Chapter 8

Inference and Resolution for Problem Solving

Chapter 8 Contents (1)

Normal Forms

- **A wff is in Conjunctive Normal Form (CNF) if it is a conjunction of disjunctions:**
  - $A_1 \land A_2 \land A_3 \land \dots \land A_n$
  - where each clause, $A_{i,}$ is of the form:
  - $B_1 \lor B_2 \lor B_3 \lor \dots \lor B_n$
  - The B's are literals.
- **E.g.:** $A \land (B \lor C) \land (\neg A \lor \neg B \lor \neg C \lor D)$
- **Similarly, a wff is in Disjunctive Normal Form (DNF) if it is a disjunction of conjunctions.**
- **E.g.:** $A \lor (B \land C) \lor (\neg A \land \neg B \land \neg C \land D)$

數位電路："Product of Sums" v.s. "Sum of Products"

Converting to CNF

- **Any** wff can be converted to CNF by using the following equivalences:

(1) $A \leftrightarrow B \equiv (A \rightarrow B) \land (B \rightarrow A)$

(2) $A \rightarrow B \equiv \neg A \lor B$

(3) $\neg(A \land B) \equiv \neg A \lor \neg B$

(4) $\neg(A \lor B) \equiv \neg A \land \neg B$

(5) $\neg\neg A \equiv A$

(6) $A \lor (B \land C) \equiv (A \lor B) \land (A \lor C)$

- Importantly, this can be converted into an algorithm – this will be useful when when we come to automating resolution.

Clauses

- Having converted a wff to CNF, it is usual to write it as a set of clauses.
- E.g.:

  $(A \rightarrow B) \rightarrow C$

- In CNF is:

  $(A \lor C) \land (\neg B \lor C)$

- In clause form, we write:

  $\{(A, C), (\neg B, C)\}$

大括弧內是 **AND**
小括弧內是 **OR**

由各別的 CNF
變成規則庫中的 clause (rule)
規則庫中的 clauses 彼此都是 Λ
如果 **CNF** 中有 **Λ (**如左例**)**
一條 **CNF** 可以被拆成多條 **clauses**

The Resolution Rule

- The resolution rule is written as follows:

$$\frac{A \lor B \qquad \neg B \lor C}{A \lor C}$$

規則庫寫作 Clause form
好處是可做破解(Resolution)

- This tells us that if we have two clauses that have a literal and its negation, we can combine them by removing that literal.
- E.g.: if we have {(A, C), (¬A, D)}
- We would apply resolution to get {C, D}

Resolution Refutation

A→B　　A ^ B →C

- Let us resolve: {(¬A, B), (¬A, ¬B, C), A, ¬C}
- We begin by resolving the first clause with the second clause, thus eliminating B and ¬B:

  {(¬A, C), A, ¬ C}

  {C, ¬C}

破解(Resolution)之最終目地是驗證
"某論點如果是錯的" 規則庫就 "不成立"

- Now we can resolve both remaining literals, which gives falsum:

  ⊥

- If we reach falsum, we have proved that our initial set of clauses were inconsistent. …. Think about **{C, ¬C} ?**
- This is written:

  {(¬A, B), (¬A, ¬B, C), A, ¬C} ╞ ⊥

何而爲規則庫 "不成立" ?

{ P, (¬P)} ╞ ⊥

Ture → False

{(¬True V False)}╞ ⊥

(1) 這是述語邏輯的 clause form
(2) 得證敘述 C 成立 (¬C 不成立)

Proof by Refutation

- If we want to prove that a logical argument is valid, we negate its conclusion, convert it to clause form, and then try to derive falsum using resolution.
- If we derive falsum, then our clauses were inconsistent, meaning the original argument was valid, since we negated its conclusion.

加入 (¬EXP) 或 (EXP → false) 皆可得證 EXP ≡ True

(EXP → false) ^ RB ╞ ⊥ 亦即 **{(¬EXP V false) ^ RB}** ╞ ⊥

Proof by Refutation - Example

- Our argument is:

  $(A \wedge \neg B) \rightarrow C$

  $A \wedge \neg B$

  $\therefore C$

把希望驗證的敘述 "之相反式" 加到規則之中與現有的規則合在一起推論其成立與否

"球技好" 而且 "沒有不良嗜好" 則將是 "未來之星"

- Negate the conclusion and convert to clauses:

  $\{(\neg A, B, C), A, \neg B, \neg C\}$

- Now resolve:

