COMP9414 Week 3 - Knowledge Representation and Constraints

Author: Junji Duan

Last Update: 2025/10/05

1 W3A: Constraint Satisfaction

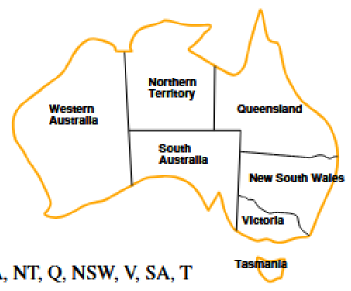
1.1 Constraint Satisfaction Problems (CSPs)

Def: Constraint Satisfaction Problems are defined by a set of variables (变量) X_i, each with a domain D_i (域) of possible values, and a set of constraints (约束) C.

Aim is to find an **assignment(值)** to each the variables X_i (a value from the domain D_i) such that all of the constraints C are satisfied.

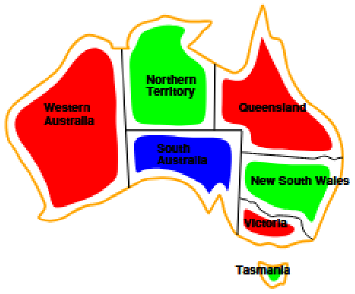
即: 要找到一个「赋值方案」(assignment)，使得问题的规则没有被违反.

1.1.1 CSP Example: Map Colouring



Variables: WA, NT, Q, NSW, V, SA, T
Domains: $D_i = \{\text{red, green, blue}\}$
Constraints: Adjacent regions have different colours (WA $\neq$ NT, etc.)

Solution is an assignment that satisfies all the constraints



{WA=red, NT=green, Q=red, NSW=green, V=red, SA=blue, T=green}

1.1.2 Real World CSPs

- Assignment problems (e.g. who teaches what class)
- Timetabling problems (e.g. which class is offered when and where?)
- Hardware configuration (e.g. minimize space for circuit layout)
- Transport scheduling (e.g. courier delivery, vehicle routing)
- Factory scheduling (optimize assignment of jobs to machines)
- Gate assignment (assign gates to aircraft to minimize transit time)

Many real world CSPs are also optimization problems

1.1.3 Varieties of CSPs

Discrete variables

- Finite domains; size $d \Rightarrow O(d^n)$ complete assignments
  - e.g. Boolean CSPs, including Boolean satisfiability (NP-complete)
- Infinite domains (integers, strings, etc.)
  - Job shop scheduling, variables are start/end days for each job
  - Need a constraint language, e.g. $\text{StartJob}_1 + 5 \leq \text{StartJob}_3$
  - Linear constraints solvable, nonlinear undecidable

Continuous variables

- e.g. start/end times for Hubble Telescope observations
- Linear constraints solvable in polynomial time by LP methods

1.1.4 Types of Constraint

- Unary constraints involve a single variable
  - $\mathbf{M \neq 0}$
- Binary constraints involve pairs of variables
  - SA $\neq$ WA
- Higher-order constraints involve 3 or more variables
  - $Y = D + E$ or $Y = D + E - 10$
- Inequality constraints on continuous variables
  - $\text{EndJob}_1 + 5 \leq \text{StartJob}_3$
- Soft constraints (Preferences)
  - 11am lecture is better than 8am lecture!

1.2 Standard search methods

1.2.1 Path Search (last week) vs Constraint Satisfaction

Important difference between path search problems and CSPs

- Constraint Satisfaction Problems (e.g. N-Queens)
  - Difficult part is knowing the final state
  - How to get there is easy
- Path Search Problems (e.g. Rubik' s Cube)
  - Knowing the final state is easy
  - Difficult part is how to get there

