1 Horn-SAT and Renamable Horn-SAT

For parts (a) and (b), let us first establish that Horn clauses can be rewritten as implications. We show this by case analysis on the structure of Horn clauses.

A Horn clause with exactly one positive literal

In the case of a Horn clause with exactly one positive literal, e.g. $(\neg x_1 \vee \neg x_2 \vee ... \vee x_k)$, material implication (the rule of inference) allows us to rewrite this clause to $(x_1 \wedge x_2 \wedge ... \Rightarrow x_k)$. The single positive literal clause (x) is a special case, whose implication can be thought of as $(true \Rightarrow x)$.

A Horn clause with no positive literals

In the case of a Horn clause with no positive literals, e.g. $(\neg x_1 \vee \neg x_2 \vee ... \vee \neg x_k)$, we can think of its implication as $(x_1 \wedge x_2 \wedge ... \wedge x_k \Rightarrow false)$. This holds for the single negative literal clause (x), i.e. $(\neg x \Rightarrow false)$.

1.1 (a) A Linear Time Algorithm for Deciding the Satisfiability of HornSAT Formulae

Assumptions and Data Structures

Let us assume that we have a Horn-SAT formula, ϕ, in CNF. To begin, we initialize a collection of data structures to assist with tracking assignments to literals.

1. For each positive literal p in ϕ, we compute a list cl containing those clauses in which p appears as a negative literal.
2. We also compute an array nls such that $nls[c]$ returns the number of negative literals in clause c that have current truth value $false$ (0). A clause c can be processed for assignment by our algorithm if $nls[c] = 0$; in other words, its positive literal *must* be $true$ (1).
3. We also create an array pls such that $pls[c]$ returns the positive literal in clause c, if one exists.
4. Last, we instantiate a queue q that will contain clauses that are ready to be processed (those for which $nls[c] = 0$ as stated above).

Procedure

To begin, we initialize q to contain those clauses that are comprised of a single positive literal. In implication terms, we can think of these clauses as those of the form $(true \Rightarrow x)$. We *know* that these positive literals must eventually be assigned the value $true$ (1) if ϕ is satisfiable; moreover, $nls[c]$ of these clauses, by definition, evaluates to 0. We then enter a **while** loop guarded by the condition that q is not empty, that is, that there are still some clauses for which $nls[c] \neq 0$.

Within the body of the **while** loop, we enter a **for** loop iterating from 0 to the length of the queue, which is initially equivalent to the number of positive unit clauses in ϕ. In the body of the **for** loop, we pop off the head of the queue and store its value in a variable c_1. We then access the positive literal p_1 in c_1 by indexing into pls ($pls[c_1]$) and set its value to $true$. We also store p_1 in a variable named $next$.

We then use $next$ to index into ϕ and obtain the clause list cl related to c_1. Concretely, cl represents the list of all clauses c' in which p_1 (now set to $true$ (1)) appears as a negative literal. We "remove" this literal from these clauses by setting $nls[c'] = nls[c'] - 1$, in essence decrementing the count of negative literals in c' that have current truth assignment $false$ (0).

We then check in a conditional if $nls[c'] = 0$, meaning that c' is now ready to be processed. If so, we use c' to index into pls ($pls[c']$) to find its positive literal p'. If the positive literal is not yet assigned a truth value, then we set its current assignment to $true$ (1) and add c' to q to be processed in a later iteration of the **while** loop. If the positive literal is assigned $false$ (0), then we've hit a clause for which all negative literals have current truth value $true$(1) and its positive literal is $false$ (0). Therefore, we return UNSAT. The **while** loop continues to process entries in the queue until it is empty and, if this point is reached, we return SAT.

Algorithm 1 Linear Time HornSAT Satisfiability

```
procedure LinearHorn(φ)
    for all p in φ do
        Initialize:
        p.cl ← list[c s.t. p is a negative literal in c]
    end for
    Initialize:
    nls ← array[0..number of clauses in φ] of 0..max num of literals
    pls ← array[0..number of clauses in φ] of 0..max num of literals
    q ← queue of clauses comprised of a single positive literal

    while q.length ≠ 0 do
        for i = 0 to q.length do
            c_1 ← q.pop()
            pls[c_1] ← true; next ← pls[c_1]

            for all c' in φ[next].cl do          ▷ Iterate over all clauses in which p_1 is a negative literal
                nls[c'] ← nls[c'] − 1
                if nls[c'] = 0 then
                    p' ← pls[c']
                    if p' exists then
                        if p' ≠ false then
                            Set p' to true
                            q.push(c')
                        else
                            return UNSAT
                        end if
                    end if
                end if
            end for
        end for
    end while

    return SAT
end procedure
```

1.2 (b) A Polynomial Time Algorithm for Deciding Whether a CNF Formula is Renamable Horn

We start by giving a description of the logical procedure for determining whether or not a formula, F, is renamable Horn before giving the precise algorithm.

Let F be represented as a set of clauses $F = \{C_1, ..., C_m\}$, with each clause C_i composed of a set of literals $\{L_{i1}, ..., L_{il}\}$. A *renaming* of F would entail replacing all literals in F whose variable n also appears in a separate set $A = \{n_1, ..., n_n\}$