  $\{(B, C), \neg B, \neg C\}$

  $\{C, \neg C\}$

  $\perp$

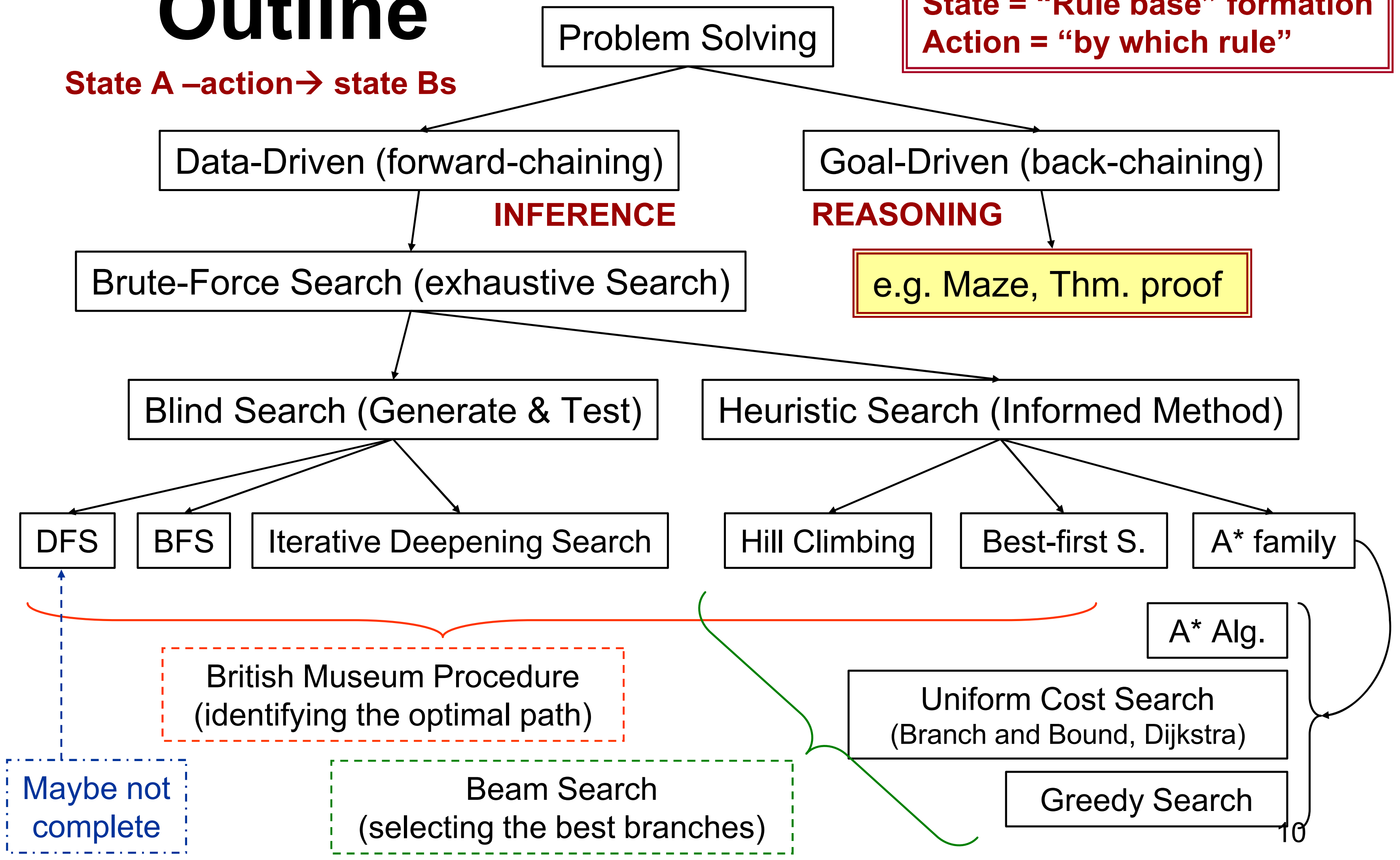
- We have reached falsum, so our original argument was valid.

Recall:

Outline

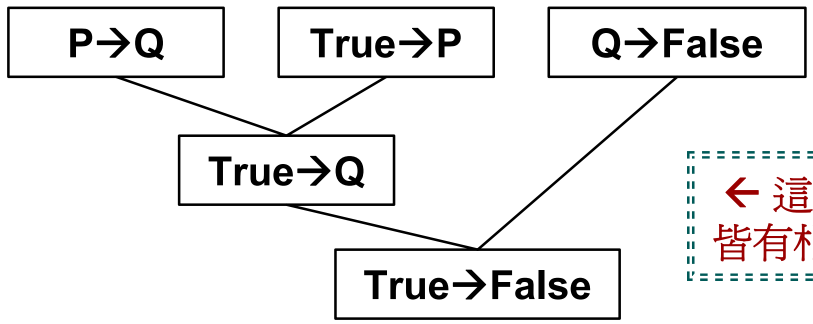
State A –action→ state Bs

State = "Rule base" formation
Action = "by which rule"

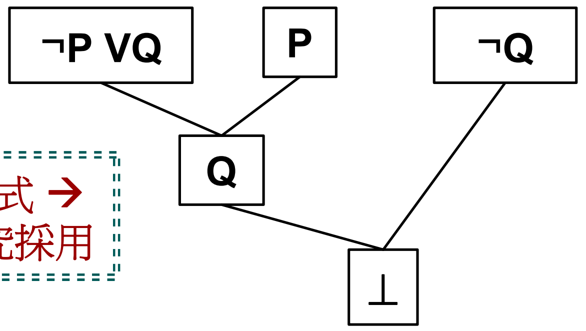

Problem Solving
Data-Driven (forward-chaining)
Goal-Driven (back-chaining)
INFERENCE
REASONING
Brute-Force Search (exhaustive Search)
e.g. Maze, Thm. proof
Blind Search (Generate & Test)
Heuristic Search (Informed Method)
DFS
BFS
Iterative Deepening Search
Hill Climbing
Best-first S.
A* family
A* Alg.
Uniform Cost Search
(Branch and Bound, Dijkstra)
Greedy Search
British Museum Procedure
(identifying the optimal path)
Beam Search
(selecting the best branches)
Maybe not
complete

