CS4150

THE UNIVERSITY OF WARWICK

Fourth Year Examinations: Summer 2016

Decision Procedures

Time allowed: 2 hours.

Answer **THREE** questions: Question 1 and **TWO** out of Questions 2, 3, 4.

Read carefully the instructions on the answer book and make sure that the particulars required are entered on **each** answer book.

Calculators are not required and not permitted.

---

1. (a) Define conjunctive normal forms (CNF), clauses and literals of propositional logic. [6]

   (b) Describe Tseitin's encoding that translates arbitrary propositional formulas to equi-satisfiable CNF, with only a linear increase in formula size. What is the price to pay? [6]

   (c) Consider the following two restrictions of CNF:

      i. there is not more than one positive literal in each clause;
      ii. no clause has more than two literals.

      For each of these two restrictions, describe a polynomial-time procedure for deciding satisfiability. [12]

   (d) Show that every CNF can be converted to another CNF which is a conjunction of the two types of formula in the previous part. In other words, in the resulting CNF, all the clauses are either unary, or binary, or have not more than one positive literal. How many additional variables are necessary for the conversion? [6]

---

2. (a) Present a procedure for deciding satisfiability of conjunctions of equalities between terms that are built from variables and uninterpreted functions. The procedure should allow uninterpreted functions with several arguments. [20]

   (b) Describe by an example how equality logic with uninterpreted functions can be used to verify correctness of optimisations of pipelined circuits. The example should involve a circuit with a multiplexer and at least three pipeline stages. [15]

---

3. (a) The formulas of *difference logic* are built from Boolean operators and predicates of the form $x - y \leq c$ where x and y are some integer variables and c is an integer constant.

   Describe how to translate the following *job-shop scheduling problem* to the satisfiability problem for difference logic:

   > We are given a finite set of n jobs, each of which consists of a chain of operations. There is a finite set of m machines, each of which can handle at most one operation at a time. Each operation needs to be performed during an uninterrupted period of given length on a given machine.
   > We need to decide whether there exists a schedule, i.e. an allocation of the operations to time intervals on the machines, whose total length is smaller than or equal to a given constant. [15]

   (b) How can the satisfiability problem for difference logic be solved efficiently? [20]

---

4. (a) Prove that the following formula is unsatisfiable using the Nelson-Oppen procedure, where the variables are interpreted over the integers:

   $$g(f(x_1 - 2)) = x_1 + 2 \wedge g(f(x_2)) = x_2 - 2 \wedge x_2 + 1 = x_1 - 1.$$ [15]

   (b) A simple improvement to the Nelson-Oppen procedure for convex theories is to restrict the equality propagation step as follows:

   > We call a variable *local* when it appears only in the literals of a single theory.
   > If an equality $x = y$ is implied by the literals F_i of a theory T_i and not by the literals F_j of a theory T_j, then we propagate it to F_j only if both x and y are not local to F_i.

   Explain why this improvement is correct. [20]

---

1. (a) *[bookwork]* A formula is in conjunctive normal form if it is a conjunction of disjunctions of literals, i.e., it has the form $\bigwedge_i \left(\bigvee_j l_{i,j}\right)$, where $l_{i,j}$ is the jth literal in the ith clause (a clause is a disjunction of literals). A literal is either an atom or its negation. [6]

   (b) *[bookwork]* The price to be paid is n new Boolean variables, where n is the number of logical gates in the formula. In Tseitin's encoding, one new variable is added for every logical gate in the original formula, and several clauses to constrain the value of this variable to be equal to the gate it represents, in terms of the inputs to this gate. The original formula is satisfiable if and only if the conjunction of these clauses together with a new variable associated with the topmost operator is satisfiable.

