Artificial Intelligence

Lecturer 4 – Constraint Satisfaction Problems & Logical Agents

Brigitte Jaumard
Dept of Computer Science and Software Engineering
Concordia University
Montreal (Quebec) Canada

Constraints Satisfaction Problems (CSPs)

- CSPs example
- Backtracking search
- Problem structure
- Local search for CSPs

Constraint Satisfaction Problems

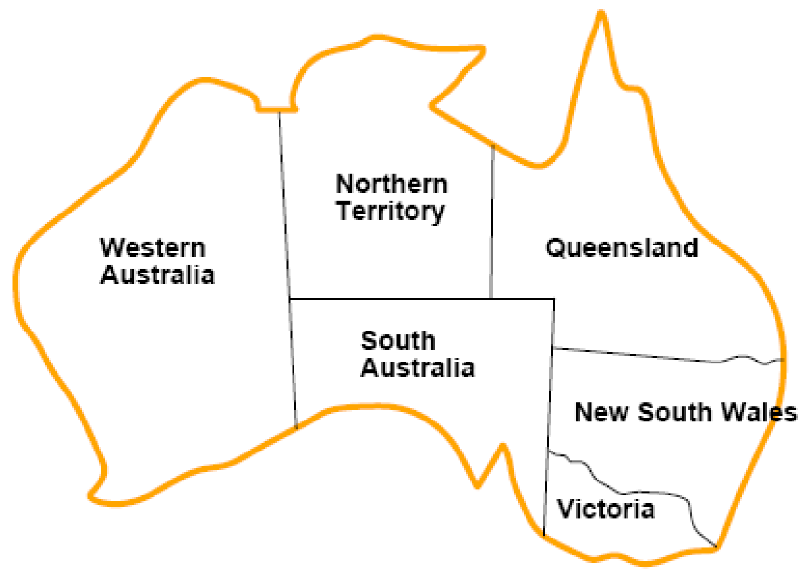
- **General Idea**
  - **Factored representation** for each state: a set of variables, each of which has a value.
  - Problem is solved when each variable has a value that satisfies all the constraints on the variable.
  - A problem described this way: a **constraint satisfaction problem**, or CSP.
- CSP search algorithms use ***general-purpose*** rather than ***problem-specific*** heuristics
- **Key idea:** eliminate large portions of the search space all at once by identifying variable/value combinations that violate the constraints.

Definition: Constraint Satisfaction Problem

- A constraint satisfaction problem consists of three components, X, D, and C
  - X is a set of variables, $\{X_1,...,X_n\}$
  - D is a set of domains, $\{D_1, . . . , D_n\}$, one for each variable.
  - C is a set of constraints that specify allowable combinations of values.

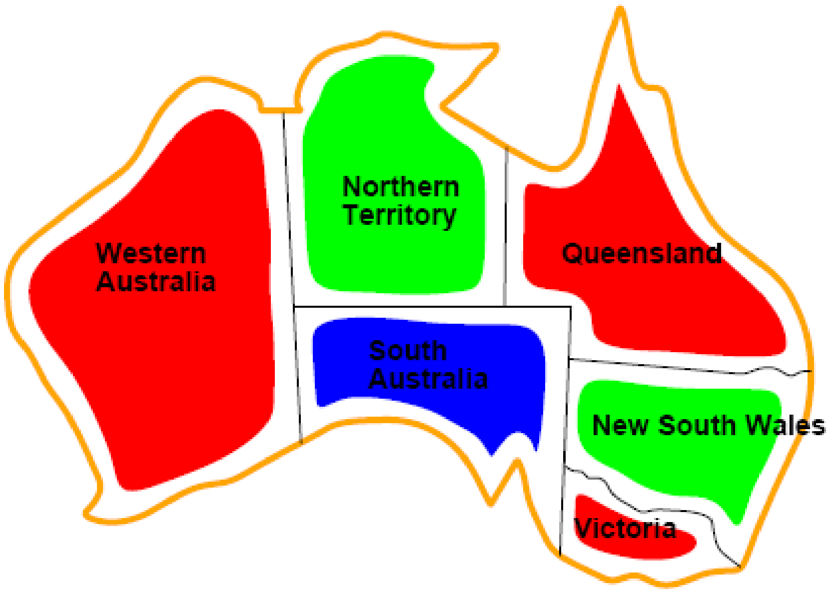
Example: Map Coloring

- Variables
  - WA, NT, Q, NSW, V , SA
- Domain
  - D_i = {red, green, blue}
- Constraint
  - Neighbor regions must have different colors
    - Color(WA) ≠ color(NT)
    - Color(WA) ≠ color(SA)
    - Color(NT) ≠ color(SA)
    - …



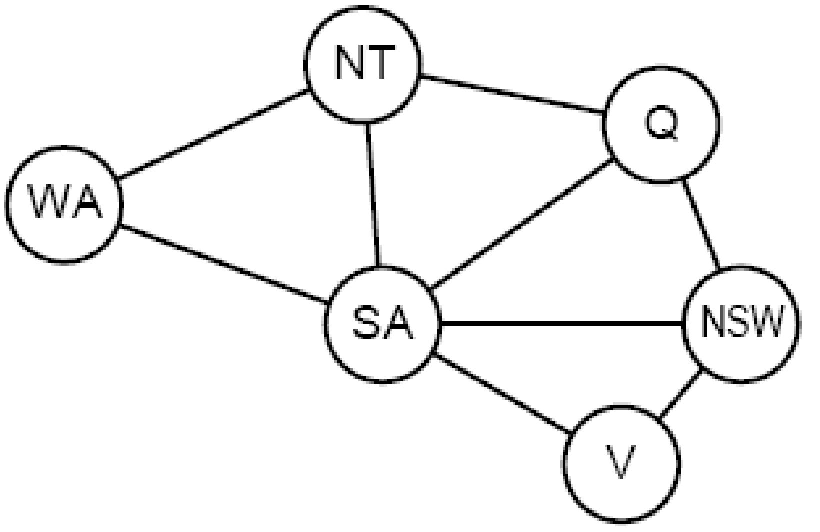
Example: Map Coloring

- Solution is an assignment of variables satisfying all constraints
  - WA=red, and
  - NT=green, and
  - Q=red, and
  - NSW=green, and
  - V=red, and
  - SA=blue



Constraint Graph

- Binary CSPs
  - Each constraint relates at most two variables
- Constraint graph
  - Node is variable
  - Edge is constraint



Varieties of CSPs

- Discrete variables
  - Finite domain, e.g, SAT Solving
  - Infinite domain, e.g., work scheduling
    - **Variables** is start/end of working day
    - **Precedence constraints:** Constraint language, e.g., $StartJob_1 + 5 \leq StartJob_3$
    - Linear constraints are decidable, non-linear constraints are undecidable
- Continuous variables
  - e.g., start/end time of observing the universe using Hubble telescope
  - Linear constraints are solvable using Linear Programming

