universität innsbruck

Constraint Solving	SS 2024	LVA 703305
Week 2		March 15, 2024

Homework

1. Consider the formula (2 P)

$$\varphi = (\neg a \vee \neg b) \wedge (\neg a \vee b \vee \neg c) \wedge (b \vee g \vee h) \wedge (d \vee e) \wedge (d \vee \neg e \vee \neg f \vee \neg g) \wedge (c \vee \neg h \vee i) \wedge (\neg h \vee g \vee \neg i)$$

and the sequence of decisions and unit propagations

$$a^d \, \neg b \, \neg c \, \neg d^d \, e \, f^d \, \neg g \, h \, \neg i \parallel \varphi$$

   (a) Construct the conflict graph and find all cuts corresponding to a UIP. What are the corresponding backjup clauses, that can be learnt?

   (b) Use the first UIP to backjump as far as possible. What does the implication graph look like after the backjump?

2. The aim is to verify that the *crossover gadget* on slide 24 is correct.

   (a) Assume ϕ is the CNF of the crossover gadget. Specify certain formulas that must be satisfiable and further some formulas that must be unsatisfiable, in order to ensure correctness of the gadget. (2 P)

   (b) Verify the correctness of the gadget by calling a SAT solver. (0.5 P)

   (c) How can one replace the clause with 4 literals in the gadget to obtain a gadget that is an instance of planar 3-SAT. (0.5 P)

3. On slide 20 we have to enumerate all *legal* windows to encode the run of a NTM as a SAT problem. Consider the transition function Δ where: (2 P)

$$\Delta(p, \vdash) = \{(p, \vdash, R), (q, \vdash, R)\} \qquad \Delta(p, a) = \{(p, b, R), (q, a, L)\}$$
$$\Delta(q, a) = \{(q, c, L), (q, b, L)\} \qquad \Delta(q, b) = \{(p, b, R), (q, c, R)\}$$

   For all other states s and symbols f we have $\Delta(s, f) = \varnothing$. Which of the following windows is *legal*?

(i)
a	b	a
a	b	a

(ii)
a	c	a
b	c	a

(iii)
q	b	a
q	b	a

(iv)
b	a	a
p	a	a

(v)
a	p	$\vdash$
a	$\vdash$	q

(vi)
c	b	q
c	b	a

(vii)
a	b	c
q	b	c

(viii)
p	a	c
b	p	c

4. We want to construct a non-deterministic Turing machine to decide 3-SAT for CNFs with up to n variables in polynomial time. For a CNF containing the variables $a_1, \ldots, a_n$ we use the input alphabet $\Sigma = \{\#, \$, a_1, \overline{a_1}, a_2, \overline{a_2}, \ldots, a_n, \overline{a_n}\}$. A 3-CNF is then encoded on the input tape by listing each clause separated by $\#$, where each clause is a sequence of literals. The input is then terminated by the \$ symbol. For example the 3-CNF $(a_1 \vee \neg a_2 \vee a_3) \wedge (a_2 \vee a_4) \wedge (a_3 \vee \neg a_4)$ is represented on the tape as $\vdash \; a_1 \, \overline{a_2} \, a_3 \, \# \, a_2 \, a_4 \, \# \, a_3 \, \overline{a_4} \, \$$. The machine should proceed in two phases. In phase (1) it non-deterministically guesses one literal per clause which should evaluate to true, and in phase (2) it checks if that guess is valid. To this end you should:

(a) Define a non-deterministic Turing machine M for phase (1). (2 P)
Based on the above alphabet and encoding, it should guess one literal per clauses and removes the rest. For the CNF above an example run could look like:

$$\vdash a_1\, \overline{a_2}\, a_3 \,\#\, a_2\, a_4 \,\#\, a_3\, \overline{a_4}\,\$ \xrightarrow[M]{*} \vdash \text{␣␣}a_3\,\text{␣}a_2\,\text{␣␣␣}\overline{a_4}\,\$$$

(b) In phase (2) we check if the output of phase (1) represents a valid assignment. (1 P)
Explain how this check could be implemented. You do not have to define a full Turing machine.