Refutation Proofs in Tree Form

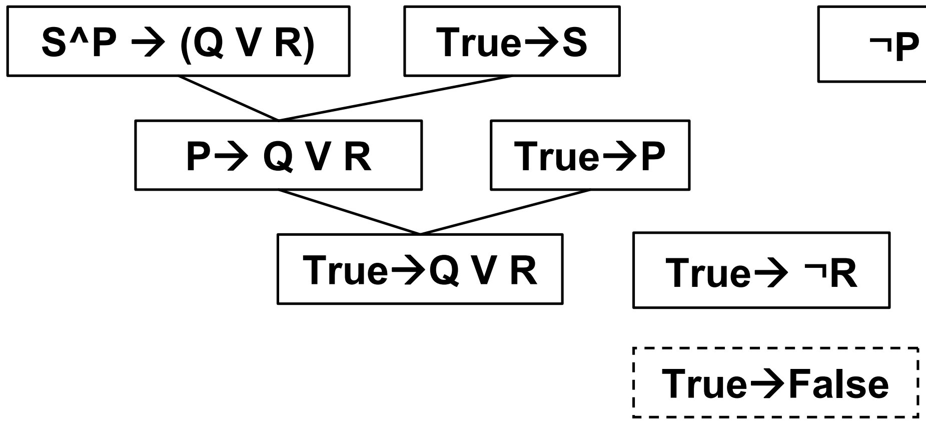
P, P→Q   Should Q be true?

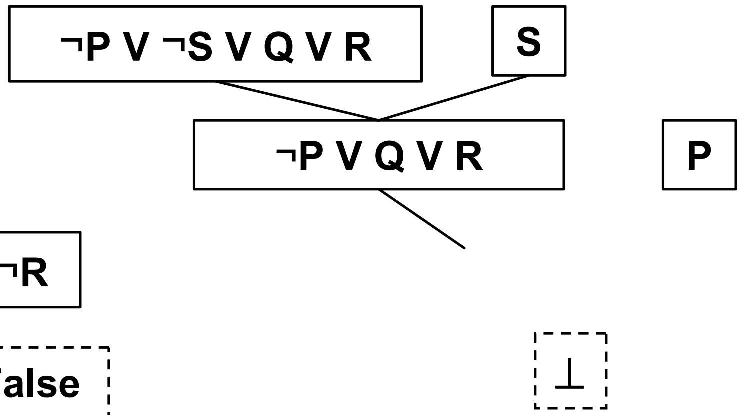


← 這兩種形式 →
皆有相關研究採用



S, P, ¬R, (S^P→Q V R) Should Q be true?

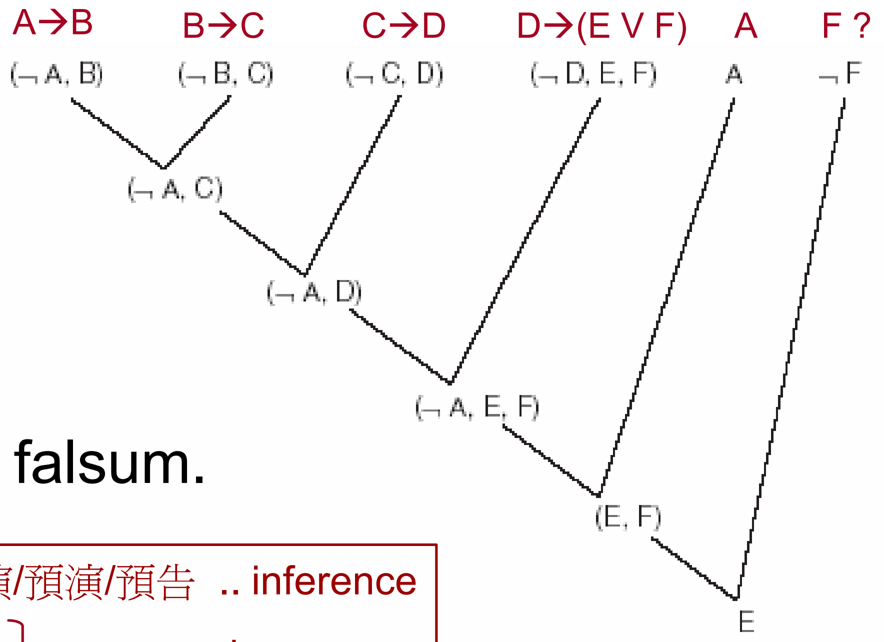




Example1: Refutation Proofs

- It is often sensible to represent a refutation proof in tree form:

想要求證 F 成立嗎？
結論是
要再加上 ¬E 才成立



- In this case, the proof has failed, as we are left with E instead of falsum.

(1) 依目前規則做無目標性的推演/預演/預告 .. inference
(2) 試求某個結論成立或不成立 }
若不成立能否知道所缺條件? } … reasoning

Skip

Example2: Graph Coloring

- Resolution refutation can be used to determine if a solution exists for a particular combinatorial problem.
- For example, for graph coloring, we represent the assignment of colors to the nodes and the constraints regarding edges as propositions, and attempt to prove that the complete set of clauses is consistent.
- This does not tell us how to color the graph, simply that it is possible.