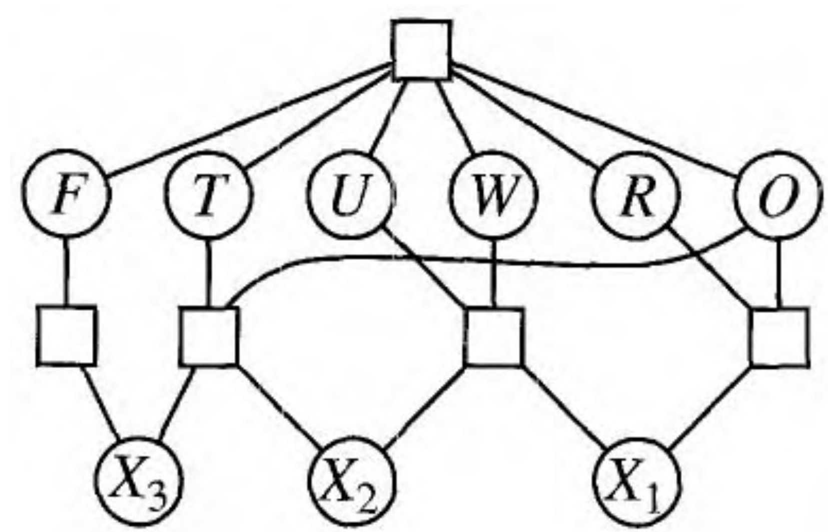
Varieties of Constraints

- **Single-variable constraints**
  - e.g., SA ≠ green
- **Binary constraints**
  - e.g., SA ≠ WA
- **Multi-variable constraints**
  - Relate at least 3 variables
- **Soft constraints**
  - Priority, e.g., red better than green
  - Cost function over variables
- **Disjunctive constraints**
  - Four workers install wheels, but need to share a tool that helps to put the axle in place
  - $Axle_F$ and $Axle_B$ must not overlap: either one comes first, or the other takes place

Example: Cryptarithmetic Puzzle

- Variables
  - F,T,O,U,R,W, X_1,X_2,X_3
- Domain
  - {0,1,2,3,4,5,6,7, 8,9}
- Constraints
  - Alldiff(F,T,O,U,R,W)
  - $O+O = R+10*C_1$
  - $C_1+W+W= U+10*C_2$
  - $C_2+T+T= O+10*C_3$
  - $C_3=F$

```
   TWO
+  TWO
------
  FOUR
```



Real World CSP

- Assignment
  - E.g., who teach which class
- Scheduling
  - E.g., when and where the class takes place
- Hardware design
- Spreadsheets
- Transport scheduling
- Manufacture scheduling

CSPs by Standard Search

- State
  - Defined by the values assigned so far
- Initial state
  - The empty assignment
- Successor function
  - Assign a value to a unassigned variable that does not conflict with current assignment
    - Fail if no legal assignment
- Goal test
  - All variables are assigned and no conflict

CSP by Standard Search

- Every solution appears at depth *d* with *n* variables
  - Use depth-first search
- Path is irrelevant
- Number of leaves
  - $n!d^n$
    - Two many

Backtracking Search

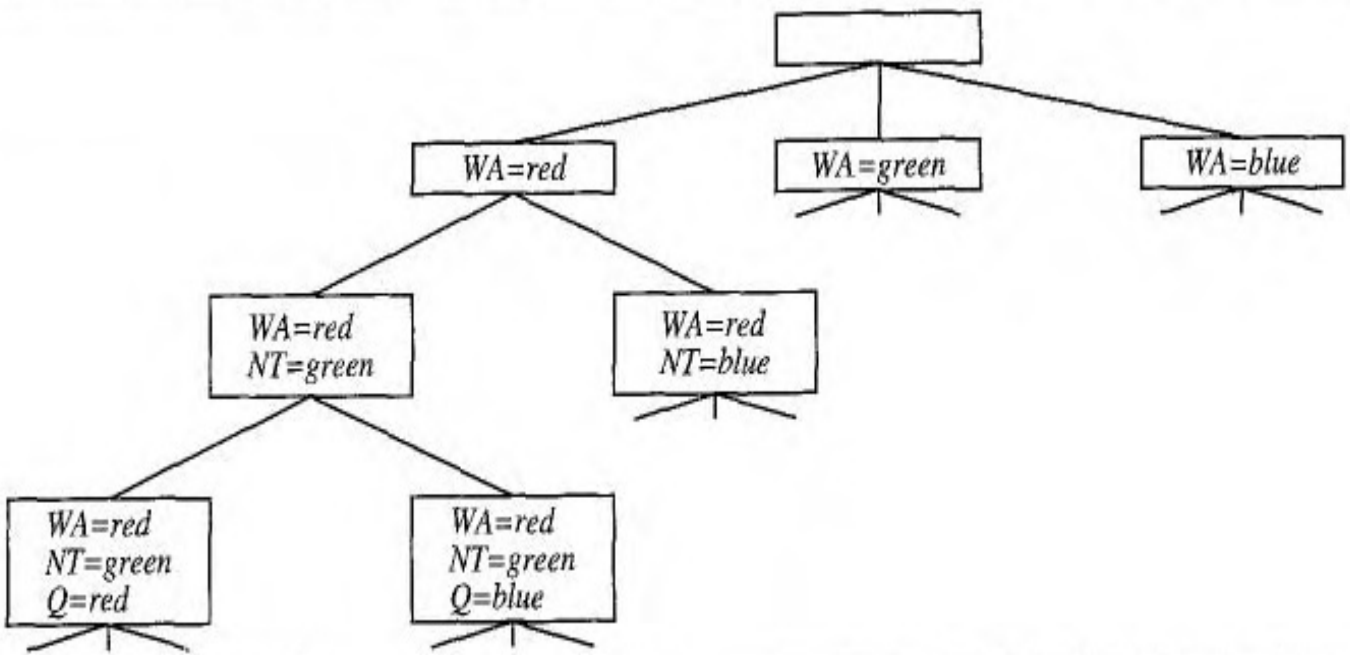
- Variable assignments are commutative, e.g.,
  - {WA=red, NT =green}
  - {NT =green, WA=red}
- Single-variable assignment
  - Only consider one variable at each node
  - d^n leaves
- Backtracking search
  - Depth-first search+ Single-variable assignment
- Backtracking search is the basic uninformed algorithm for CSPs
  - Can solve n-Queen with $n = 25$

Backtracking Search Algorithm

```
function BACKTRACKING-SEARCH(csp) returns solution/failure
   return RECURSIVE-BACKTRACKING({ }, csp)

function RECURSIVE-BACKTRACKING(assignment, csp) returns soln/failure
   if assignment is complete then return assignment
   var ← SELECT-UNASSIGNED-VARIABLE(VARIABLES[csp], assignment, csp)
   for each value in ORDER-DOMAIN-VALUES(var, assignment, csp) do
       if value is consistent with assignment given CONSTRAINTS[csp] then
           add {var = value} to assignment
           result ← RECURSIVE-BACKTRACKING(assignment, csp)
           if result ≠ failure then return result
           remove {var = value} from assignment
   return failure
```

Backtracking Search Algorithm



Improving Backtracking Search

- Which variable should be assigned next?
- In what order should its values be tried?
- Can we detect inevitable failure early?
- Can we take advantage of problem structure?

Choosing Variables

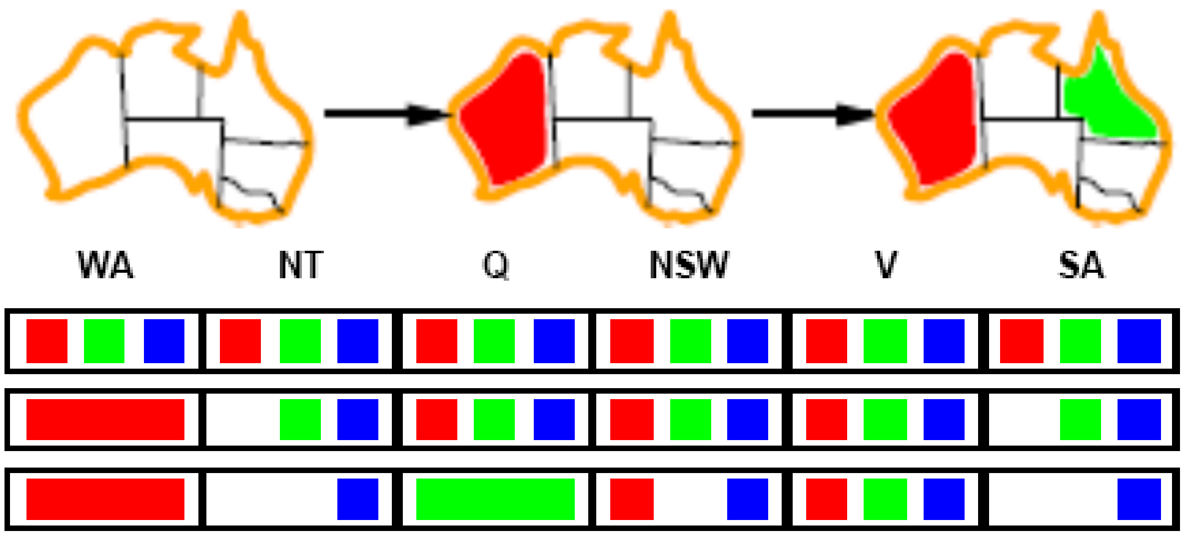
- Minimum remaining values (MRV)
  - Choose the variable with the fewest legal values
- Degree heuristic
  - Choose the variable with the most constraints on remaining variables

Choosing Values

- Least constraining value (LCV)
  - Choose the least constraining value
    - the one that rules out the fewest values in the remaining variables
- Keep track of remaining legal values for unassigned variables
  - Terminate search when any variable has no legal values

Forward Checking

- Constraint propagation



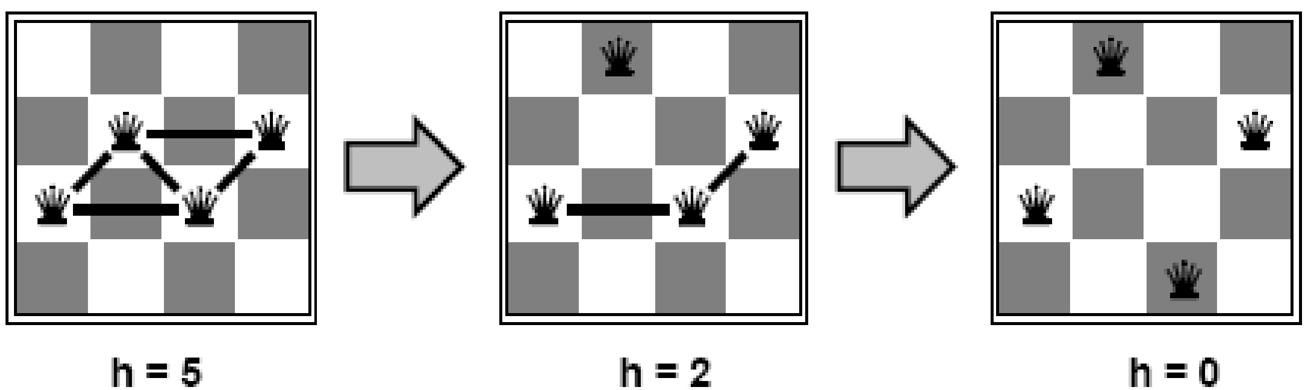
  - NT and SA cannot both be blue
- Simplest form of propagation makes each arc consistent
  - X -> Y is consistent iff for each value x of X there is some allowed value y for Y

Iterative Algorithms for CSPs

- Hill-climbing, Simulated Annealing can be used for CSPs
  - Complete state, e.g., all variables are assigned at each node
- Allow states with unsatisfiable constraints
- Operators reassign variables
- Variable selection
    - Random
- Value selection by min-conflicts heuristic
  - Choose value that violates the fewest constraints
    - i.e., hill climbing with h(n) = total number of violated constraints

Example: 4-Queens

- State: 4 queens in four columns (4*4 = 256 states)
- Operators: move queen in column
- Goal test: no attacks
- Evaluation: h(n) = number of attacks



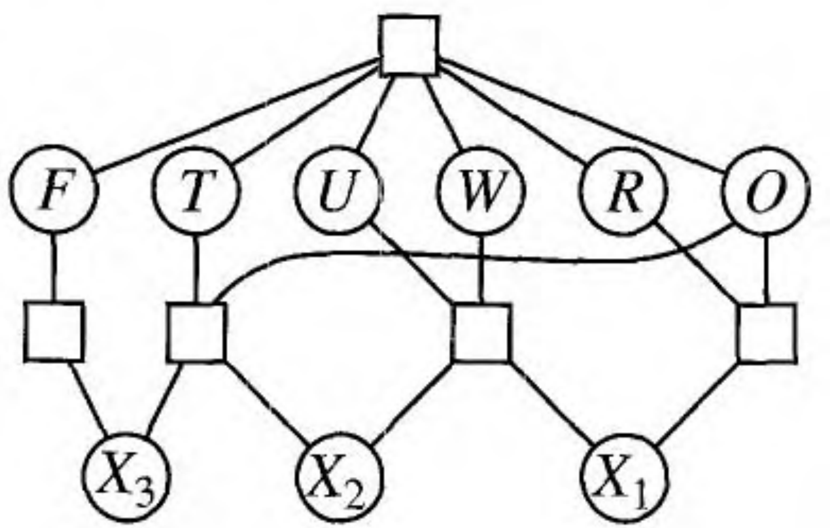
Summary

- CSPs are a special kind of problem:
  - states defined by values of a fixed set of variables
  - goal test defined by constraints on variable values
- Backtracking = depth-first search with one variable assigned per node
- Variable ordering and value selection heuristics help significantly
- Forward checking prevents assignments that guarantee later failure
- Constraint propagation (e.g., arc consistency) does additional work to constrain values and detect inconsistencies
- The CSPs representation allows analysis of problem structure
- Tree-structured CSPs can be solved in linear time
- Iterative min-conflicts is usually effective in practice