1.2.2 Backtracking search and heuristics

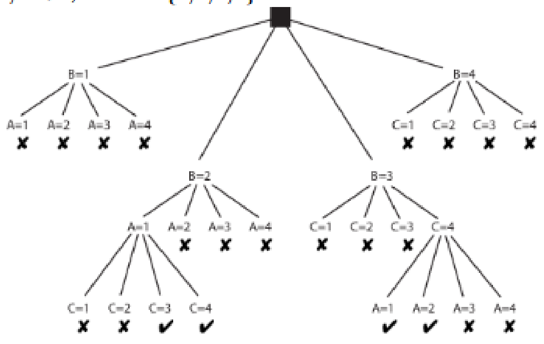
CSPs can be solved by assigning values to variables one by one, in
different combinations. Whenever a constraint is violated, go back to the most recently assigned variable and assign it a new value.

Can be implemented using Depth-First Search on a special kind of state space, where states are defined by the values assigned so far.

- Initial state: Empty assignment
- Successor function: Assign a value to an unassigned variable that
does not conflict with previously assigned values of other variables
- Goal state: All variables are assigned a value and all constraints are satisfied

Backtracking search has problem:

$A < B, B < C$, Domains $\{1,2,3,4\}$



Repeatedly compares A = 4 to all values of B

Backtracking Search Space has very specific properties

- If there are n variables, every solution is at depth exactly n
- Variable assignments are commutative
[WA = red then NT = green] same as [NT = green then WA = red]

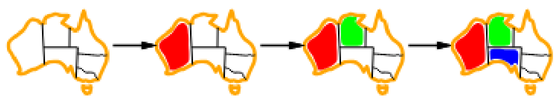
Backtracking search can solve N-Queens for N 25

1.2.3 Improvements to Backtracking Search

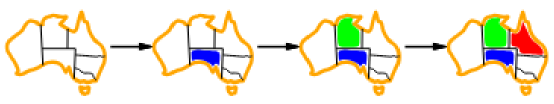
1. Which variable should be assigned next?

- 变量选择（Variable ordering）
  - MRV (Minimum Remaining Values)：选剩余合法值最少的变量
  - Degree heuristic：当 MRV 有平手时，用它打破平局（选与其他变量约束最多的变量）

- Minimum Remaining Values (MRV)
  - Choose variable with the fewest legal values
  - Choose randomly between them if more than one
  - Apply constraints to eliminate values of other variables



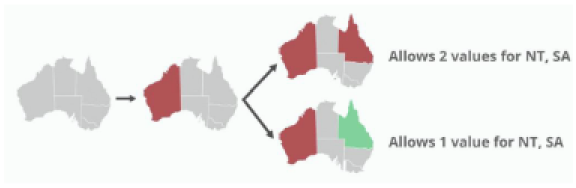
- Tie-breaker among MRV variables
  - Choose variable with most constraints to remaining variables



2. In what order should its values be tried?

- 值选择（Value ordering）
  - LCV (Least Constraining Value)：优先选对其他变量影响最小的值（留下最多可能性）

- Given a variable, choose the least constraining value
  - One that rules out the fewest values in the remaining variables



More generally, 3 allowed values would be better than 2, etc.

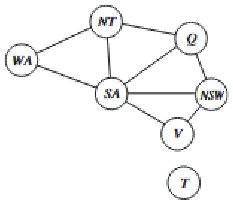
Combining these heuristics makes 1000-Queens feasible

3. Can inevitable failure be detected early?

- 提前失败检测（Early failure detection）
  - 通过约束传播 (Constraint Propagation)
    - ∗ MRV 在某种程度上也能帮助早发现失败（某变量域变成空 → 立即剪枝）
    - ∗ 更典型的方法是 Forward Checking / Arc Consistency (AC-3) 等，用约束图 (Constraint Graph) 来提前剪枝。

Binary CSP: Each constraint relates at most two variables

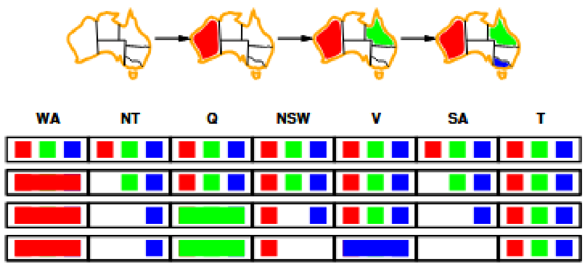
Constraint Graph: Nodes are variables, arcs show constraints



General-purpose CSP algorithms use the graph structure to speed up search, e.g. Tasmania is an independent subproblem!