Skip

Example: Graph Coloring (2)

- Available colors are r (red), g (green) and b (blue).
- A_r means that node A has been coloured red.
- Each node must have exactly one color:

  $A_r \lor A_g \lor A_b$

  $\neg A_r \lor \neg A_g$ $(\equiv A_r \rightarrow \neg A_g)$

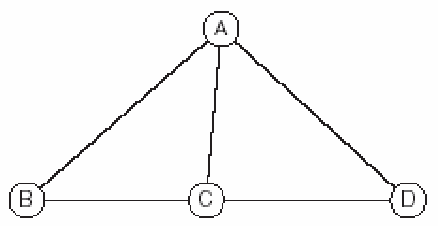
  $\neg A_g \lor \neg A_b$

  $\neg A_b \lor \neg A_r$
- If (A,B) is an edge, then:

  $\neg A_r \lor \neg B_r$

  $\neg A_b \lor \neg B_b$

  $\neg A_g \lor \neg B_g$
- Now we construct the complete set of clauses for our graph, and try to see if they are consistent, using resolution.



Normal Forms in Predicate Calculus

- A FOPC expression can be put in prenex normal form by converting it to CNF, with the quantifiers moved to the front.
- For example, the wff:

如果世上每個樂觀的人都長壽
則必存在某個樂觀且長壽的人

$$(\forall x)(A(x) \rightarrow B(x)) \rightarrow (\exists y)(A(y) \land B(y))$$

- Converts to prenex normal form as:

$$(\exists x)(\exists y)(((A(x) \lor A(y)) \land (\neg B(x) \lor A(y)) \land (A(x) \lor B(y)) \land (\neg B(x) \lor B(y))))$$

由此開始講 Predicate Calculus 之中變數的 (∀x), (∃x) 概念及處理方式

Unification (1)

- To resolve clauses we often need to make substitutions. For example:

  {(P(w,x)), (¬P(a,b))}

  e.g. 全城居民都相親相愛

- To resolve, we need to substitute a for w, and b for x, giving:

  {(P(a,b)), (¬P(a,b))}

- Now these resolve to give falsum. （可完全消去）

對於 (∀x) … 我們略去，因爲規則庫都是通則
其中變數 x 須在 Resolution 過程之中進行代換。

Unification (2)

- A substitution that enables us to resolve a set of clauses is called a unifier.
- We write the unifier we used above as:

  {a/w, b/x}
- A unifier (u) is a most general unifier (mgu) if any other unifier can be formed by the composition of u with some other unifier. (規則庫中大家都用得上的替代)
- An algorithm can be generated which obtains a most general unifier for any set of clauses.

Skolemization

那對於 (∃x) … 我們要如何處理？

- Before resolution can be applied, ∃ must be removed, using skolemization.
- Variables that are existentially quantified are replaced by a constant:

  ∃(x) P(x)   e.g. ∃(y) GoodMan(y)

- is converted to:

  P(c)   e.g. GoodMan(c)

- c must not already exist in the expression.

存在某人去刺殺暴君，被刺殺則死亡 **{ ∃(x) kill(x, Caesar),   ∀(y,z) kill(y,z) → die(z) }**

Qu: 暴君死亡了嗎？   或者 **{ ∃(x) kill(x, Caesar),   ∀(z) ∃(y) kill(y,z) → die(z) }**

Ans: 死了(依代換過程,是那個 “刺殺暴君的人”殺的）**{ kill(killer(Caesar), Caesar), …}**

Skolem functions

- If the existentially quantified variable is within the scope of a universally quantified variable, it must be replaced by a skolem function, which is a function of the universally quantified variable.
- Sounds complicated, but is actually simple:

  $(\forall x)(\exists y)(P(x,y))$

- Is Skolemized to give:

  $(\forall x)(P(x,f(x))$ e.g. $(\forall x)(Love(x,lover(x))$

$(\forall x)$ … 略去，規則庫預設都是通則
$(\exists y)$ … 如上頁以 Skolem function 取代

- After skolemization, $\forall$ is dropped, and the expression converted to clauses in the usual manner.

