gives you a total of 80 points for the solutions.

If you get at least 1 point for the solutions, you can get additional 20 points are for the README and the summary.

If the grading scheme does not work well, we might adjust it later on (likely in your favor).

Parsing with Prolog

Parsing with lists

Consider the sentence

There is a pit in a and there is a pit in b.

which we will represent as the token sequence

```
[there, is, a, pit, in, a, and, there, is, a, pit, in, b]
```

We can parse this with a ternary predicate sentence/3, where the first argument is the resulting PLNQ expression, the second argument is the list of words to parse, and the third argument is words that remain to be parsed.

You can define the predicate as follows:

```
sentence(pit(X), [there, is, a, pit, in, X | T], T).
sentence(and(A, B), S0, T) :-
   sentence(A, S0, [and | S1]),
   sentence(B, S1, T).
```

You can use it by calling

```
?- sentence(Expr, [there,is,a,pit,in,a,and,there,is,a,pit,in,b], []).
```

Note that we directly obtain a PNLQ formula as a parse tree.

This trick of using lists is a common way to parse in Prolog. In fact, there is essentially a short-hand for this in Prolog, called *definite clause grammar* (*DCG*). There are many tutorials on the web that explain how to use *DCGs* in *Prolog*. For example, the Wikipedia page on *DCGs* is a good starting point.

Here is the same parser as a *DCG*:

```
sentence(pit(X)) --> [there, is, a, pit, in, X].
sentence(and(A, B)) --> sentence(A), [and], sentence(B).
```