   This is best illustrated with an example. Given a propositional formula $x_1 \implies (x_2 \wedge x_3)$, let us assign the variable a_2 to the AND gate (corresponding to the subexpression $x_2 \wedge x_3$) and a_1 to the IMPLICATION gate (corresponding to $x_1 \implies a_2$), which is also the topmost operator of this formula. We need to satisfy a_1 together with two equivalences:

$$a_1 \iff (x_1 \implies a_2) \qquad\qquad a_2 \iff (x_2 \wedge x_3).$$

   The first equivalence can be rewritten in CNF as

$$(a_1 \vee x_1) \;\wedge\; (a_1 \vee \neg a_2) \;\wedge\; (\neg a_1 \vee \neg x_1 \vee a_2)$$

   and the second equivalence can be rewritten in CNF as

$$(\neg a_2 \vee x_2) \;\wedge\; (\neg a_2 \vee x_3) \;\wedge\; (a_2 \vee \neg x_2 \vee \neg x_3).$$

   Thus, the overall CNF formula is the conjunction of the latter two formulas and the unit clause a_1 which represents the topmost operator. [6]

   (c) *[comprehension]*

      i. Horn formulas: literals in positive unit clauses are assigned true, and the rest are assigned false. The formula is satisfiable if no empty clause results. Why? Let l be a literal assigned true. All clauses containing l are satisfied. All clauses containing $\neg l$ still have negated literals. Hence they will be satisfied by setting them to false. This strategy is equivalent to performing Boolean constraint propagation starting, as usual, from the unit clauses. [6]

      ii. 2-CNF: construct a directed graph $G(V, E)$ in which the vertices V represent literals and an edge (u, v) represents the fact that u implies v. Hence, a 2-CNF clause contributes two edges. A 2-CNF formula is satisfiable if and only if no literal and its negation belong to the same strongly connected component of the graph. [6]

   (d) *[application]* Associate with each variable v a new variable a_v . Every time v appears positively in a clause, replace it with $\neg a_v$. Then, add constraints for $\neg a_v = v$, which is a 2-CNF formula $(a_v \vee v) \wedge (\neg a_v \vee \neg v)$. Note that this transformation leaves us with a formula in which all clauses contain only positive literals or are 2-CNF. We can relax the transformation and leave up to one positive literal in a clause—this will still be a combination of Horn and 2-CNF formulas. [6]

---

2. (a) *[bookwork, comprehension]*

   i. Build congruence-closed equivalence classes.

      A. Initially, put two terms t_1 , t_2 (either variables or uninterpreted-function instances) in their own equivalence class if $t_1 = t_2$ is a predicate in the given formula φ^{UF}. All other variables form singleton equivalence classes.

      B. Given two equivalence classes with a shared term, merge them. Repeat until there are no more classes to be merged.

      C. Compute the congruence closure: For two uninterpreted-function instances $F(t_1, \ldots, t_n)$ and $F(t'_1, \ldots, t'_n)$, if t_i and t'_i are in the same equivalence class for all $i \in \{1, \ldots, n\}$, then merge the classes of these two instances. Repeat until there are no more such instances.

   ii. If there exists a disequality $t_i = t_j$ in φ^{UF} such that t_i and t_j are in the same equivalence class, return 'Unsatisfiable'. Otherwise return 'Satisfiable'. [20]

   (b) *[bookwork, application]* Consider a pipelined circuit as follows. The input, denoted by I, is processed in the first stage. We model the combinational gates within the stages with uninterpreted functions, denoted by C, F, G, H, K, and D. For the sake of simplicity, we assume that they each impose the same delay. The circuit applies function F to the input I, and stores the result in latch L_1. This can be formalized as follows:

$$L_1 = F(I)$$

   The second stage computes values for L_2, L_3, and L_4:

$$\begin{aligned} L_2 &= L_1 \\ L_3 &= K(G(L_1)) \\ L_4 &= H(L_1) \end{aligned}$$

   The third stage contains a multiplexer. A multiplexer is a circuit that selects between two inputs according to the value of a Boolean signal. In this case, this selection signal is computed by a function C. The output of the multiplexer is stored in latch L_5:

$$L_5 = C(L_2)\,?\,L_3 \;:\; D(L_4)$$

   Observe that the second stage contains two functions, G and K, where the output of G is used as an input for K. Suppose that this is the longest path within the circuit.

