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

Lennert Jansen

Question 1

We are given a set of formulas Φ and the formula ϕ, that may be propositional logic formulas of arbitrarily complex structure. The goal of this task is to describe an algorithm for deciding the satisfiability of $\Phi \models \phi$, that is, whether ϕ is logically entailed by Φ, or even more simply, decide if $\Phi \models \phi$ is true. Furthermore, it is assumed we have an efficient algorithm for deciding satisfiability (SAT) of propositional CNF formulas. Based on the fact that the efficient algorithm for deciding satisfiability only works for propositional formulas in conjunctive normal form, the current task is to reduce $\Phi \models \phi$ to CNF.

Page 250 of Russell and Norvig provides the useful result $\alpha \models \beta$ *if and only if the sentence* $(\alpha \wedge \neg\beta)$ *is unsatisfiable.*

Based on the first given remark, I assume that the formulas in Φ and ϕ may contain bidirectionals ($\Leftrightarrow$), implications ($\Rightarrow$), and other symbols and structures of propositional logic that aren't permitted in CNF. We must therefore apply the series of steps for converting the sentence $(\Phi \wedge \neg\phi)$ to CNF, outlined in pages 253-254 of Russell and Norvig. Note that a result in the same range of pages states that *every sentence of propositional logic is logically equivalent to a conjunction of clauses*, meaning that we are guaranteed to arrive at $(\Phi \wedge \neg\phi)$'s CNF translations, $(\Phi' \wedge \neg\phi')$.

The following steps are applied to the formulas in Φ and ϕ:

1. Eliminate bidirectionals ($\Leftrightarrow$) by replacing $\alpha \Leftrightarrow \beta$ with $(\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)$.
2. Eliminate implications ($\Rightarrow$) by replacing $\alpha \Rightarrow \beta$ with $\neg\alpha \vee \beta$.
3. In CNF, negations ($\neg$) are only allowed to appear in literals, so we move $\neg$ inwards by repeated application of the following equivalences (whenever it applies):
   - $\neg(\neg\alpha) \equiv \alpha$
   - $\neg(\alpha \wedge \beta) \equiv (\neg\alpha \vee \neg\beta)$
   - $\neg(\alpha \vee \beta) \equiv (\neg\alpha \wedge \neg\beta)$
4. Distribute $\vee$ over $\wedge$ whenever possible by applying the distributivity law on page 249 of Russell and Norvig:
   - $(\alpha \vee (\beta \wedge \gamma)) \equiv ((\alpha \vee \beta) \wedge (\alpha \vee \gamma))$

The previously mentioned steps are then repeatedly applied until a CNF, $(\Phi' \wedge \neg\phi')$, of the original problem is obtained. The efficient algorithm for deciding the satisfiability of this CNF will then return `True` if $\Phi \models \phi$, and `False` if ϕ is *not* logically entailed by Φ.

Question 2

The goal of this exercise is analogous to that of Question 1. The task is again to decide if $\Phi \models \phi$ is true. However, in this question we are given an efficient algorithm for deciding if a given answer set program has *at least one* answer set (which is equivalent to deciding satisfiability if the program is logically equivalent to a CNF). The goal is therefore now to reduce the task of propositional reasoning to answer set programming.

This task is specifically concerned with showing entailment. From Definition 7.2.4 on page 291 of Van Harmelen, Lifschitz, and Porter, we can establish that *A program* Π *entails a ground literal* l $(\Pi \models l)$
if l *is satisfied by* ***every*** *answer set of* Π.

This definition is not yet applicable to our given algorithm, as the algorithm can only efficiently determine if a given answer set program has **at least one** answer set. Some conversions and modifications are therefore needed to employ the previously mentioned definition for our task. We must...

- Rephrase Definition 7.2.4 of Van Harmelen, Lifschitz, and Porter such that it is compatible with our efficient algorithm for deciding if an ASP has at least one answer set.

- ...encode the formulas that constitute Φ into an answer set program Π.
- ...define a ground literal l such that it holds true *if and only if* ($\Leftrightarrow$) ϕ is true.

Definition 7.2.4 has the following implications

$$\begin{aligned} & (\Pi \models l) \textit{ if every answer set of } \Pi \textit{ satisfies } l \\ \Leftrightarrow \quad & (\Pi \models l) \textit{ if every answer set of } \Pi \textit{ does not satisfy } \neg l \\ \Leftrightarrow \quad & (\Pi \models l) \textit{ if } \textbf{\textit{no}} \textit{ answer set of } \Pi \textit{ satisfies } \neg l \end{aligned}$$

So if we define Π' to be the following program:

```
Π.
¬l.
```

and apply our given algorithm to Π', it will return `True` if Π' has at least one answer set (i.e., l is **not** logically entailed by Π), and `False` if Π' has no answer sets, meaning that $\Pi \models l$.