Exercice

- Solve the following cryptarithmetic problem by combining the heuristics
  - Constraint Propagation
  - Minimum Remaining Values
  - Least Constraining Values

$$
\begin{array}{r}
TWO \\
+\ \ TWO \\
\hline
FOUR
\end{array}
$$



Exercice

- $O+O = R+10*C_1$
- $C_1+W+W= U+10*C_2$
- $C_2+T+T= O+10*C_3$
- $C_3= F$

1. Choose C_3: domain {0,1}
2. Choose $C_3=1$: use constraint propagation $F\neq 0$
3. F = 1
4. Choose C_2: C_1 and C_2 have the same remaining values
5. Choose $C_2=0$
6. Choose C_1: C_1 has Minimum remaining values (MRV)
7. Choose $C_1=0$
8. Choose O: O must be even, less than 5 and therefore has MRV (T+T=O +10 ; O+O=R+10*0)
9. Choose O = 4
10. R=8
11. T=7
12. Choose U: U must be even, less than 9
13. U=6: constraint propagation
14. W=3

Reading & Suggested Exercises

- Chapter 6: 6.3, 6.7, 6.8

LOGICAL AGENTS

Outline

- What is Logic
- Propositional Logic
  - Syntax
  - Semantic
- Inference in Propositional Logic
  - Forward Chaining
  - Backward Chaining

Knowledge-based Agents

- Know about the world
  - They maintain a collection of facts (sentences) about the world, their Knowledge Base, expressed in some formal language.
- Reason about the world
  - They are able to derive new facts from those in the KB using some inference mechanism.
- Act upon the world
  - They map percepts to actions by querying and updating the KB.

What is Logic ?

- A logic is a triplet <L,S,R>
  - **L**, the language of the logic, is a class of sentences described by a precise syntax, usually a formal grammar
  - **S**, the logic's semantic, describes the meaning of elements in L
  - **R**, the logic's inference system, consisting of derivation rules over L
- Examples of logics:
  - Propositional, First Order, Higher Order, Temporal, Fuzzy, Modal, Linear, …

Propositional Logic

- Propositional Logic is about facts in the world that are either true or false, nothing else
- Propositional variables stand for basic facts
- Sentences are made of
  - propositional variables (A,B,…),
  - logical constants (TRUE, FALSE), and
  - logical connectives (not,and,or,..)
- The meaning of sentences ranges over the Boolean values {True, False}
  - Examples: It’s sunny, John is married

Language of Propositional Logic

- Symbols
  - Propositional variables: A,B,…,P,Q,…
  - Logical constants: TRUE, FALSE
  - Logical connectives:
    $$\neg, \wedge, \vee, \Rightarrow, \Leftrightarrow$$
- Sentences
  - Each propositional variable is a sentence
  - Each logical constant is a sentence
  - If α and β are sentences then the following are sentences
    $$(\alpha), \neg\alpha, \alpha \wedge \beta, \alpha \vee \beta, \alpha \Rightarrow \beta, \alpha \Leftrightarrow \beta$$

Formal Language of Propositional Logic

- Symbols
  - Propositional variables: A,B,…,P,Q,…
  - Logical constants: T, F
  - Logical connectives: $\neg, \wedge, \vee, \Rightarrow, \Leftrightarrow$
- Formal Grammar
  - Sentence -> Asentence | Csentence
  - Asentence -> TRUE | FALSE | A | B|…
  - Csentence -> (Sentence) | $\neg$ Sentence | Sentence Connective Sentence
  - Connective -> $\neg \mid \wedge \mid \vee \mid \Rightarrow \mid \Leftrightarrow$

Semantic of Propositional Logic

- The meaning of TRUE is always True, the meaning of FALSE is always False
- The meaning of a propositional variable is either True or False
  - depends on the interpretation
    - assignment of Boolean values to propositional variables
- The meaning of a sentence is either True or False
  - depends on the interpretation

Semantic of Propositional Logic

- True table

P	Q	Not P	P and Q	P or Q	P implies Q	P equiv Q
False	False	True	False	False	True	True
False	True	True	False	True	True	False
True	False	False	False	True	False	False
True	True	False	True	True	True	True

$a \Rightarrow b \Leftrightarrow \neg a \vee b \Leftrightarrow \neg b \Rightarrow \neg a$

Semantic of Propositional Logic

- Satisfiability
  - A sentence is satisfiable if it is true under some interpretation
  - Ex: P or H is satisfiable

    P and ¬P is unsatisfiable (not satisfiable)
- Validity
  - A sentence is valid if it is true in every interpretation
  - Ex: ((P or H) and ¬ H) => P is valid

    P or H is not valid

Semantic of Propositional Logic

- Entailment
  - Given
    - A set of sentences Γ
    - A sentence ψ
  - **Logical Entailment**

$$\Gamma \models \psi$$

    if and only if every interpretation that makes all sentences in Γ true also makes ψ true
  - We said that Γ entails ψ

Semantic of Propositional Logic

- Satisfiability vs. Validity vs. Entailment
  - ψ is valid iff True $\models \psi$ (also written $\models \psi$)
  - ψ is unsatisfiable iff $\psi \models$ False
  - $\Gamma \models \psi$ iff $\Gamma \cup \{\neg \psi\}$ is unsatisfiable

Inference in Propositional Logic

- Forward Chaining
- Backward Chaining

Forward Chaining

- Given a set of rules, i.e., formulae of the form

$$p_1 \wedge p_2 \wedge \ldots \wedge p_n \Rightarrow q$$

  and a set of known facts, i.e., formulae of the form

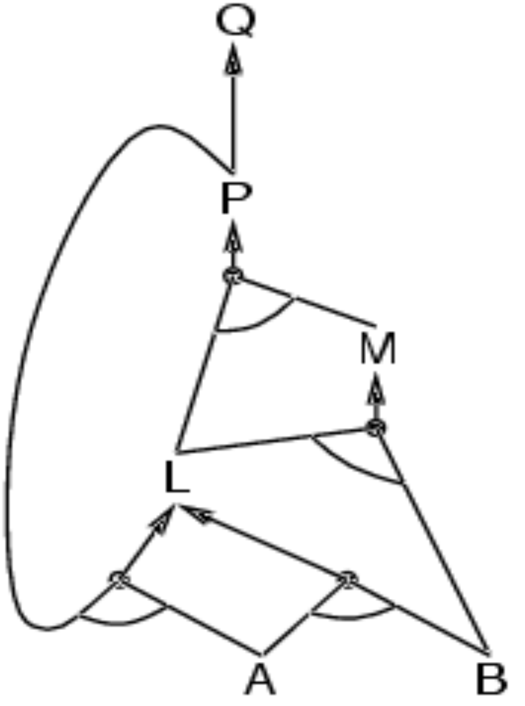
$$q, r, \ldots$$

- A new fact *p* is added
- Find all rules that have *p* as a premise
- If the other premises are already known to hold then
  - add the consequent to the set of know facts, and
  - trigger further inferences

