### Satisfiability/validity of DNF and CNF

#### Satisfiability

Formulas in **DNF** can be checked in linear time. A formula in DNF is satisfiable if at least one of its conjunctions is satisfiable. A conjunction is satisfiable if for every atomic formula \( A \) the conjunction does not contain both \( A \) and \( \neg A \) as literals.

- **Satisfiable:** \( (\neg B \land A \land B) \lor (\neg A \land C) \)
- **Unsatisfiable:** \( (A \land \neg A \land B) \lor (C \land \neg C) \)

#### Validity

Formulas in **CNF** can be checked in linear time. A formula in CNF is valid if all its disjunctions are valid. A disjunction is valid if for some atomic formula \( A \) the disjunction contains both \( A \) and \( \neg A \) as literals (or the disjunction is empty.)

- **Valid:** \( (A \lor \neg A \lor B) \land (C \lor \neg C) \)
- **Not valid:** \( (A \lor \neg A) \land (\neg A \lor C) \)

### Efficient satisfiability checks

- **Theorem:** Satisfiability of formulas in **CNF** is NP-complete.
- **Theorem:** Validity of formulas in **DNF** is NP-complete.

- **In the following:**
  - A very efficient satisfiability check for the special class of Horn formulas.
  - Efficient satisfiability checks for arbitrary formulas in CNF: DPLL, resolution.
Horn formulas

A formula \( F \) in CNF is a **Horn formula** if every disjunction in \( F \) contains at most one positive literal.

**Notation:**

\[
(\neg A \lor \neg B \lor C) \quad \text{becomes} \quad (A \land B \rightarrow C)
\]

\[
(\neg A \lor \neg B) \quad \text{becomes} \quad (A \land B \rightarrow 0)
\]

\[
A \quad \text{becomes} \quad (1 \rightarrow A)
\]

Satisfiablity check for Horn formulas

**Input:** a Horn formula \( F \).

**for** every atomic formula \( A \) occurring in \( F \) **do**

**if** \( F \) has a subformula of the form \( (1 \rightarrow A) \) **then** mark every occurrence of \( A \) in \( F \)

**while** \( F \) has a subformula \( G \) of the form \( A_1 \land \ldots \land A_k \rightarrow B \) or \( A_1 \land \ldots \land A_k \rightarrow 0 \) **and** \( A_1, \ldots, A_k \) are already marked **and** \( B \) is not yet marked **do**

**if** \( G \) has the first form **then** mark every occurrence of \( B \)

**else** return "unsatisfiable" and **halt**

**return** "satisfiable" and **halt**

Correctness of the marking algorithm

**Theorem.** The marking algorithm is correct and halts after at most \( n \) iterations of the while loop, where \( n \) is the number of atomic formulas that occur in \( F \).

**Proof:** (a) **Termination:** after \( n \) iterations all atomic formulas are marked, and so the loop condition does not hold.

(b) If "unsatisfiable" then unsatisfiable.

Observe: if the algorithm marks \( A \), then \( A(A) = 1 \) for every assignment \( A \) such that \( A(F) = 1 \).

Assume \( A(A) = 1 \) for some \( A \). Let \( (A_1 \land \ldots \land A_k \rightarrow B) \) be the subformula causing "unsatisfiable". Since \( A_1, \ldots, A_k \) are marked, \( A(A_1) = \ldots = A(A_k) = 1 \). Then \( A(A_1 \land \ldots \land A_k \rightarrow 0) = 0 \) and so \( A(F) = 0 \), contradiction. So \( F \) has no satisfying assignments.

(c) If "satisfiable" then satisfiable.

We show that the assignment given by \( A(A_i) = 1 \) if \( A_i \) is marked after termination satisfies \( F \):

- In every \( (A_1 \land \ldots \land A_k \rightarrow B) \) either \( B \) is marked or at least one \( A_i \) is not marked.
- In every \( (A_1 \land \ldots \land A_k \rightarrow 0) \) at least one \( A_i \) is not marked (otherwise the algorithm would have terminated with "unsatisfiable").
Let $n$ be the number of atomic formulas in $F$.
Let $m$ be the length of $F$.
Step (1) can be executed in $O(nm)$ time (at most two scans of the formula for each variable).
The number of iterations of the while loop is bounded by $n$, and the runtime of an iteration is bounded by $m$.
Overall runtime: $O(nm)$.
In the next slides we sketch a faster $O(m)$ algorithm.

### Data structure:
- Array of conjuncts, each conjunct stored as a list. (e.g., $A_1 \land A_2 \rightarrow B$ stored as $A_1 \mapsto A_2 \mapsto B$)
- Array of size $n$, where the $i$-th element is a list of pointers to all occurrences of $A_i$ on left-hand-sides of conjuncts.
- Single-linked list $W$ of length at most $n$ to store the variables that have been marked but not yet processed.
- Bitvector $V$ of length $n$ to store the variables that have been marked.

### An $O(m)$ algorithm

1. $W, V = \{A | 1 \rightarrow A \text{ is conjunct of } F \}$  
   1, $O(m)$
2. while $W \neq \emptyset$  
   $n$, $O(n)$
3. pick (and delete) $A$ from $W$  
   $n$, $O(n)$
4. for each conjunct $G \rightarrow H$ s.t. $A$ occurs in $G$ do  
   $O(m)$, $O(m)$
5. delete $A$ from $G$  
   $O(m)$, $O(m)$
6. if $G$ is empty then  
   $O(m)$, $O(m)$
7. if $H = B$ and $B \notin V$ then add $B$ to $W, V$  
   $O(m)$, $O(m)$
8. else /*$H = 0$*/ return “unsatisfiable”  
   1, $O(1)$
9. return “satisfiable”  
   1, $O(1)$

For each line we give the number of times it is executed and the total time required by all executions together.

Correctness argument (informal):
The algorithm mimics the original one. Marking an atomic formula corresponds to adding it to the worklist.
MYCIN: Rule system for treatment of blood infections developed in the 1970s.

Beispiel:

IF the infection is primary-bacteremia
AND the site of the culture is one of the sterile sites
AND the suspected portal of entry is the gastrointestinal tract
THEN there is suggestive evidence (0.7) that infection is bacteroid.