We now aim to transform the circuit in order to make it work faster. This can be done in this case by moving the gates represented by K down into the third stage. Observe also that only one of the values in L_3 and L_4 is used, as the multiplexer selects one of them depending on C. We can therefore remove one of the latches by introducing a second multiplexer in the second stage. The circuit after these changes can be formalized as follows:

$$\begin{aligned}
L'_1 &= F(I)\\
L'_2 &= C(L'_1)\\
L'_3 &= C(L'_1)\,?\,G(L'_1)\ :\ H(L'_1)\\
L'_5 &= L'_2\,?\,K(L'_3)\ :\ D(L'_3)
\end{aligned}$$

The final result of the computation is stored in L_5 in the original circuit, and in L'_5 in the modified circuit. We can show that the transformations are correct by proving that for all inputs, the conjunction of the above equalities implies $L_5 = L'_5$. This proof can be automated by using a decision procedure for equalities and uninterpreted functions. [15]

---

3. (a) *[application]* Given a set of machines $M = \{m_1, \ldots, m_m\}$, we have that job J_i with $i \in \{1, \ldots, n\}$ is a sequence of n_i pairs of the form (machine, duration):

$$Ji = (m^i_1, d^i_1), \ldots, (m^i_{n_i}, d^i_{n_i}),$$

such that m^i_1, …, $m^i_{n_i}$ are elements of M. The durations can be assumed to be integer numbers. We denote by O the multiset of all operations from all jobs. For an operation $v \in O$, we denote its machine by $M(v)$ and its duration by $\tau(v)$. A schedule is a function that defines, for each operation v, its starting time $S(v)$ on its specified machine $M(v)$. A schedule S is feasible if the following three constraints hold:

- Firstly, for every pair of consecutive operations $v_i, v_j \in O$ in the same job, the second operation does not start before the first ends: $S(v_i) + \tau(v_i) \leq S(v_j)$.
- Secondly, every pair of different operations $v_i, v_j \in O$ scheduled on the same machine ($M(v_i) = M(v_j)$) is mutually exclusive:

$$S(v_i) + \tau(v_i) \leq S(v_j) \ \vee\ S(v_j) + \tau(v_j) \leq S(v_i).$$

- Finally, the length of the schedule S is defined as

$$\max_{v,v' \in O} S(v') + \tau(v') - S(v),$$

and this length needs to be smaller than or equal to a given constant C.

It is clear that the constraints can be formulated with difference logic. Note the disjunction in the second constraint. [15]

(b) *[bookwork, comprehension]* The Boolean structure can dealt with by the $\mathrm{DPLL}(T)$ framework, so we need to consider only the conjunctive fragment.

Given a set S of difference predicates, let the inequality graph G be the graph comprising of one edge (x, y) with weight c for every constraint of the form $x - y \leq c$ in S. It is known that S is satisfiable if and only if there is no negative cycle in G.

Therefore deciding a difference logic formula amounts to searching for a negative cycle in a graph. This can be done with the BellmanFord algorithm for finding the single-source shortest paths in a directed weighted graph, in time $O(|V| \cdot |E|)$. To make the graph single-source, we introduce a new node and add an edge with weight 0 from this node to each of the roots of the original graph. Although finding the shortest paths is not our goal, we exploit a side-effect of this algorithm: if there exists a negative cycle in the graph, the algorithm finds it and aborts. [20]

---

4. (a) *[application]* Purification results in an arithmetic formula over the integers,

$$x_2 + 1 = x_1 - 1 \,\wedge\, a_1 = x_1 - 2 \,\wedge\, a_2 = x_1 + 2 \,\wedge\, a_3 = x_2 - 2,$$

together with a formula in equality logic with uninterpreted functions,

$$g(f(a_1)) = a_2 \,\wedge\, g(f(x_2)) = a_3.$$

From the first formula, we deduce $a_1 = x_2$ and propagate this to the second formula. From

$$g(f(a_1)) = a_2 \,\wedge\, g(f(x_2)) = a_3 \,\wedge\, a_1 = x_2,$$

we deduce $a_2 = a_3$ and propagate this back to the first formula, which reveals a contradiction. [15]

(b) *[comprehension]* An informal argument is the following. The only way in which a local variable v can convey useful information to other theories is by being included in a pair of implied equations such as $v = x_1$ and $v = x_2$, where x_1 and x_2 are not local. But then the equation $x_1 = x_2$ is implied and propagated as well, which makes the first two equalities unnecessary. [20]