Skolem function 視爲特殊的 “常數”，
只可代換別人，不可被別人代換，並不違背 FOPC

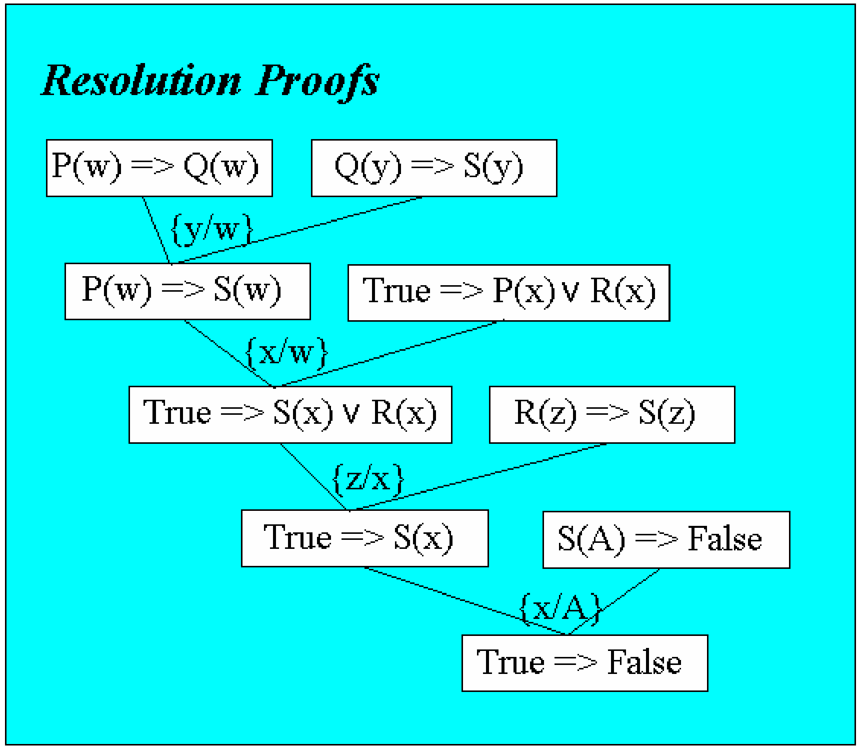
The Resolution Algorithm

1 First, negate the conclusion and add it to the list of assumptions.
2 Now convert the assumptions into Prenex Normal Form
3 Next, skolemize the resulting expression
4 Now convert the expression into a set of clauses
5 Now resolve the clauses using suitable unifiers.

- This algorithm means we can write programs that automatically prove theorems using resolution.

此外，要注意不同規則**(rule)**之間，要使用或置換成不同的變數名稱

Example (substitution 1)



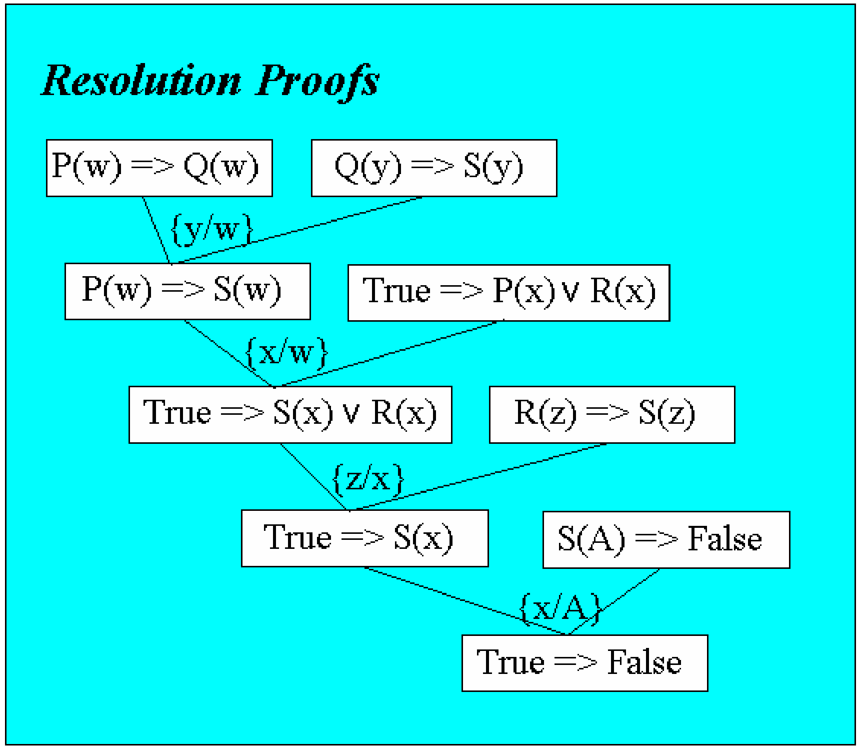
- 在本社區中
- **P(w)→Q(w)** 每個早起者都晨跑
- **Q(y)→S(y)** 每個晨跑者都健康
- **P(x) V R(x)** 每人都早起或快樂
- **R(z)→S(z)** 每個快樂者都健康
- **S(A)?** 某 A 健康嗎?