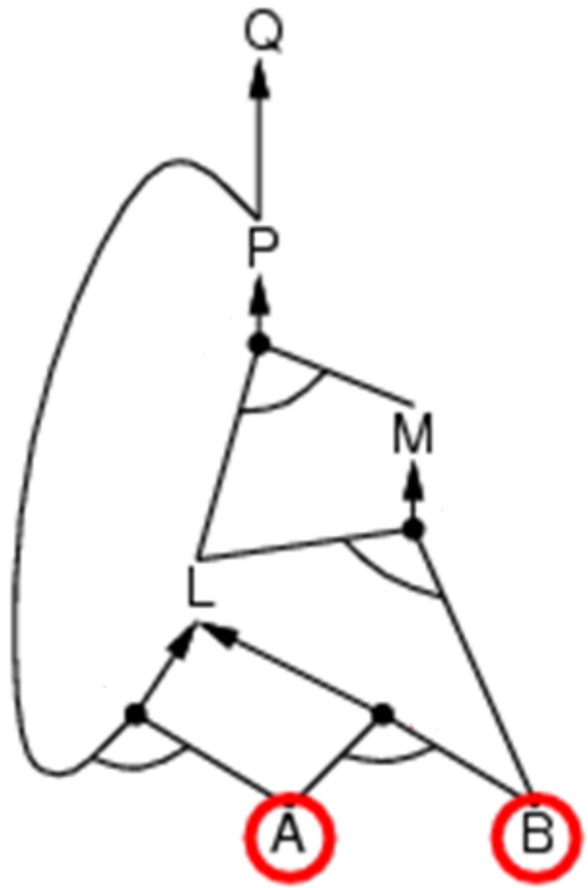
Forward Chaining

- Example

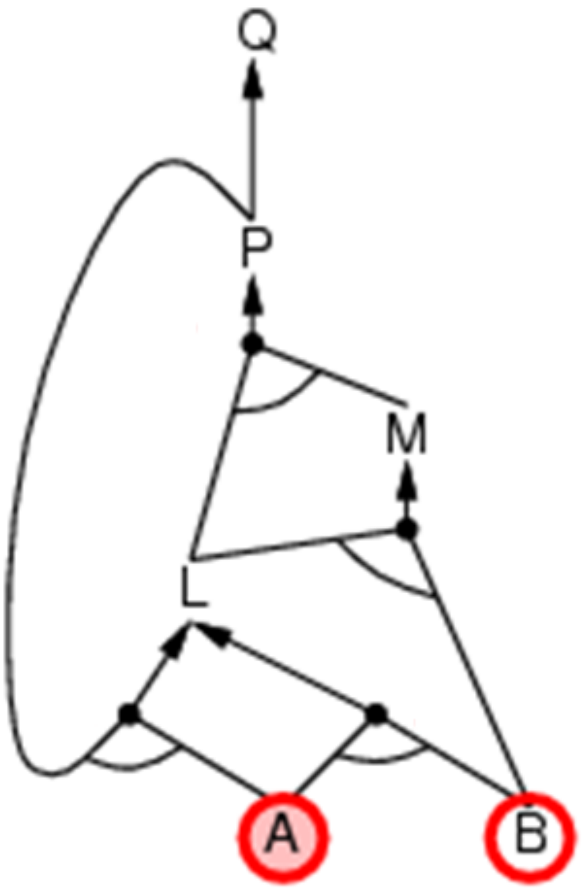
$$P \Rightarrow Q$$
$$L \wedge M \Rightarrow P$$
$$B \wedge L \Rightarrow M$$
$$A \wedge P \Rightarrow L$$
$$A \wedge B \Rightarrow L$$
$$A$$
$$B$$



