A disadvantage of the conjunctive normal form, or CNF, is checking whether a formula is satisfiable. In the worst case, an algorithm has to check every row of a truth table. Because the truth table grows exponentially with the number of propositions in a sequent, being computer scientists we are interested in more efficient algorithms.

Alfred Horn, in 1951, observed that a formula composed of only conjunction and implication can be checked by a concise algorithm. Such formulas are named Horn clauses after him.

Consider a clause that has a conjunction for its antecedent and a single proposition for its consequent. The basic concept is that such a clause is easy to check for satisfiability: we establish the truth or falsity of the antecedent, then of the consequent, and use the definition of implication to determine whether the clause is true or false.

Horn added contradiction and tautology to propositions, which greatly extends the applicability without complicating the satisfiability check. He also permitted sets of clauses to be simultaneously checked by permitting conjunctions.

Recall that $\bot$, pronounced “bottom”, is a false or contradictory formula.

Consider the symbol $\top$, pronounced “top”; we will use this to denote a true formula, which is a tautology.

In words, a proposition $P$ is a contradiction, a tautology, or a propositional symbol.

An antecedent $A$ is a conjunction of propositions $P_i$.

A clause $C$ is $A \rightarrow P$ for an antecedent $A$ and a proposition $P$.

A Horn formula $H$ is a conjunction of clauses $C_j$.

Symbolically, we can write the definition of a Horn clause using a kind of assignment statement that is common in symbolic processing:

\[
\begin{align*}
P & ::= \bot \mid \top \mid p \\
A & ::= P \mid P \land A \\
C & ::= A \rightarrow P \\
H & ::= C \mid (C) \land A
\end{align*}
\]
Examples of Horn formulas:

\[ p \land q \land r \rightarrow p \]
\[ p \land q \rightarrow \bot \]
\[ \top \rightarrow p \]
\[ (p_2 \land p_3 \land p_5 \rightarrow p_{13}) \land (\top \rightarrow p_5) \land (p_5 \land p_{11} \rightarrow \bot) \]

Counterexamples of Horn formulas:

\[ p \land q \land s \rightarrow \neg p \]
\[ \neg q \land r \rightarrow p \]
\[ p_2 \land p_3 \land p_5 \rightarrow p_{13} \land p_{27} \]

The textbook, among many other sources, gives an algorithm for checking the satisfiability of Horn clauses. It works by “marking” propositions that are definitely true, working through the implications until everything has been checked.

The algorithm can be thought of as a check of the opposite: if it ever finds that a contradiction \( \bot \) is “marked”, then the clauses are inconsistent and not satisfiable. It does not distinguish between situations where there is a definite satisfiable assignment or where one exists but has not been found; in later courses, this will be an important distinction because it relates to the computational time it takes to perform algorithms.

**Algorithm: Horn Satisfiability**

Function HORN (\( \phi \))

//precondition: \( \phi \) is a Horn formula

//postcondition: Decides satisfiability for \( \phi \)

{  
mark all occurrences of \( \top \) in \( \phi \)

while there is a conjunct \( p_1 \land p_2 \land \ldots p_n \rightarrow P \) of \( \phi \)

such that all \( p_j \) are marked but \( P \) is not

mark \( P \)

end while

if \( \bot \) is marked

return ‘unsatisfiable’

else

return ‘satisfiable’

end if

}

36
We can understand the algorithm by working through some examples. We will start with two simple examples, then turn to a more complicated example.

**Example #1:**

\[ \phi = (p \land q \land s \rightarrow p) \land (q \land r \rightarrow p) \land (p \land s \rightarrow s) \]

We see that:

- There is no occurrence of \( T \) in the formula \( \phi \)
- Nothing is marked
- Return: **satisfiable**

The algorithm knows there is a model that satisfies \( \phi \), but it does not attempt to find such a model.

By inspection, we can see that if we assign \( F \) to all of the propositions \( p, q, r, \) and \( s \), then the antecedents of every implication is false. This means that every implication is \( T \), so the conjunction is \( T \), so \( \phi \) is satisfied.

**Example #2:**

\[ \phi = (p \land q \land s \rightarrow \bot) \land (q \land r \rightarrow \bot) \land (p \land s \rightarrow \bot) \]

This is like Example #1 but now the consequents of the implications are all contradictions. We can see that the truth model we used will also apply — false antecedents imply any consequent — so this formula is also satisfiable.

It is worthwhile to go through this yourself and determine what the algorithm will do.
Example #3:

\[ \phi = (\top \rightarrow q) \land (\top \rightarrow s) \land (w \rightarrow \bot) \land (p \land q \land s \rightarrow v) \land (v \rightarrow s) \land (\top \rightarrow r) \land (r \rightarrow p) \]

For this formula, the algorithm will:

- Mark all \( \top \)
- Because \( \top \) is marked, it will mark \( q, s, r \)
- Because \( r \) is marked, it will mark \( p \)
- Because \( p, q, \) and \( s \) are marked, it will mark \( v \)
- All antecedents have been checked
- Return: **satisfiable**

Which conjunct does not have a truth model? What truth model makes this conjunct satisfiable?

In this example, the algorithm “almost” finds a model – even though it was not designed to do so.