# Homework assignment 2 – Symbolic Systems I – UvA, June 2020

*Ard Snijders*

## Question 1: monotonic reasoning using SAT

We can test whether a set $\Phi$ logically entails a formula $\phi$ by testing for satisfiability of its contradiction, i.e. $\Phi \wedge \neg\phi$. Such an approach could be described as determining entailment by proof by refutation, or proof by contradiction (Norvig, 2016). We can do so by first converting the problem to Conjunctive Normal Form (CNF). We could then take different approaches for determining satisfiability, such as applying resolution. In our case however, we already have an algorithm which can determine satisfiability of propositional CNF formulas. Therefore, we only need to concern ourselves with the conversion of our set of propositional logic sentences to CNF, i.e. we need to express sentences as conjunctions of clauses.

In principle, a sentence of propositional logic can be expressed as a conjunction of clauses such that they are logically equivalent. A general procedure for converting a sentence of propositional logic to CNF thus involves the application of standard logical equivalences; by applying rules such as elimination, De Morgan's law and the distributive law, we can re-write an arbitrary sentence of propositional logic until it is comprised of nothing but conjunctions of disjunctions of literals, i.e. such that the original sentence is in CNF. However, this procedure would be a naive approach, as the size of the resulting equation can increase exponentially, which is not efficient. Rather, we can choose to abandon the concept of logical equivalence and instead generate a CNF expression which is only equisatisfiable, which is sufficient for determining if there is an assignment that satisfies the original propositional formulas. One method we can use to achieve this is the Tseytin transformation, whose output formula scales linearly with the input.

We can then employ the already given satisfiability algorithm to determine whether this expression is satisfiable, and the outcome of this procedure can then be used to determine whether $\Phi$ logically entails a formula $\phi$ - that is, if the contradiction $\Phi \wedge\neg\phi$ is satisfiable, it follows that $\Phi$ does not entail $\phi$, but if this contradiction is not satisfiable, it follows that $\Phi$ entails $\phi$.

## Question 2: monotonic reasoning using ASP

Similarly to the solution to assignment 1, we can test whether a set $\Phi$ logically entails a formula $\phi$ by testing for satisfiability of its contradiction, i.e. $\Phi \wedge\neg\phi$. We can do so by first converting the set of propositional formulas into CNF. The conversion of propositional logic formulas to CNF can be accomplished via the method outlined in Question 1. We can then encode a CNF formula into ASP by generating a cardinality rule for each clause in the CNF formula, where for each rule, 1 item can be chosen amongst all the items (where each item represents a literal from the clause). The answer set program will only return answer sets if all clauses hold.

For instance, we can express the following formula (in CNF) into ASP with the following code:

$$\varphi = \quad (x_1 \vee x_2 \vee \neg x_3) \wedge (x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_2 \vee \neg x_3) \wedge (x_3 \vee x_4) \wedge (x_3 \vee \neg x_4)$$

```
    1 { item(x1); item(x2); not item(x3)}.
    1 { item(x1); not item(x2); not item(x3)}.
    1 { not item(x1); item(x2); not item(x3)}.
    1 { item(x3); item(x4)}.
    1 { item(x3); not item(x4)}.

Answer set: {item(x1), item(x2), item(x3)}
Answer set: {item(x1), item(x2), item(x3), item(x4)}
```

Thus, by encoding each clause of the CNF of $\Phi \wedge\neg\phi$, we can find whether or not $\Phi$ logically entails $\phi$; if there is no answer set for this program, it must mean that $\Phi$ logically entails $\phi$.

## Question 3: encoding default logic in ASP

We can solve this task by encoding the PDFN default theory into ASP such that we can guess possible answer sets E which follow from the PDFN default theory. We can first enter each of the propositional literals in W as rules without bodies, or facts. Then, we can generate rules which set the logical consequent of each default d in D to true if (1) The prerequisite of the default is satisfied by our current knowledge, and (2) Our current knowledge does not contradict the justification of the default. Since we are only werking with PDFN defaults, where the logical consequent is identical to the justification, it suffices to simple generate if-then rules where a logical consequent is accepted if the prerequisite holds. For instance, the default theory pair as provided in the question can be expressed in ASP, via the rules explained above, as follows (comments denoted by use of #):

```
# P
    p. # This literal belongs to W, thus we can encode it as a fact
    q.
    r :- q. # default 1
    -s :- p, r. # default 2
    q :- p, r. # default 2

Answer set: {-s, p, q, r} # The resulting extension
```

The answer sets generated by this ASP program will then correspond to the extensions that would have been derived from the original PDFN default theory. Note that in ASP, we cannot have rules where the head is comprised of a conjunction, e.g. $a\&b$. In this case, we need to generate separate rules for each literal in the logical consequent of an accepted default, if its consequent consists of multiple literals.

We can check whether the literal $l$ is among the possible extensions by defining an additional constraint such that the literal must be true for all answer sets;

```
:- not l.
```

That way, if the literal $l$ is in one of the answer sets, the program will produce that answer set, and otherwise there will be no answer set (from which we can derive that there is no answer set containing the literal $l$).

## Question 4 : expressing cardinality rules in basic ASP

I am assuming that for this question, the task is to write an explicit ASP program which had the same outcomes for u, m and n as it would using the cardinality rule, as opposed to writing a Python function which dynamically generates the ASP program.

I was able to generate all possible permutations for a given constant $u$ using the code below, but I haven't got the faintest clue on how I can then restrict the set of answer sets such that the implicit constraints (as specified by $m$ and $n$ in the provided cardinality rule example) are satisfied, without employing choice rules such as count or aggregates.

```
    #const n=2.
    #const m=3.
    #const u=3.
    value(1..u).
    item(X) :- not not_item(X), value(X).
    not_item(X) :- not item(X), value(X).
    #show item/1.

Answer set: {item(3)}
Answer set: {}
Answer set: {item(1), item(2), item(3)}
Answer set: {item(1), item(2)}
Answer set: {item(2), item(3)}
Answer set: {item(2)}
Answer set: {item(1), item(3)}
Answer set: {item(1)}
```

## Question 5: disjunction is a true extension of ASP

To tackle this problem, we could employ Saturation.