Conversion to CNF

B_{1,1} \iff (P_{1,2} \lor P_{2,1})

1. Eliminate $\iff$, replacing $\alpha \iff \beta$ with $(\alpha \implies \beta) \land (\beta \implies \alpha)$.
   
   $$(B_{1,1} \implies (P_{1,2} \lor P_{2,1})) \land ((P_{1,2} \lor P_{2,1}) \implies B_{1,1})$$

2. Eliminate $\implies$, replacing $\alpha \implies \beta$ with $\neg \alpha \lor \beta$.
   
   $$(\neg B_{1,1} \lor P_{1,2} \lor P_{2,1}) \land (\neg (P_{1,2} \lor P_{2,1}) \lor B_{1,1})$$

3. Move $\neg$ inwards using de Morgan's rules and double-negation:
   
   $$(\neg B_{1,1} \lor P_{1,2} \lor P_{2,1}) \land (\neg (P_{1,2} \land \neg P_{2,1}) \lor B_{1,1})$$

4. Apply distributivity law ($\land$ over $\lor$) and flatten:
   
   $$(\neg B_{1,1} \lor P_{1,2} \lor P_{2,1}) \land (\neg P_{1,2} \lor B_{1,1}) \land (\neg P_{2,1} \lor B_{1,1})$$
Resolution algorithm

• Proof by contradiction, i.e., show $KB \land \neg \alpha$ unsatisfiable

```plaintext
function PL-Resolution(KB, \alpha) returns true or false

    clauses ← the set of clauses in the CNF representation of $KB \land \neg \alpha$
    new ← \{\}
    loop do
        for each $C_i, C_j$ in clauses do
            resolvents ← PL-Resolve($C_i, C_j$)
            if resolvents contains the empty clause then return true
            new ← new \cup resolvents
        if new \subseteq clauses then return false
    clauses ← clauses \cup new
```
Resolution example

- $KB = (B_{1,1} \iff (P_{1,2} \lor P_{2,1})) \land \neg B_{1,1}$
- To prove $\alpha = \neg P_{1,2}$
Definite clauses

- A **definite clause** is a disjunction with exactly one positive literal
- Equivalent to \((P_1 \land \ldots \land P_n) \Rightarrow Q\)

- Basis of logic programming (Prolog)
- Horn Clauses more general (at most one positive literal),
- Efficient (linear-time) complete inference through *forward chaining* and *backward chaining*
Forward chaining

- Idea: fire any rule whose premises are satisfied in the $KB$,
  - add its conclusion to the $KB$, until query is found

\[
\begin{align*}
P & \Rightarrow Q \\
L \land M & \Rightarrow P \\
B \land L & \Rightarrow M \\
A \land P & \Rightarrow L \\
A \land B & \Rightarrow L \\
A & \\
B &
\end{align*}
\]
Example of Forward Chaining

\[ P \Rightarrow Q \]
\[ L \land M \Rightarrow P \]
\[ B \land L \Rightarrow M \]
\[ A \land P \Rightarrow L \]
\[ A \land B \Rightarrow L \]
\[ A \]
\[ B \]
Forward chaining algorithm

function PL-FC-ENTAILS?(KB, q) returns true or false

local variables: count, a table, indexed by clause, initially the number of premises
inferred, a table, indexed by symbol, each entry initially false
agenda, a list of symbols, initially the symbols known to be true

while agenda is not empty do
    p ← POP(agenda)
    unless inferred[p] do
        inferred[p] ← true
        for each Horn clause c in whose premise p appears do
            decrement count[c]
            if count[c] = 0 then do
                if HEAD[c] = q then return true
            end if
        end for
    end unless
    PUSH(HEAD[c], agenda)
return false

• Forward chaining is sound and complete for Horn KB
Proof of completeness

- FC derives every atomic sentence that is entailed by $KB$
  1. FC reaches a fixed point where no new atomic sentences are derived
  2. Consider the final state as a model $m$, assigning true/false to symbols
  3. Every clause in the original $KB$ is true in $m$
     \[ a_1 \land \ldots \land a_k \implies b \]
  4. Hence $m$ is a model of $KB$
  5. If $KB \models q$, $q$ is true in every model of $KB$, including $m$
Backward chaining

Idea: work backwards from the query \( q \):

- to prove \( q \) by BC,
  - check if \( q \) is known already, or
  - prove by BC all premises of some rule concluding \( q \)

Avoid loops: check if new subgoal is already on the goal stack

Avoid repeated work: check if new subgoal

1. has already been proved true, or
2. has already failed
Forward vs. backward chaining

• FC is **data-driven**, automatic, unconscious processing,
  • e.g., object recognition, routine decisions

• May do lots of work that is irrelevant to the goal

• BC is **goal-driven**, appropriate for problem-solving,
  • e.g., Where are my keys? How do I get into a PhD program?

• Complexity of BC can be **much less** than linear in size of KB