12.2.1 Resolution Principle(1)

- Resolution *refutation* proves a theorem by negating the statement to be proved and adding this negated goal to the set of axioms that are known to be true.
- Use the resolution rule of inference to show that this leads to a contradiction.
- Once the theorem prover shows that the negated goal is inconsistent with the given set of axioms, follows that the original goal must be consistent.

12.2.1 Resolution Principle(2)

- **Steps for resolution refutation proofs**
  - Put the premises or axioms into clause form(12.2.2).
  - Add the negation of what is to be proved, in clause form, to the set of axioms.
  - Resolve these clauses together, producing new clauses that logically follow from them(12.2.3).
  - Produce a contradiction by generating the empty clause.
  - The substitutions used to produce the empty clause are those under which the opposite of the negated goal is true(12.2.5).

12.2.1 Resolution Principle(3)

- **Prove that "Fido will die." from the statements**

  "Fido is a dog.",

  "All dogs are animals." and

  "All animals will die."

  - Changing premises to predicates
    - 𝛥(x) (dog(X) → animal(X))
    - dog(fido)
  - Modus Ponens and {fido/X}
    - animal(fido)
    - 𝛥(Y) (animal(Y) → die(Y))
  - Modus Ponens and {fido/Y}
    - die(fido)

12.2.1 Resolution Principle(4)

- **Equivalent Reasoning by Resolution**
  - Convert predicates to clause form

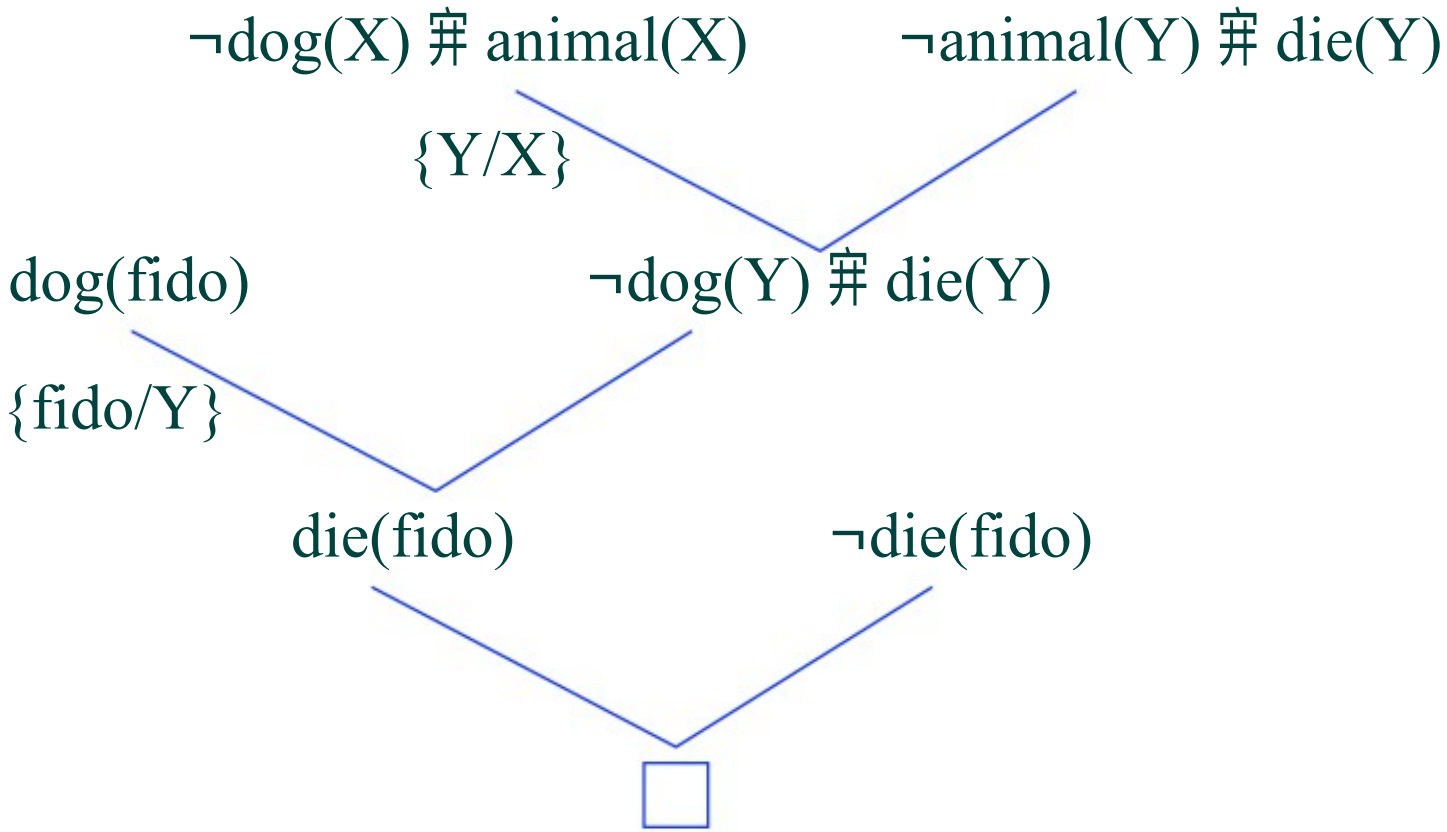
Predicate form	Clause form
1. ∀(x) (dog(X) → animal(X))	¬dog(X) ∨ animal(X)
2. dog(fido)	dog(fido)
3. ∀(Y) (animal(Y) → die(Y))	¬animal(Y) ∨ die(Y)

  - Negate the conclusion

4. ¬die(fido)	¬die(fido)

12.2.1 Resolution Principle(4)

- **Equivalent Reasoning by Resolution(continued)**



Resolution proof for the “dead dog” problem

12.2.2 Converting to Clause Form(1)

- **Step 1: Eliminate the logical connectives → and 审**
  - a 审 b = (a → b) 弁 (b → a)
  - a → b = ¬a 弁 b
- **Step 2: Reduce the scope of negation**
  - ¬(¬a) = a
  - ¬(a 弁 b) = ¬a 弁 ¬b
  - ¬(a 弁 b) = ¬a 弁 ¬b
  - ¬(宝X) a(X) = (宓X) ¬a(X)
  - ¬(宓X) b(X) = (宝X) ¬b(X)

12.2.2 Converting to Clause Form(2)

- **Step 3: Standardize by renaming all variables so that variables bound by different quantifiers have unique names.**
  - Ü (宓X) a(X) 幷 (宓X) b(X) = (宓X) a(X) 幷 (宓Y) b(Y)
- **Step 4: Move all quantifiers to the left to obtain a *prenex normal* form.**
- **Step 5: Eliminate existential quantifiers by using skolemization.**

12.2.2 Converting to Clause Form(3)

- **Step 6: Drop all universal quantifiers**
- **Step 7: Convert the expression to the conjunction of disjuncts form**
  - