School of Computing and Information Systems
COMP30026 Models of Computation Tutorial Week 4

26–28 August 2020

Plan

This is the last tutorial covering propositional logic. If you have time, do Exercise 34 and learn more about exclusive or normal form (XNF). Alternatively (or additionally), you may want to spend a bit of time on any more general questions you have, on the various logic concepts and/or Haskell. A reminder that Grok Worksheet 1 is due on 30 August.

The exercises

30. On the Island of Knights and Kn s always tell the truth. Knaves always lie. Y ays to you: "If I am a knight then my two fri l logic to determine whether this statement gives you any real information. What, if anything, can be deduced about A, B, and C?

11111. Low-level programming languages (including C) offer bitwise operators, including bitwise negation, conjunction, disjunction and exclusive or. Bitwise means the logical operation is applied bit for bit, across whole words, with 0 representing false and 1 representing true. In such languages an expr perator $\oplus$ is applied bit by bit to t

$$\begin{array}{cccccccc} \ldots & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ \ldots & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{array}$$

and this yields the binary representation of 31.

When the task is to swap the contents of two registers on a machine with few registers available, an old idea is the "xor trick". To swap the contents of R_1 and R_2, rather than call upon a third register, we can perform three xor operations on R_1 and R_2.

$$\begin{array}{lcl} R_1 & := & R_1 \oplus R_2 \\ R_2 & := & R_1 \oplus R_2 \\ R_1 & := & R_1 \oplus R_2 \end{array}$$

Explain why that has the desired effect.

32. Given these six formulas:

$$\begin{array}{ll} F_1: & (P \Rightarrow \neg Q) \wedge (P \Rightarrow \neg R) \\ F_2: & (P \Rightarrow \neg Q) \vee (P \Rightarrow \neg R) \\ F_3: & (Q \wedge R) \Rightarrow \neg P \\ F_4: & P \vee Q \vee R \\ F_5: & \neg P \vee \neg Q \vee \neg R \\ F_6: & P \Rightarrow \neg(Q \wedge R) \end{array}$$

group logically equivalent formulas together.

33. A graph colouring is an assignment of colours to nodes so that no edge in the graph connects two nodes of the same colour. The graph colouring problem asks whether a graph can be coloured using some fixed number of colours. The question is of great interest, because many scheduling problems are graph colouring problems in disguise. The case of three colours is known to be hard (NP-complete).

    How can we encode the three-colouring problem in propositional logic, in CNF to be precise? (One reason we might want to do so is that we can then make use of a SAT solver to determine colourability.) Using propositional variables

    - B_i to mean node i is blue,
    - G_i to mean node i is green,
    - R_i to mean node i is red;
    - E_{ij} to mean i and j are different but connected by an edge,

    write formulas in CNF for th

    (a) Every node (0 to
    (b) Every node has at most on
    (c) No two connected nodes have the same colour.

    For a graph with $n+1$ nodes, what is the size of the CNF formula?

34. In the Week 3 lecture we saw that conjunctive normal form (CNF) and disjunctive normal form (DNF) are not *canonical*. For example, $(\neg P \vee \neg Q \vee \neg R) \wedge (P \vee Q \vee R) \wedge (P \vee \neg R) \wedge (\neg Q \vee R)$ and the logically equivalent $P \wedge \neg Q$ are both in reduced CNF. It would be nice if equivalent formulas were syn cking would then become much simpler, jus

    It turns out that we can a nstead of or ($\vee$). XNF (exclusive-or normal form) can express every possible Boolean function. In XNF, a Boolean function is expressed as a sum of products, with addition cor ng to exclusive or, $\oplus$, and multiplication corresponding to conjunction, $P \wedge (Q \Leftrightarrow R)$ we write $P \oplus (P \wedge Q) \oplus (P \wedge R)$. Actually we usually ab $\oplus PQ \oplus PR$, with the understanding that a "product" PQ is a conjunction. We call an expression like this—one that is written as a sum of products—a *polynomial*. Each summand (like PQ) is a *monomial*.

    The connective $\neg$ is not used in XNF, only $\oplus$ and $\wedge$. To compensate, one of the truth value constants, $\mathbf{t}$, is needed. If F is a formula in XNF then $F \oplus \mathbf{t}$ is its negation. The other constant, $\mathbf{f}$, is not needed. This is because $\mathbf{f}$ is like zero: It is a "neutral element" for $\oplus$ (that is, $F \oplus \mathbf{f}$ is just F) and it is an "annihilator" for $\wedge$ ($F \wedge \mathbf{f}$ is just $\mathbf{f}$). So we can't have $\mathbf{f}$ *in* a monomial, because the entire monomial disappears if $\mathbf{f}$ enters it. And we don't need $\mathbf{f}$ *as* a monomial, because "adding" $\mathbf{f}$ really adds nothing.

    In this "Boolean ring" algebra, we are really dealing with arithmetic modulo 2, with $\wedge$ playing the role of multiplication, and $\oplus$ playing to role of addition. Given this, express these in XNF:

    (a) $\neg P$ (b) $P \wedge Q$ (c) $P \wedge \neg Q$ (d) $P \Leftrightarrow Q$ (e) $P \vee Q$ (f) $P \vee (Q \wedge R)$

    Note that $\wedge$ distributes over $\oplus$, but not the other way round. Also note carefully "cancelling out" properties. If we conjoin the monomials PQR and $PRST$, we get $PQRST$; that is, "duplicates disappear". However, if we have $F = \mu_1 \oplus \mu_2 \oplus \mu_3$ and $F' = \mu_2 \oplus \mu_3 \oplus \mu_4$ and we want to find $F \oplus F'$, then we find that "duplicates cancel out pairwise": $F \oplus F' = \mu_1 \oplus \mu_4$. This happens because $F \oplus F = \mathbf{f}$ for all Boolean functions F.

    Given this, turn these into XNF:

    (g) $\neg(P \oplus Q)$ (h) $(P \oplus Q) \wedge R$ (i) $(PQ \oplus PQR \oplus R) \wedge (P \oplus Q)$

    (j) $Q \wedge (P \oplus PQ \oplus \mathbf{t})$ (k) $Q \vee (P \oplus PQ)$