Assignment1 – SymNLProj: Model generation

Task 1.1 (Implement model generation)
Implement tableau-based model generation in ELPI and connect it to the semantics construction from the previous assignment.

If you want to restrict yourself to propositional tableaux, you can get partial points if you connect it to fragment 1. For full points, your implementation should support the NP fragment from the previous assignment. In this case, you can skip fragment 1 (and of course the PP fragment).

Supporting the description operator is tricky. Therefore, it is optional and you can get bonus points for it. In this case, you might still want to restrict yourself to the logical expressions that can actually occur in the fragment (no functions, equality is only introduced by the description operator, etc.). If you support description, you should also include the unique name assumption.

Tips

1. As always, feel free to reach out if you have any questions/problems (whether it's about ELPI or how to approach some part of the assignment).
2. Simplify the logical expressions as a pre-processing step, so that you are left with a small set of connectives (e.g. only negation, conjunction and universal quantification).
3. For propositional tableau, you can make a predicate with two arguments. The first argument is the "todo list". It lists all formulae that still have to be processed. For example, if you process $A \wedge B^T$, then you should put A^T and B^T on the todo list. The second argument is the output. It contains the model as a list of atomic formulae marked as true or false. ELPI's backtracking allows you to explore more models.
4. For branching, have one rule for each branch. ELPI's DFS will then first explore the first branch. If it fails, it will backtrack and try the second branch.
5. For first-order tableau, you will need more arguments (e.g. the Herbrand base, the set of universally quantified formulae, etc.).
6. Make sure you are aware of prolog's/ELPI's cut operator (!).

ELPI tips

Using pi Because of higher-order abstract syntax, the body of a quantification is a function, not a logical expression. This can lead to some challenges, which can be solved by using the **pi** operator. Essentially, **pi** is ELPI's way of saying "for all". Here is an example if you want to simplify a formula:

```
type simplify oo −> oo −> prop.
% ...
simplify (forall A) (forall A2) :− pi x \ simplify (A x) (A2 x).
```

Tracing the model generation A simple trick to debug the model generation is to execute a print statement for every call:

```
type model_generation ... −> ... −> prop.
model_generation A B :− print "model_generation" A B, fail.
% ... (the actual implementation)
```

Generating witnesses One way to generate witnesses is to use the following constructor:

```
type witness int −> ii.
```

Submission and Points

At the deadline, you have to submit:

1. All your code,
2. a README file that
   (a) explains briefly your implementation (what files are relevant for what, etc.),
   (b) shows a few example commands for testing your implementation.

You can get up to 100 points for this assignment – 50 if you support fragment 1 and 100 if you support the NP fragment. If you also support description, you can get up to 30 bonus points. If the grading scheme does not work well, we might adjust it later on.