Assignment 2 for COSC410 Logic for AI

Due 16 May 2016 for 11 marks = 11%

1. Let L_A be arbitrary and let $\alpha, \beta, \varphi \in L_A$. Suppose that $\preccurlyeq$ is a total preorder on S. Let $\mid\!\sim$ be the rational consequence relation induced by $\preccurlyeq$.

   Show that $\mid\!\sim$ has the property of conditional insertion, i.e. that if $\alpha \wedge \beta \mid\!\sim \varphi$ then $\alpha \mid\!\sim \beta \rightarrow \varphi$.
   (1 mark)

2. Consider the 3-Card System with $A = \{r_1, r_2, r_3, g_1, \ldots, b_3\}$ and $S = \{rgb, rbg, grb, gbr, brg, bgr\}$. Assume that the total preorder faithful to belief set K is given by the following:

rbg	brg
rgb	grb
gbr	bgr

   Let K be the set of beliefs to which the ordering is faithful. Give, for each of the following, its set of models and then express the belief set in the form $Cn(\{\alpha\})$:

   K

   $K * g_2$

   $K - (\neg g_1)$

   (2 marks)

3. Show that agents have negative introspection in epistemic logic, i.e. that if s satisfies $\neg[i]\varphi$ then s satisfies $[i]\neg[i]\varphi$. (2 marks)

4. Using a SAT solver (6 marks)

   The aim of this question is to give you the experience of using a well known SAT solver to solve a problem. Although describing the problem takes some space, it is not intended to be terribly difficult. But you'll learn a lot.

   The solver you will use is zChaff[1]. There are some nice slides about it[2] by Lilia Yerosheva.

`~ok/COSC410/zchaff.macos`	zchaff compiled for MacOS X
`~ok/COSC410/zchaff.linux`	zchaff compiled for Linux
`~ok/COSC410/zchaff.d`	zchaff sources

   (The Linux version isn't built yet.)

   zchaff is a command line program that takes a file name and an optional time limit as arguments. The input file describes a problem in Conjunctive Normal Form, more precisely, in the DIMACS notation for CNF.

   Conjunctive Normal Form will be explained in Richard O'Keefe's first lecture, but here's a quick summary:

   - an *atom* is a propositional variable;
   - a *literal* is either an atom or the logical negation of an atom;
   - a *clause* is a set of literals, considered as combined by disjunction ($\vee$); and
   - a *CNF formula* is a set of clauses, considered as combined by conjunction ($\wedge$).

[1] https://www.princeton.edu/ chaff/zchaff.html
[2] http://www.cse.nd.edu/Reports/2005/TR-2005-04.pdf

It's called *Conjunctive* Normal Form because the top level operation is conjunction.

The DIMACS format for CNF is

⟨DIMACS CNF⟩ ⟶ ⟨comment line⟩* ⟨problem line⟩ ⟨clause⟩*

⟨comment line⟩ ⟶ any line beginning with "c ".

⟨problem line⟩ ⟶ "p cnf " ⟨variable count⟩ " " ⟨clause count⟩

⟨variable count⟩ ⟶ the number of variables, n, as a decimal integer.

⟨clause count⟩ ⟶ the number of clauses, m, as a decimal integer.

⟨clause⟩ ⟶ ⟨literal⟩$^+$ 0

⟨literal⟩ ⟶ i or $-i$ where $1 \leq i \leq n$.

For example, $p \wedge (p \Rightarrow q) \wedge \neg q$ in CNF would be

$$\{\{p\}, \{\neg p, q\}, \{\neg q\}\}$$

and might be given to zchaff as

```
c p & (p => q) & ~q
p cnf 2 3
1 0
-1 2 0
-2 0
```

Note that zchaff doesn't report models. It only reports whether the formula it was given is satisfiable or not. It also reports some statistics. Several applications embed the underlying solver library and do get models out of it. This question is about formulating a problem in CNF and running a SAT checker, not about writing a SAT checker.

Your problem is to construct an n-bit *binary adder*[3] using n *full adder*[4] stages. You will need to use $4n + 1$ variables:

- $4k + 1$: C_k for $0 \leq k \leq n$ (carry)
- $4k + 2$: B_k for $0 \leq k < n$ (addend)
- $4k + 3$: A_k for $0 \leq k < n$ (augend)
- $4k + 4$: S_k for $0 \leq k < n$ (sum)

Each full adder stage satisfies these equations:

$$2C_{k+1} + S_k = A_k + B_k + C_k$$

which turns into the logical forms

$$\begin{aligned} S_k &\equiv A_k \oplus B_k \oplus C_k \\ C_{k+1} &\equiv (A_k \wedge B_k) \vee (C_k \wedge (A_k \vee B_k)) \end{aligned}$$

where $\oplus$ represents exclusive or.

You will generate the same *pattern* of clauses n times. This makes it easy to generate the clauses using a simple program. The easiest way to turn the two equations above into clauses is to use the Tseitin transformation[5]. If you do that, you will need a new variable for each operator, 2 $\equiv$, 2 $\oplus$, 2 $\wedge$, 2 $\vee$, making a total of $12n + 1$ variables.

[3] https://en.wikipedia.org/wiki/Adder_(electronics)
[4] https://en.wikipedia.org/wiki/Adder_(electronics)#Full_adder
[5] https://en.wikipedia.org/wiki/Tseytin_transformation

What should you test? You should test that some sums (A+B) have an answer and that some differences (S-A) have an answer, though zchaff won't tell you what they are. You should test that some wrong answers ($A_0 = 0$, $B_0 = 0$, $C_0 = 0$, $S_0 = 1$) are not satisfiable. You might try, for example, A = (001) $n/3$ times, S = (010) $n/3$ times, and see how zchaff's time grows with n. The main point of testing is to check that you are generating the right clauses, so most of your testing should be $n = 0$ (there is just one variable, C_0), $n = 1$, and $n = 2$.