1.2.4 Forward checking

Idea: Keep track of remaining legal values for unassigned variables Terminate search when any variable has no legal values.



核心思想：

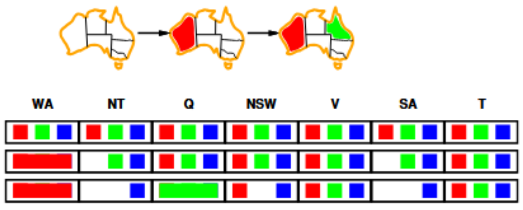
- 在搜索时，不只是给当前变量赋值，还要检查它对未赋值变量的影响。
- 具体做法：记录每个未赋值变量还剩多少“合法值”。
- 如果发现某个变量剩余合法值为 0，就立刻停止（剪枝）。

直觉：提前发现“死胡同”，避免走到底才回溯。

例子：假如 NSW 只能用“红”，那么 SA（相邻州）就不能用“红”；如果 SA 的域正好只有 {红}，那么域就变成空集 → 提前终止。

Constraint Propagation (对 Forward Checking 的扩展)：Forward checking only propagates from assigned to unassigned variables, but does not detect all failures. Constraint propagation repeatedly enforces constraints locally, achieving stronger pruning.

Forward checking propagates information from assigned to unassigned variables, but doesn't provide early detection for all failures



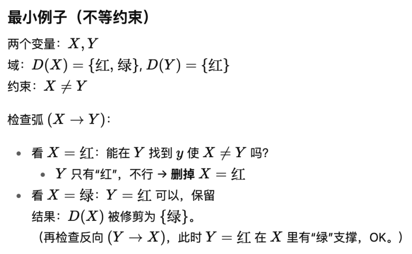
NT and SA cannot both be blue!

Constraint propagation repeatedly enforces constraints locally

Arc consistency: Simplest form of constraint propagation is arc consistency.

对任何一个变量 X 的每个候选值 x ，在与它相连的另一个变量 Y 的候选集中，都能找到至少一个 " 搭得上 " 的值 y ，使二者不违背约束。

找不到 " 搭得上 " 的，就把 x 从 X 的候选里删掉。这就叫让 $(X \to Y)$ 这条 " 弧 " 一致（arc - consistent）。



最小例子（不等约束）

两个变量：X, Y

域：$D(X) = \{红, 绿\}, D(Y) = \{红\}$

约束：$X \neq Y$

检查弧 $(X \to Y)$：

- 看 $X =$ 红：能在 Y 找到 y 使 $X \neq Y$ 吗?
  - Y 只有“红”，不行 → **删掉** $X =$ 红
- 看 $X =$ 绿：$Y =$ 红 可以，保留

  结果：$D(X)$ 被修剪为 $\{绿\}$。

  （再检查反向 $(Y \to X)$，此时 $Y =$ 红 在 X 里有“绿”支撑，OK。）

即:
1. 把所有弧（方向边，如 $X \to Y$ ）放进一个队列。
2. 取出一条弧，按上面的 " 有支撑就留、没支撑就删 " 的规则修剪域。

3. 如果修剪了 X 的域，就把 " 指向 X'' 的其他弧重新丢回队列（因为 X 变了，会影响它的别的邻居）。
4. 队列空了就停；若任何域被列空 ⇒ 当前前缀下无解，应回溯。

Arc consistency detects failure earlier than forward checking.
For some problems, it can speed up the search enormously.
For others, it may slow the search due to computational overheads.