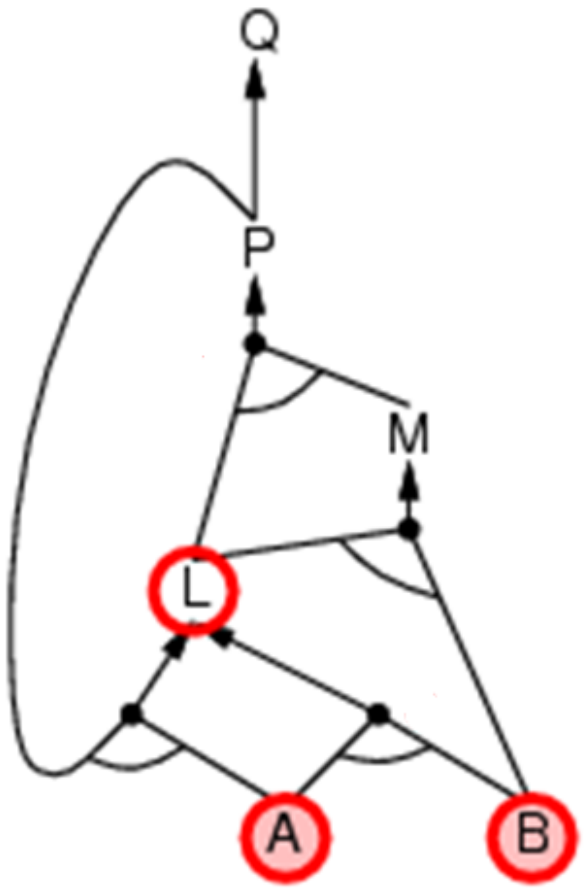
Forward Chaining



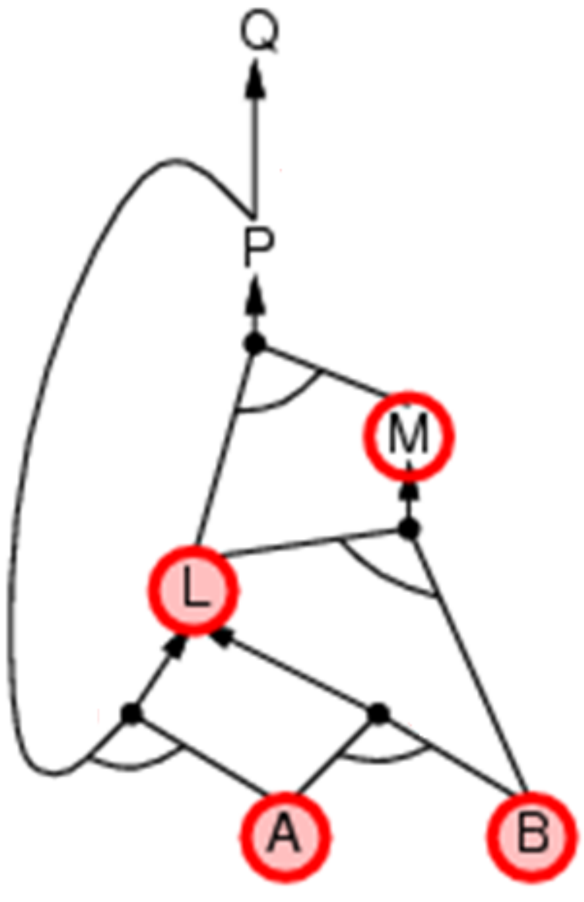
Forward Chaining



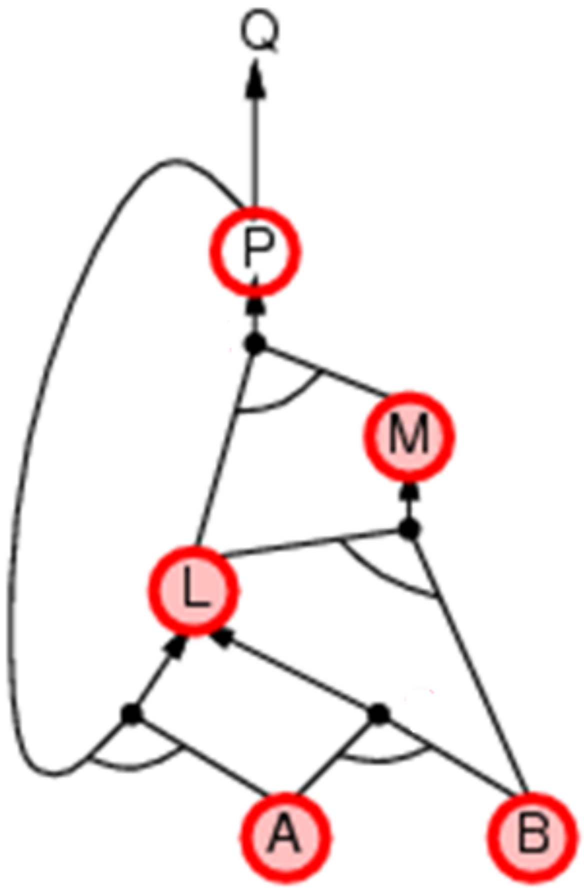
Forward Chaining



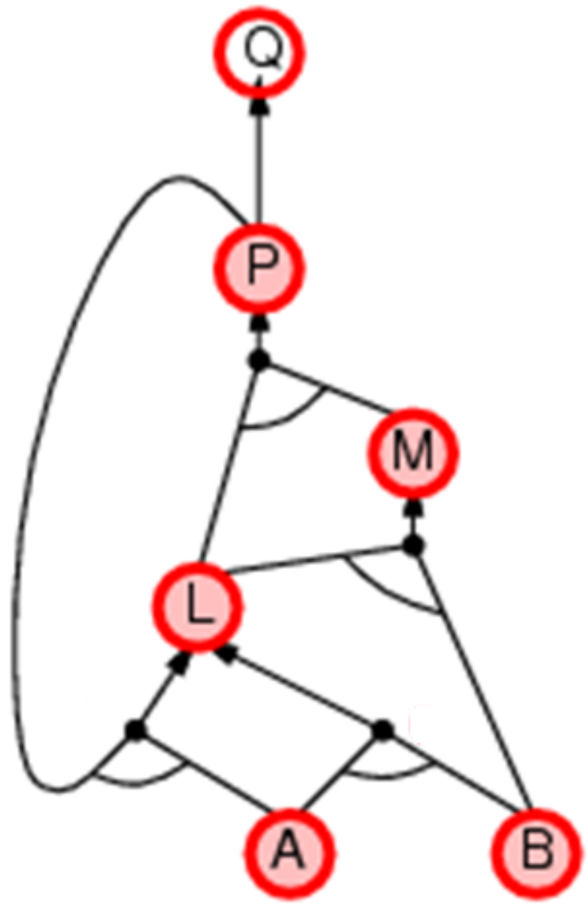
Forward Chaining



Forward Chaining



Forward Chaining



Forward Chaining

- Soundness
  - Yes
- Completeness
  - Yes

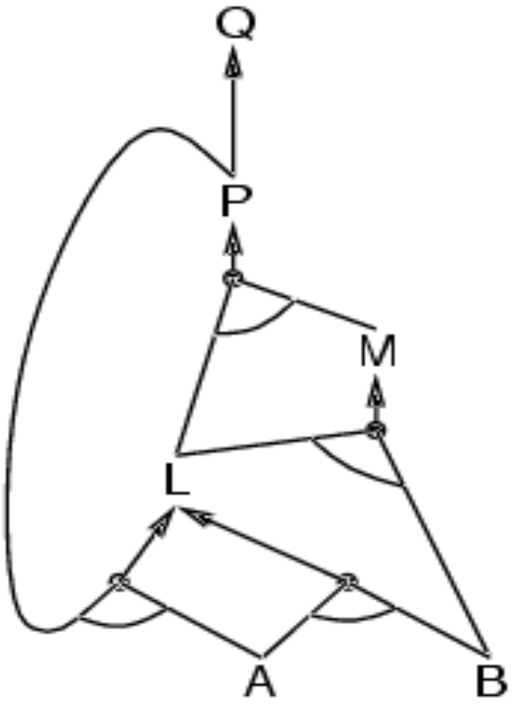
Backward Chaining

- Given a set of rules, and a set of known facts
- We ask whether a fact *P* is a consequence of the set of rules and the set of known facts
- The procedure check whether *P* is in the set of known facts
- Otherwise find all rules that have *P* as a consequent
  - If the premise is a conjunction, then process the conjunction conjunct by conjunct