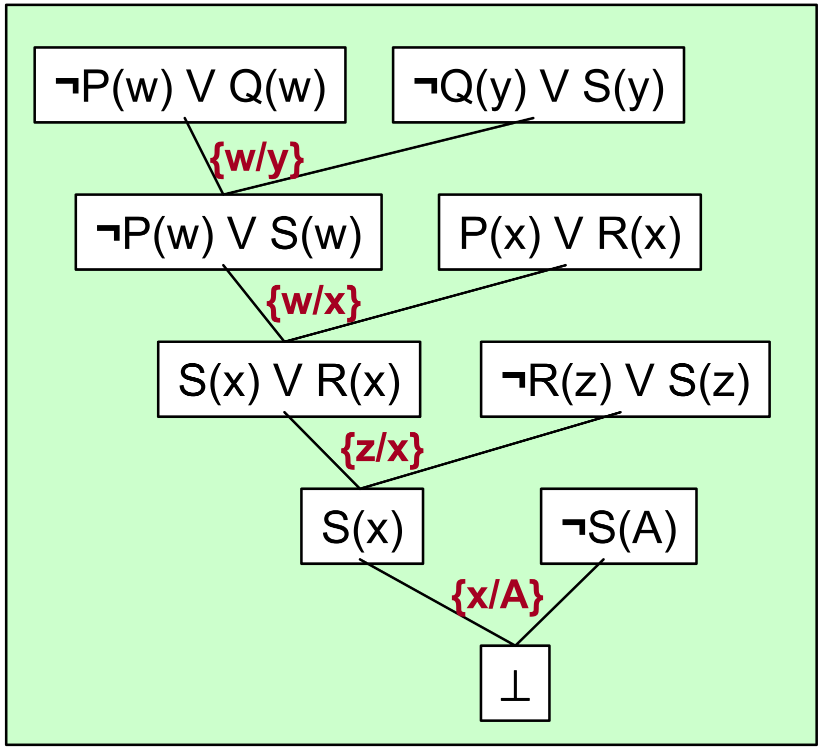
- 求 **S(A)→False** 是錯的

本例應把S(A)→False 提至最前面
才顯得出 Back chaining 的威力
而目前看到的推導過程是順推&逆推的混合系統

NOTE: 本例等同於用 CNF 做消去

Example (substitution 2)

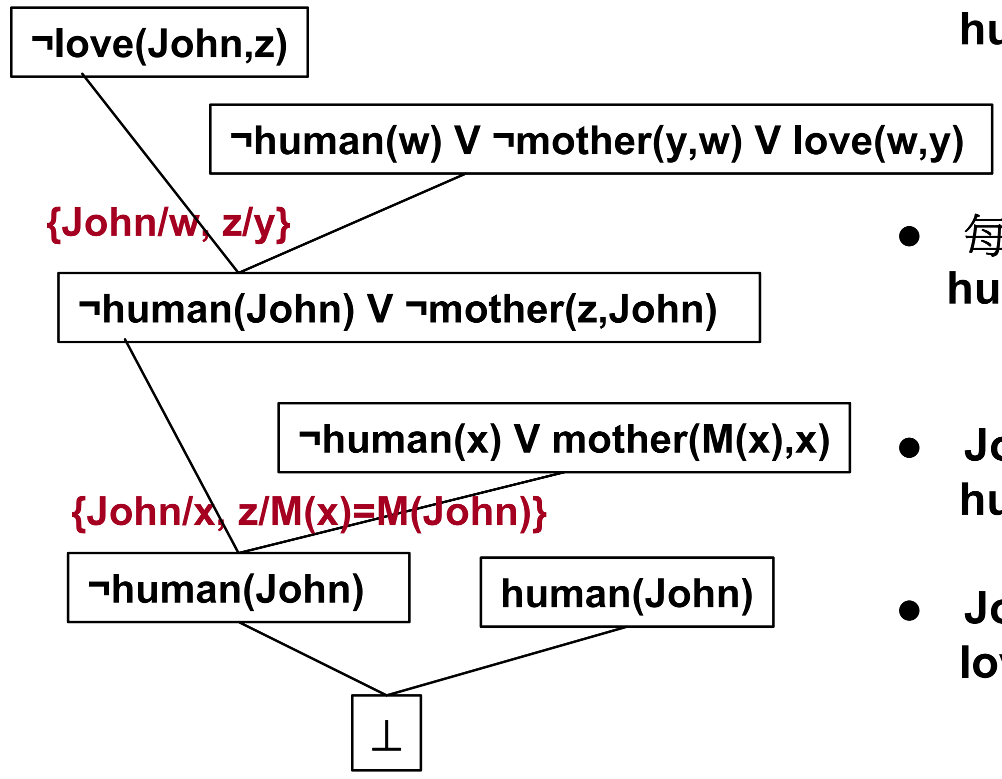




求 **S(A)→False** 是錯的

亦即求 **¬S(A)** 是錯的

Example (Skolem function)


¬love(John,z)
¬human(w) V ¬mother(y,w) V love(w,y)
{John/w, z/y}
¬human(John) V ¬mother(z,John)
¬human(x) V mother(M(x),x)
{John/x, z/M(x)=M(John)}
¬human(John)
human(John)
⊥

Ans: 原式成立, 而且 z= M(John)

- 每個人都存在有媽媽
  human(x) → mother(M(x), x)

  ∀(x) ∃(y) human(x) → mother(y, x)

- 每個人都愛他的媽媽
  human(w)Λmother(y, w) →love(w,y)

  ∀(w,y) human(w) ^ mother(y,w)→..