1.2.5 Domain splitting and arc consistency

States are whole CSPs (not partial assignments)

- Make CSP domain consistent and arc consistent
  - Domain consistent = all unary constraints are satisfied
- To solve CSP using Depth-First Search
  - Choose a variable X with more than one value in domain
  - Split the domain of X into two subsets
  - This gives two smaller CSPs
  - Make each CSP arc consistent (if possible)
  - Solve each resulting CSP (or backtrack if unsolvable)

即：不是像普通回溯那样“给变量逐个赋一个具体值”，而是把整个 CSP（变量 → 域）当成一个状态来搜索；每一步通过切分某个域（domain splitting）、再用弧一致（arc consistency）做传播来缩小问题，直到解出来或证明这条路不通。

1.2.6 Constraint Optimization Problems

States are whole CSPs (not partial assignments) with costs

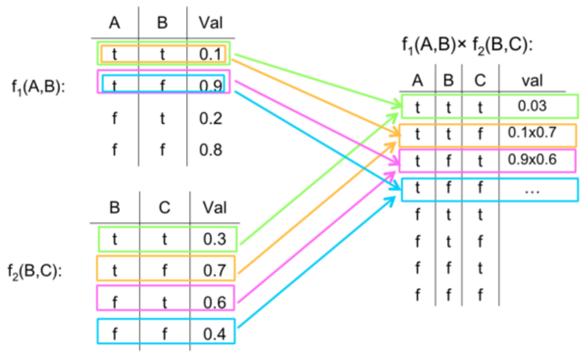
- Make CSP domain consistent and arc consistent
- Add CSP to priority queue
- To solve CSP using A* Search
  - Remove CSP with minimal f from priority queue
  - Choose a variable X with more than one value in domain
  - Split the domain of X into two subsets
  - This gives two smaller CSPs
  - Make each CSP arc consistent (if possible) - add to priority queue
- cost(CSP)   sum of costs to violate soft constraints

即: 把 “整个 CSP（变量 → 域)” 当成一个节点/状态来搜索，但现在每个状态还有 “代价”（因为有软约束），用 A* 在这些 “CSP 状态” 之间找总成本最小的解。

1.2.7 Variable elimination

把变量 X " 从问题里拿走 "：
把所有涉及 X 的约束合成一个 " 大表 " R_1 → 把 X 这一列删掉（即对 X 求存在消去）得到 R_2 → 用 R_2 替换掉原来所有含 X 的约束。这样不再显式提到 X ，但它对其它变量的影响已被折reference到 R_2 里了。

- If there is only one variable, return the intersection of its (unary) constraints
- Otherwise
  - Select a variable X
  - Join the constraints in which X appears, forming constraint R_1
  - Project R_1 onto its variables other than X, forming R_2
  - Replace all of the constraints in which X appears by R_2
  - Recursively solve the simplified problem, forming R_3
  - Return R_1 joined with R_3



1.3 Local search

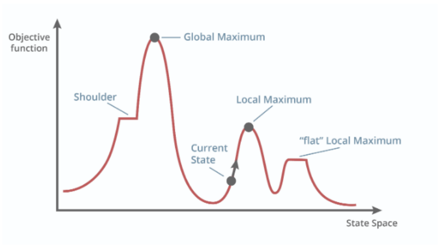
Starts from a complete solution and iteratively improves it via small neighborhood moves; doesn’t guarantee global optimality but often finds good solutions fast in huge spaces.

从一个完整但可能很差的解出发，通过对解做小改动（邻域移动）不断改进目标值；不保证全局最优，但常在巨大空间里快找到“够好”的解。

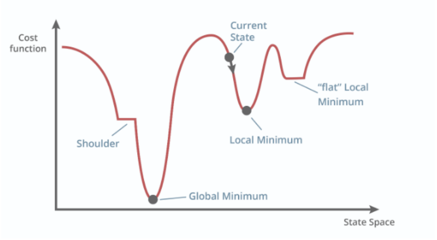
1.3.1 Hill climbing

Greedily moves to a better neighbor each step—simple and fast, but prone to local optima and plateaus.