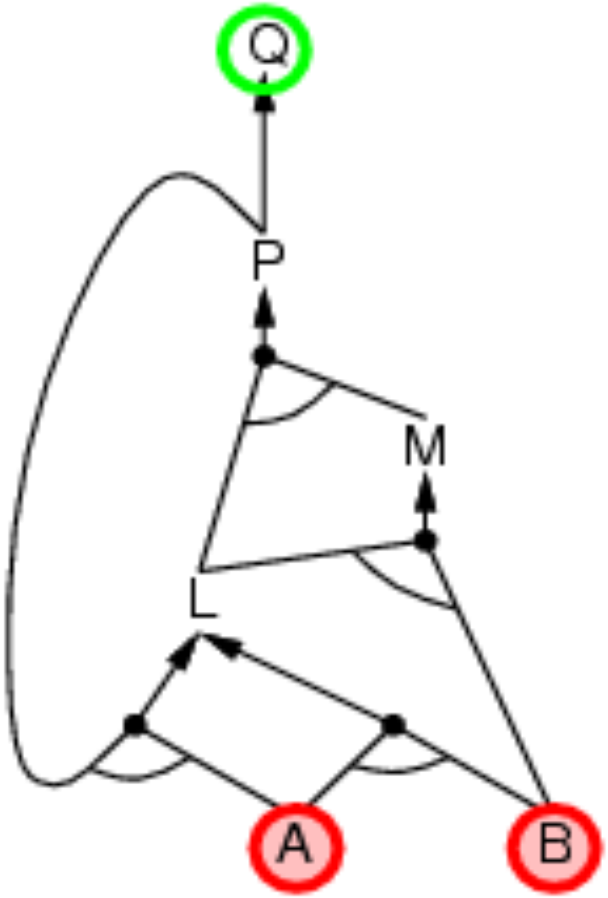
Backward Chaining

- Example

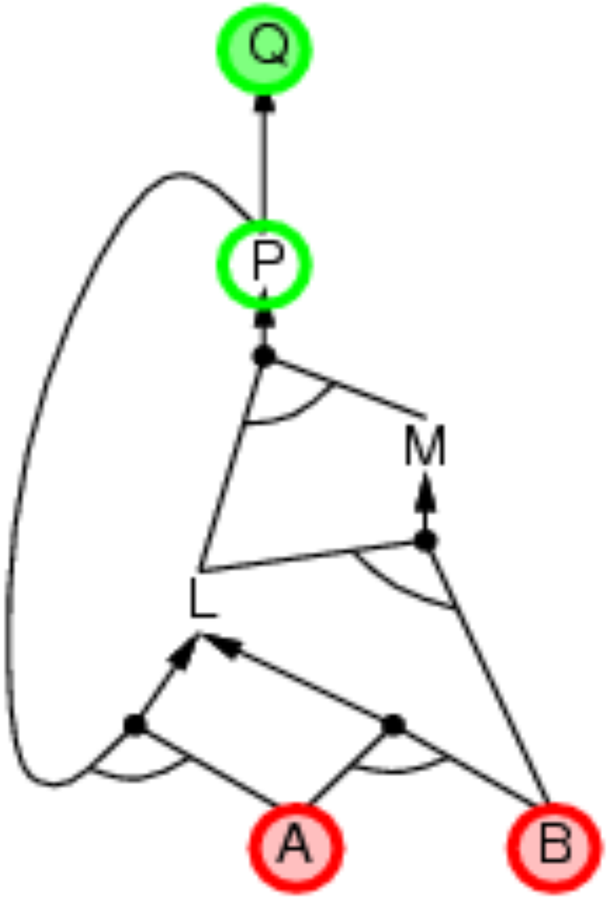
$$\begin{aligned} P &\Rightarrow Q \\ L \wedge M &\Rightarrow P \\ B \wedge L &\Rightarrow M \\ A \wedge P &\Rightarrow L \\ A \wedge B &\Rightarrow L \\ A & \\ B & \end{aligned}$$



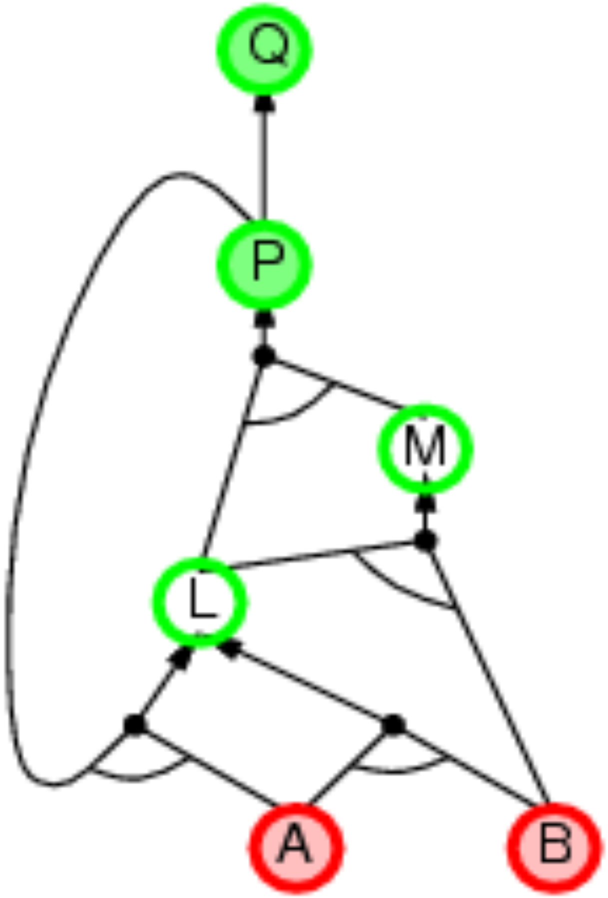
Backward Chaining



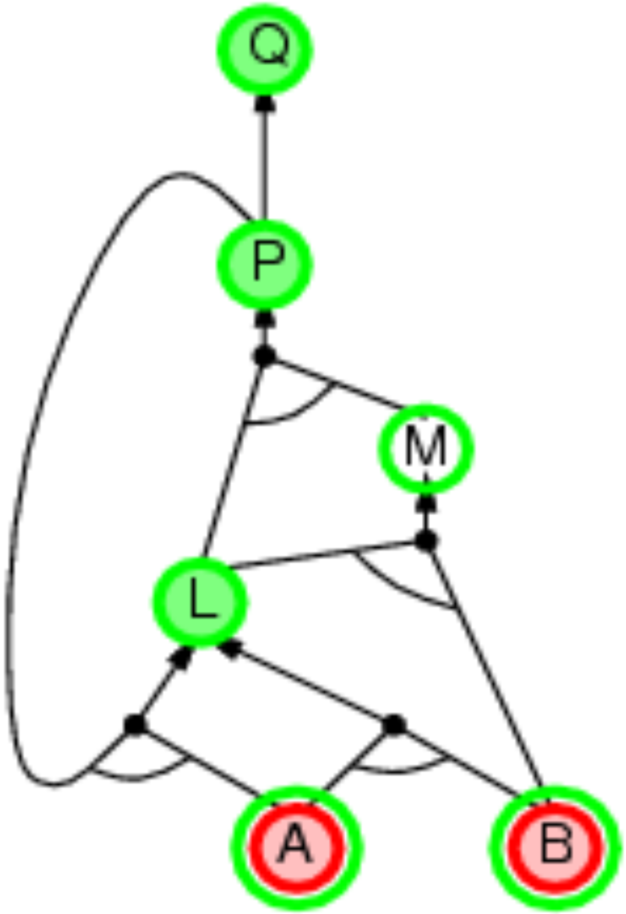
Backward Chaining



Backward Chaining



Backward Chaining



Backward Chaining

- Soundness
  - Yes
- Completeness
  - Yes

Transformation rules

$$(\alpha \wedge \beta) \equiv (\beta \wedge \alpha)$$
$$(\alpha \vee \beta) \equiv (\beta \vee \alpha)$$