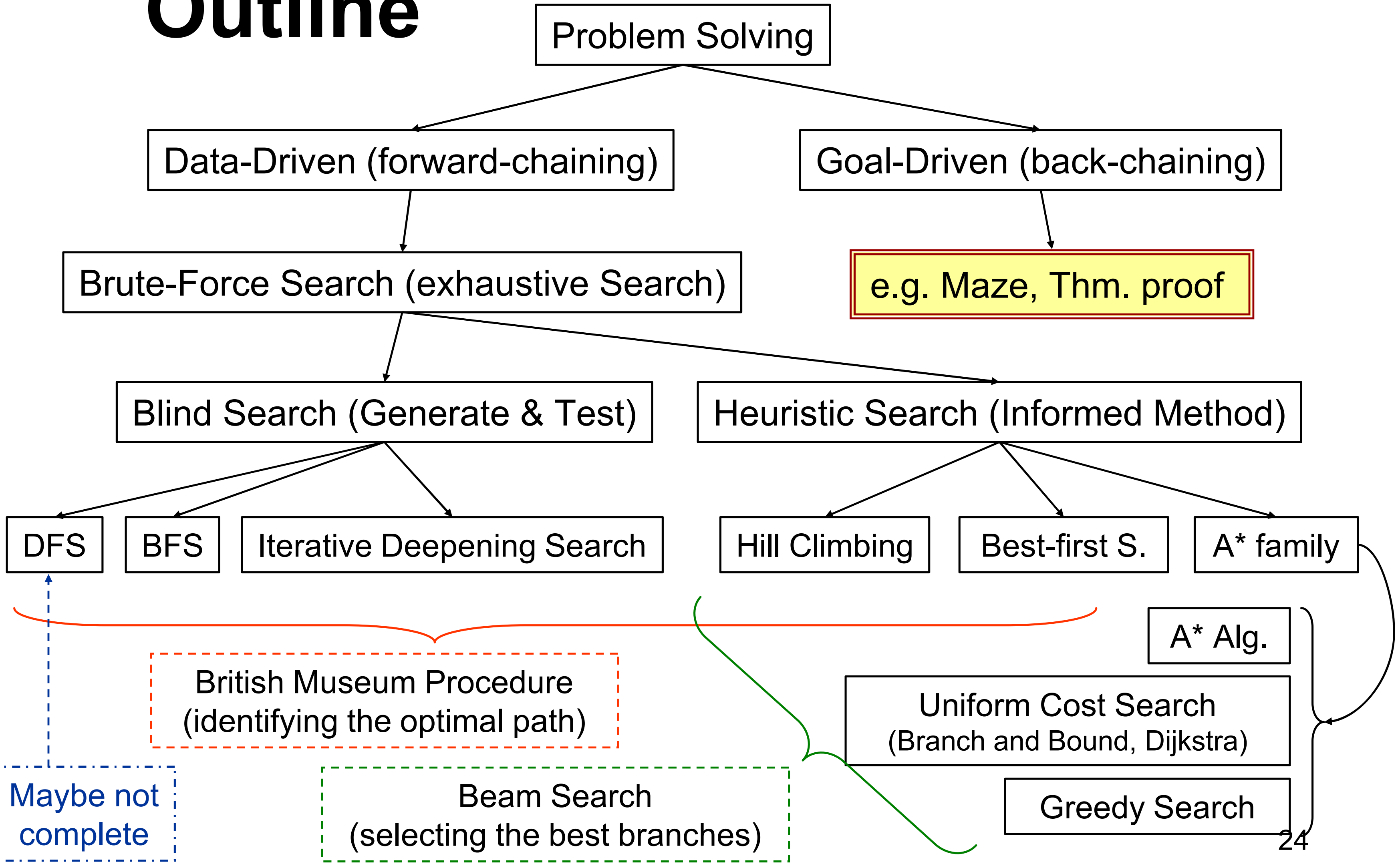
- **John** 是一個人
  human(John)

- **John** 有沒有他所愛的人？
  love(John,z)

  → 求 ¬ love(John,z) 是錯的

Recall:

Outline


Problem Solving
Data-Driven (forward-chaining)
Goal-Driven (back-chaining)
Brute-Force Search (exhaustive Search)
e.g. Maze, Thm. proof
Blind Search (Generate & Test)
Heuristic Search (Informed Method)
DFS
BFS
Iterative Deepening Search
Hill Climbing
Best-first S.
A* family
A* Alg.
Uniform Cost Search
(Branch and Bound, Dijkstra)
Greedy Search
British Museum Procedure
(identifying the optimal path)
Maybe not complete
Beam Search
(selecting the best branches)

Recall:

Deduction over FOPC --- forward-chaining

- Dog(X) ^ Meets(X,Y)^Dislikes(X,Y) → Barks_at(X,Y)
- Close_to(Z, DormG) → Meets(Snoopy, Z)
- Man(W) → Dislikes(Snoopy, W)
- Man(John), Dog(Snoopy), Close_to(John,DormG)