每步只接受能让目标更好的邻居（贪心上升），实现简单、速度快，但容易卡在局部最优/台地。



Sometimes, have to go sideways or even backwards to make progress towards a globally optimal solution



When minimizing violated constraints, it makes sense to think of starting at the top of a ridge and climbing down into the valleys

1.3.2 Simulated annealing

Occasionally accepts worse moves (probabilistically) to escape local traps, then becomes greedier as temperature cools; more robust but needs a good cooling schedule.

在高温时允许以一定概率接受更差解以跳出局部最优，随降温逐渐转向贪心；质量更稳健但需要调好温度与降温策略。

---

2 W3B: Knowledge Representation

2.1 Knowledge Representation and Logic

Def: Knowledge representation (KR) aims to model information in a structured manner to formally represent it as knowledge in knowledge-based systems whereas knowledge representation and reasoning (KRR) also aims to understand, reason, and interpret knowledge.

Def: Syntax(句法/语法) Describes the legal sentences in a knowledge representation language.

句法只管“形状”，不管“意思”。

- 例（命题逻辑）：如果 P 是句子、Q 是句子，那么 $P \wedge Q$ $\neg P$ $P \to Q$ 也是合法句子。
- Syntax = form rules (what you’ re allowed to write).

Def: Semantics(语义) Refers to the meaning of sentences. Relates sentences (and sentence fragments) to aspects of the world the sentence is about.

- 例：在一个世界里设＂Raining＂为真，＂$\neg$ Raining＂就为假；＂$P \to Q$＂在所有＂P 为真却 Q 为假＂的世界里为假，其余为真。
- Semantics = meaning (how sentences map to the world/truth).

Def: Propositions(命题) are entities (facts or non-facts) that can be true or false.

命题是一个可以判真假的陈述内容（fact or non-fact 都行，只要“可真可假”）

- 例：今天下雨了”是命题（可真可假）；“请把窗关上”不是命题（命令句没真值）。在命题逻辑中，用符号 P, Q 指代命题；它们的真值由语义/模型给出。
- Proposition = truth-evaluable content (something that’ s either true or false).

2.2 Logical Arguments

- An argument relates a set of premises to a conclusion.
  - invalid if the conclusion can be false when the premises are all true
    - 在所有前提都为真时，结论仍有可能为假
  - valid if the conclusion necessarily follows from the premises
    - 只要前提全真，结论必真

2.3 Propositional Logic

Use letters to stand for "basic" propositions; combine them into more
complex sentences using operators for not, and, or, implies, iff

2.3.1 Syntax

$\neg$	negation	$\neg P$	"not P"
$\wedge$	conjunction	$P \wedge Q$	"P and Q"
$\vee$	disjunction	$P \vee Q$	"P or Q"
$\rightarrow$	implication	$P \rightarrow Q$	"If P then Q"
$\leftrightarrow$	bi-implication	$P \leftrightarrow Q$	"P if and only if Q"

The semantics of the connectives can be given by truth tables

P	Q	$\neg P$	$P \wedge Q$	$P \vee Q$	$P \rightarrow Q$	$P \leftrightarrow Q$
True	True	False	True	True	True	True
True	False	False	False	True	False	False
False	True	True	False	True	True	False
False	False	True	False	False	True	True

2.3.2 Semantics

- A sentence is valid if it is True under all possible assignments of True/False to its variables (e.g. $P \vee \neg P$)
- A tautology is a valid sentence.
- Two sentences are equivalent if they have the same truth table, e.g. $P \wedge Q$ and $Q \wedge P$
  - So P is equivalent to Q if and only if $P \leftrightarrow Q$ is valid
- A sentence is satisfiable if there is some assignment of True/False to its variables for which the sentence is True
- A sentence is unsatisfiable if it is not satisfiable (e.g. $P \wedge \neg P$)
  - Sentence is False for all assignments of True/False to its variables
  - So P is a tautology if and only if $\neg P$ is unsatisfiable