This outlined procedure now assumes that the given algorithm has the capability to extend the ASP language with classical negation and disjunction. If it *is* necessary to, Gebser et al. provide the steps for achieving this extension in paragraphs 2.3.4-5. More specifically, we would need to convert every formula in Φ and ϕ into CNF (which is guaranteed by the result of Russell and Norvig in the previous question). Then to convert from CNF to ASP and write Π, we apply the following rules: add to Π a cardinality constraint of the form `1 {`$\mathtt{a}_1$`, ..., `$\mathtt{a}_n$`, ¬ `$\mathtt{a}_{n+1}$`, ..., ¬ `$\mathtt{a}_m$`}.` for every disjunction of literals $a_1, ..., a_n, \neg a_{n+1}, ..., a_m, (m \geq n)$.

Question 3

The goal is to describe an algorithm that can decide if an arbitrary PDFN default theory (W, D) has an extension that includes some given propositional literal l. Chen et al. show that it is possible to convert an arbitrary default theory (W, D) into an ASP P, such that there is one-to-one correspondence between the extensions of (W, D) and the answer sets of P. Therefore, the main goal of this exercise is equivalent to establishing whether default theory (W, D)'s ASP counterpart P has an answer set that contains l. Which in turn is equivalent to deciding whether logic program $P' = P, l.$ has an answer set. This last equivalence is solvable by our given efficient algorithm. Thus, I describe a procedure for converting PDFN default theory (W, D) into ASP P', based on the results of Chen et al.:

1. Convert and add each formula in the set of propositional formulas $W = \{w_1, ..., w_{|W|}\}$ to P':

   ```
   w_i.
   ```

   $\forall 1 \leq i \leq |W|$.

2. Convert and add every default in D (which are of the form $a : b_1, ..., b_n / c$)):

   ```
   c :- a, not ¬b_1, ..., not ¬b_n.
   ```

3. As we are limited to PDFN default theories, we will not be dealing with disjunctions in (W, D). Therefore, every formula can be separated in terms of its literals, converted and added to P'.

4. Finally, we add the propositional literal l:

   ```
   l.
   ```

Now, when applying the given algorithm to logic program P', it will return `True` is an answer set is found (meaning that (W, D) has an extension that includes l), and `False` when no answer set is found, i.e., there is no extension of (W, D) that includes the literal l.

Question 4

Using only the basic language of answer set programming and range notation, e.g., `item(1..u)`, the following answer set program `P` yields the exact same answer sets as the given ASP consisting of the line `n { item(1..u) } m.`, for each combination of declared values `n`, `m`, and `u`. It is loosely based on the explanation by Gebser et al. (page 19) about substituting cardinality rules for basic ASP syntax.

`P` works by first establishing all possible values `item()` will be able to assume, using `number(1..u).`, using `count/2` it decides if an answer set satisfies having `K` or more items with index value `X` or less. It then decides which items are

allowed in the set. Finally, it enforces the correct range by not allowing the range to exceed `m`, while being at least `n`.

```
#const n=2.
#const m=3.
#const u=4.
number(1..u).

count(X, K) :- count(X-1, K), number(X).
count(X+1, K+1) :- count(X+1, K), count(X, K), item(X+1), number(X+1).
count(1,1) :- count(1,0), number(1), item(1).
count(1,0).

item(X) :- not unchoose(X), number(X).
unchoose(X) :- not item(X), number(X).

n_or_more :- count(u, n).
m1_or_more :- count(u, m+1).
exceed_range :- n_or_more, not m1_or_more.
:- not exceed_range.

#show item/1.
```

Question 5

The goal of this exercise is to encode a problem in an ASP `P` such that it takes two propositional logic formulas in CNF ϕ_1 and ϕ_2 as input, and decides if both ϕ_1 is satisfiable and ϕ_2 is unsatisfiable. Such a program `P` would consist of one separate part for each CNF (of course, converted to ASP, as done in previous questions). The saturation technique would then be applied to both parts.

References

[Che+10] Yin Chen et al. "dl2asp: implementing default logic via answer set programming". In: *European Workshop on Logics in Artificial Intelligence*. Springer. 2010, pp. 104–116.

[Geb+12] Martin Gebser et al. "Answer set solving in practice". In: *Synthesis lectures on artificial intelligence and machine learning* 6.3 (2012), pp. 1–238.

[RN02] Stuart Russell and Peter Norvig. "Artificial intelligence: a modern approach". In: (2002).

[VLP08] Frank Van Harmelen, Vladimir Lifschitz, and Bruce Porter. *Handbook of knowledge representation*. Elsevier, 2008.