{John/W}

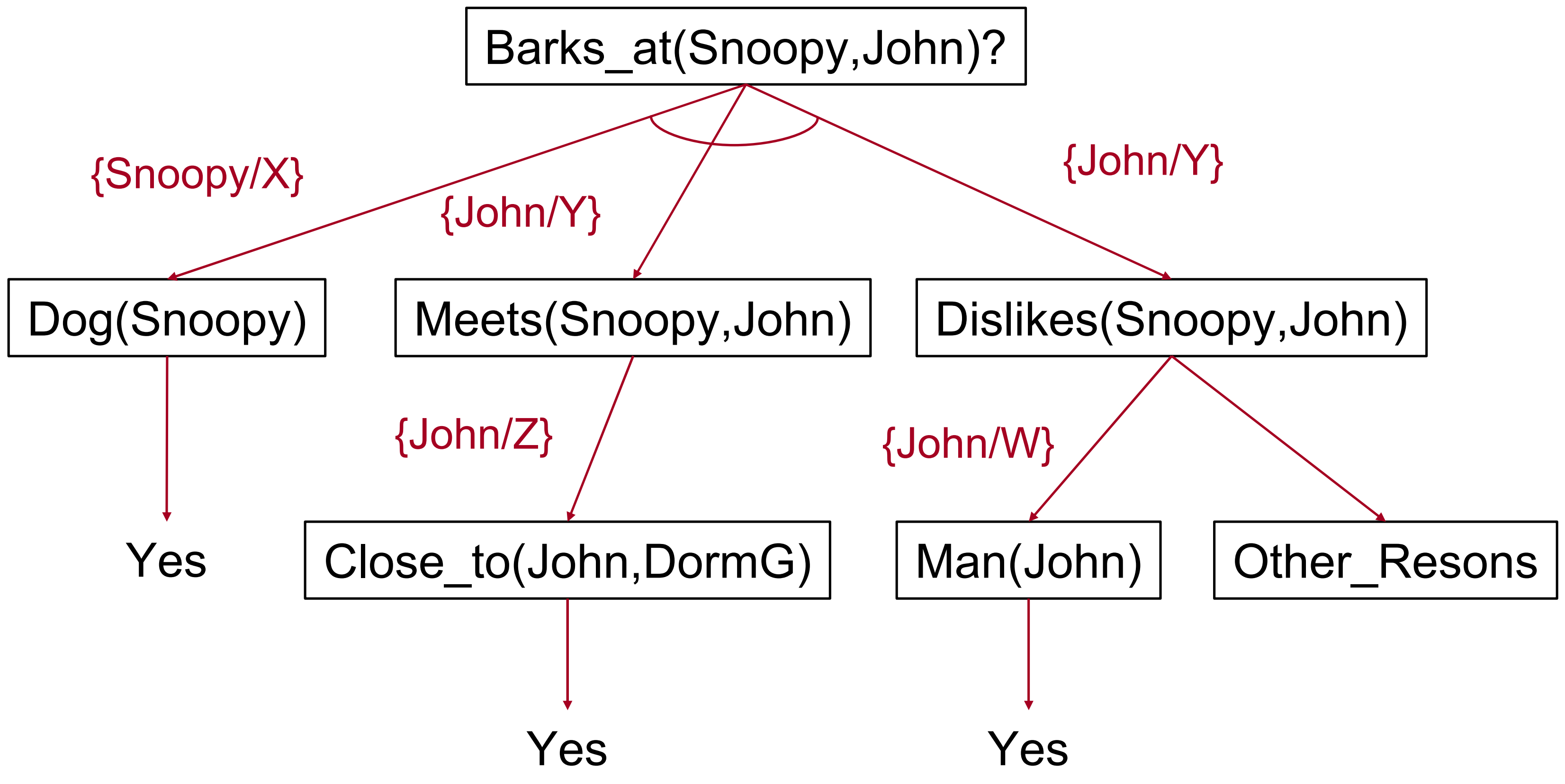
- Dog(X) ^ Meets(X,Y)^Dislikes(X,Y) → Barks_at(X,Y)
- Close_to(Z, DormG) → Meets(Snoopy, Z)
- Dislikes(Snoopy, John)
- Dog(Snoopy), Close_to(John,DormG)

??

Barks_at(Snoopy,John)

Recall:

Deduction over FOPC --- Goal Tree



Deduction over FOPC --- Resolution

$$\begin{tikzcd}
\text{Barks\_at(Snoopy,John)} \rightarrow \text{False} \arrow[d] & \text{Dog(X)} \wedge \text{Meets(X,Y)} \wedge \text{Dislikes(X,Y)} \rightarrow \text{Barks\_at(X,Y)} \arrow[dl, "\{X/Snoopy,\ Y/John\}"'] \\
\text{Dog(Snoopy)} \wedge \text{Meets(Snoopy,John)} \wedge \text{Dislikes(Snoopy,John)} \rightarrow \text{F} \arrow[dr] & \text{T} \rightarrow \text{Dog(Snoopy)} \arrow[d] \\
 & \text{Meets(Snoopy,John)} \wedge \text{Dislikes(Snoopy,John)} \rightarrow \text{False} \arrow[d, "\{Z/John\}"] \\
\text{Close\_to(Z,DormG)} \rightarrow \text{Meets(Snoopy, Z)} \arrow[r] & \text{Close\_to(John,DormG)} \wedge \text{Dislikes(Snoopy,John)} \rightarrow \text{False} \arrow[d] \\
\text{True} \rightarrow \text{Close\_to(John, DormG)} \arrow[r] & \text{Dislikes(Snoopy,John)} \rightarrow \text{False} \arrow[d, "\{W/John\}"] \\
\text{Man(W)} \rightarrow \text{Dislikes(Snoopy,W)} \arrow[r] & \text{Man(John)} \rightarrow \text{False} \arrow[d] \\
\text{True} \rightarrow \text{Man(John)} \arrow[r] & \text{True} \rightarrow \text{False}
\end{tikzcd}$$

Skip

Horn Clauses in PROLOG (1)

- **PROLOG uses resolution.**
- **A Horn clause has at most one positive literal:**

  $A \lor \neg B \lor \neg C \lor \neg D \lor \neg E \ldots$

- **This can also be written as an implication:**

  $B \land C \land D \land E \rightarrow A$

- **In PROLOG, this is written:**

  A :- B, C, D, E

Skip

Horn Clauses in PROLOG (2)

- Horn clauses can express rules:

  A :- B, C, D
- Or facts:

  A :-
- Or goals:

  :- B, C, D, E
- If a set of clauses is valid, PROLOG will definitely prove it using resolution and depth first search.