Commutativity:	$P \wedge Q \leftrightarrow Q \wedge P$	$P \vee Q \leftrightarrow Q \vee P$
Associativity:	$P \wedge (Q \wedge R) \leftrightarrow (P \wedge Q) \wedge R$	$P \vee (Q \vee R) \leftrightarrow (P \vee Q) \vee R$
Distributivity:	$P \wedge (Q \vee R) \leftrightarrow (P \wedge Q) \vee (P \wedge R)$	$P \vee (Q \wedge R) \leftrightarrow (P \vee Q) \wedge (P \vee R)$
Implication:	$(P \rightarrow Q) \leftrightarrow (\neg P \vee Q)$	
Idempotent:	$P \wedge P \leftrightarrow P$	$P \vee P \leftrightarrow P$
Double negation:	$\neg\neg P \leftrightarrow P$	
Contradiction:	$P \wedge \neg P \leftrightarrow$ FALSE	
Excluded middle:		$P \vee \neg P \leftrightarrow$ TRUE
De Morgan:	$\neg(P \wedge Q) \leftrightarrow (\neg P \vee \neg Q)$	$\neg(P \vee Q) \leftrightarrow (\neg P \wedge \neg Q)$

2.3.3 Entailment(蕴含)

- S entails $P(S \models P)$ if whenever all formulae in S are True, P is True
  - Semantic definition - concerns truth (not proof)
  - 只要 S 中所有公式为真, P 也必真.
- Compute whether $S \models P$ by calculating a truth table for S and P
  - Syntactic notion - concerns computation/proof
  - Not always this easy to compute (how inefficient is this?)
- A tautology is a special case of entailment where S is the empty set
  - All rows of the truth table are True

Write $P \models Q$ for $\{P\} \models Q$

$P \wedge Q \models P$	$P \wedge Q \models Q$
$P \models P \vee Q$	$Q \models P \vee Q$
$P \models \neg\neg P$	$\neg\neg P \models P$
$\{P, P \rightarrow Q\} \models Q$	
If $P \models Q$ then $\models P \rightarrow Q$	

2.3.4 Tableau Method

Alpha Rules:

$$\begin{array}{c} A \wedge B \\ \hline A \\ B \end{array} \qquad \begin{array}{c} \neg(A \vee B) \\ \hline \neg A \\ \neg B \end{array} \qquad \begin{array}{c} \neg(A \rightarrow B) \\ \hline A \\ \neg B \end{array}$$

¬¬-Elimination:

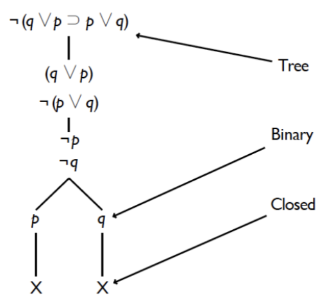
$$\begin{array}{c} \neg\neg A \\ \hline A \end{array}$$

Beta Rules:

$$\begin{array}{c|c} \multicolumn{2}{c}{A \vee B} \\ \hline A & B \end{array} \qquad \begin{array}{c|c} \multicolumn{2}{c}{A \rightarrow B} \\ \hline \neg A & B \end{array} \qquad \begin{array}{c|c} \multicolumn{2}{c}{\neg(A \wedge B)} \\ \hline \neg A & \neg B \end{array}$$