Commutativity rules

$$((\alpha \wedge \beta) \wedge \gamma) \equiv (\alpha \wedge (\beta \wedge \gamma))$$
$$((\alpha \vee \beta) \vee \gamma) \equiv (\alpha \vee (\beta \vee \gamma))$$

Associativity rules

$$\neg(\neg\alpha) \equiv \alpha$$

Double-negation elimination

$$(\alpha \Rightarrow \beta) \equiv (\neg\beta \Rightarrow \neg\alpha)$$

Contraposition

$$(\alpha \Rightarrow \beta) \equiv (\neg\alpha \vee \beta)$$
$$(\alpha \Leftrightarrow \beta) \equiv ((\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha))$$

$$\neg(\alpha \wedge \beta) \equiv (\neg\alpha \vee \neg\beta)$$
$$\neg(\alpha \vee \beta) \equiv (\neg\alpha \wedge \neg\beta)$$

de Morgan

$$(\alpha \wedge (\beta \vee \gamma)) \equiv ((\alpha \wedge \beta) \vee (\alpha \wedge \gamma))$$
$$(\alpha \vee (\beta \wedge \gamma)) \equiv ((\alpha \vee \beta) \wedge (\alpha \vee \gamma))$$

Distributivity

Transformation rules (con't)

- $(A \vee (A \wedge B) \equiv A$
- $(A \wedge (A \vee B)) \equiv A$

- $A \wedge 0 \Leftrightarrow 0$
- $A \vee 1 \Leftrightarrow 1$
- $\neg 1 \Leftrightarrow 0$
- $A \vee 0 \Leftrightarrow A$
- $A \wedge 1 \Leftrightarrow A$
- $\neg 0 \Leftrightarrow 1$

- $\neg A \vee A \Leftrightarrow 1$

- $\neg A \wedge A \Leftrightarrow 0$

Transform into CNF

$B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})$

1. Remove $\Leftrightarrow$, replace $\alpha \Leftrightarrow \beta$ by $(\alpha \Rightarrow \beta)\wedge(\beta \Rightarrow \alpha)$.
   $(B_{1,1} \Rightarrow (P_{1,2} \vee P_{2,1})) \wedge ((P_{1,2} \vee P_{2,1}) \Rightarrow B_{1,1})$

2. Remove $\Rightarrow$, replace $\alpha \Rightarrow \beta$ by $\neg\alpha\vee \beta$.
   $(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge (\neg(P_{1,2} \vee P_{2,1}) \vee B_{1,1})$

3. Move negation inward using the de Morgan's rule :
   $(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge ((\neg P_{1,2} \wedge \neg P_{2,1}) \vee B_{1,1})$

4. Applying the "and" distribution rule :
   $(\neg B_{1,1} \vee P_{1,2} \vee P_{2,1}) \wedge (\neg P_{1,2} \vee B_{1,1}) \wedge (\neg P_{2,1} \vee B_{1,1})$

Example

$(A \vee B) \rightarrow (C \rightarrow D)$

1. Remove $\Rightarrow$

   $\neg(A \vee B) \vee (\neg C \vee D)$

2. Move negation inward

   $(\neg A \wedge \neg B) \vee (\neg C \vee D)$

3. Distribution

   $(\neg A \vee \neg C \vee D) \wedge (\neg B \vee \neg C \vee D)$

Exercises

Transform the following expression into CNF.

1. $P \vee (\neg P \wedge Q \wedge R)$
2. $(\neg P \wedge Q) \vee (P \wedge \neg Q)$
3. $\neg(P \Rightarrow Q) \vee (P \vee Q)$
4. $(P \Rightarrow Q) \Rightarrow R$
5. $(P \Rightarrow (Q \Rightarrow R)) \Rightarrow ((P \wedge S) \Rightarrow R)$
6. $(P \wedge (Q \Rightarrow R)) \Rightarrow S$
7. $P \wedge Q \Rightarrow R \wedge S$
8. $((a \vee b) \wedge c) \rightarrow (c \wedge d)$

Priority: $\neg \ \wedge \ \vee \ \rightarrow \ \leftrightarrow$

$$
\begin{aligned}
(\alpha \wedge \beta) &\equiv (\beta \wedge \alpha) \\
(\alpha \vee \beta) &\equiv (\beta \vee \alpha) \\
((\alpha \wedge \beta) \wedge \gamma) &\equiv (\alpha \wedge (\beta \wedge \gamma)) \\
((\alpha \vee \beta) \vee \gamma) &\equiv (\alpha \vee (\beta \vee \gamma)) \\
\neg(\neg\alpha) &\equiv \alpha \\
(\alpha \Rightarrow \beta) &\equiv (\neg\beta \Rightarrow \neg\alpha) \\
(\alpha \Rightarrow \beta) &\equiv (\neg\alpha \vee \beta) \\
(\alpha \Leftrightarrow \beta) &\equiv ((\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)) \\
\neg(\alpha \wedge \beta) &\equiv (\neg\alpha \vee \neg\beta) \\
\neg(\alpha \vee \beta) &\equiv (\neg\alpha \wedge \neg\beta) \\
(\alpha \wedge (\beta \vee \gamma)) &\equiv ((\alpha \wedge \beta) \vee (\alpha \wedge \gamma)) \\
(\alpha \vee (\beta \wedge \gamma)) &\equiv ((\alpha \vee \beta) \wedge (\alpha \vee \gamma))
\end{aligned}
$$

1. $P \vee (\neg P \wedge Q \wedge R)$
2. $(\neg P \wedge Q) \vee (P \wedge \neg Q)$
3. $\neg(P \Rightarrow Q) \vee (P \vee Q)$
4. $(P \Rightarrow Q) \Rightarrow R$
5. $(P \Rightarrow (Q \Rightarrow R)) \Rightarrow ((P \wedge S) \Rightarrow R)$
6. $(P \wedge (Q \Rightarrow R)) \Rightarrow S$
7. $P \wedge Q \Rightarrow R \wedge S$

Definitive and Horn Clauses

- **Definitive Clause**
  - Disjunction of literals of which ***exactly one*** *is positive*.
  - Example: clause (¬L1,1 ∨ ¬Breeze ∨ B1,1) is a definite clause, whereas (¬B1,1 ∨ P1,2 ∨ P2,1) is not.
- **Horn Clause**
  - Disjunction of literals of which *at most one is positive*.
  - All definite clauses are Horn clauses, as are clauses with no positive literals; these are called **goal clauses**.

Reading & Suggested Exercises

- Chapter 7
- Exercises 7.4, 7.5, 7.10, 7.19, 7.20