Branch Closure:

$$\begin{array}{c} A \\ \neg A \\ \hline \times \end{array}$$



3 W3B Part2: Automated Reasoning

Automated Reasoning（自动推理）：用形式化规则让计算机“做证明/推理”，像数学家一样但可程序化。

3.1 Soundness, completeness, decidability

- A proof system is sound(健全性) if (intuitively) it preserves truth
  - 证明系统不会“证明出假东西”；直觉：保真——“真前提”推出的“结论”在所有模型里都真
  - Whenever $S \vdash P$, if every formula in S is True, P is also True
  - Whenever $S \vdash P, S \models P$
  - If you start with true assumptions, any conclusions must be true
- A proof system is complete(完备性) if it is capable of proving all consequences of any set of premises (including infinite sets)
  - 证明系统不会漏掉任何语义上成立的结论; 直觉：不漏——凡“真的”，都能被系统证明出来
  - Whenever P is entailed by S, there is a proof of P from S
  - Whenever $S \models P, S \vdash P$
- A proof system is decidable(可判定性) if there is a mechanical procedure (computer program) which when asked whether $S \vdash P$, can always answer 'yes' - or 'no' - correctly
  - 存在一个机械过程/程序，对任意输入(S,P)，判断“是否 S P”并必定在有限时间内给出正确的“是/否”；直觉：总会停并答对的判定算法

3.2 Resolution and Refutation 归结与反证

Another type of proof system based on refutation.

Better suited to computer implementation than systems of axioms and rules (can give correct 'no' answers).

Decidable in the case of Propositional Logic.

Generalizes to First-Order Logic (e.g. Planning) item Needs all formulae to be converted to clausal form.

For Propositional Logic, resolution refutation is sound, complete and decidable.

3.2.1 Normal Forms

A literal ℓ is a propositional variable or the negation of a propositional variable (P or $\neg P$)

A clause is a disjunction of literals $\ell_1 \vee \cdots \vee \ell_n$

Conjunctive Normal Form (CNF) - a conjunction of clauses, e.g. $(P \vee Q \vee \neg R) \wedge (\neg S \vee \neg R)$ - or just one clause, e.g. $P \vee Q$

Disjunctive Normal Form (DNF) - a disjunction of conjunctions of literals, e.g. $(P \wedge Q \wedge \neg R) \vee (\neg S \wedge \neg R) -$ or just one conjunction, e.g. $P \wedge Q$

Every Propositional Logic formula can be converted to CNF and DNF

Every Propositional Logic formula is equivalent to its CNF and DNF

3.2.2 Conversion to Conjunctive Normal Form

Eliminate $\leftrightarrow$ rewriting $P \leftrightarrow Q$ as $(P \rightarrow Q) \wedge (Q \rightarrow P)$

Eliminate $\rightarrow$ rewriting $P \rightarrow Q$ as $\neg P \vee Q$

Use De Morgan's laws to push $\neg$ inwards (repeatedly)

- Rewrite $\neg(P \wedge Q)$ as $\neg P \vee \neg Q$
- Rewrite $\neg(P \vee Q)$ as $\neg P \wedge \neg Q$

Eliminate double negations: rewrite $\neg\neg P$ as P

Use the distributive laws to get CNF [or DNF] - if necessary

- Rewrite $(P \wedge Q) \vee R$ as $(P \vee R) \wedge (Q \vee R)$ [for CNF]
- Rewrite $(P \vee Q) \wedge R$ as $(P \wedge R) \vee (Q \wedge R)$ [for DNF]

3.2.3 Clausal Form

- $\neg(P \rightarrow (Q \wedge R))$
- $\neg(\neg P \vee (Q \wedge R))$
- $\neg\neg P \wedge \neg(Q \wedge R)$
- $\neg\neg P \wedge (\neg Q \vee \neg R)$
- $P \wedge (\neg Q \vee \neg R)$
- Clausal Form: $\{P, \neg Q \vee \neg R\}$

Clausal Form = set of clauses in the CNF

- 即: 把原式等价变形后得到 CNF$P \wedge (\neg Q \vee \neg R)$ ，对应的子句集合就是 $\{P, \neg Q \vee \neg R\}$ 。这是进行 resolution 时所需的输入格式。

参考文献

[1] UNSW COMP9414 Lecture slides –Prof. W.Wobcke

[2] Other online resources (eg.graph)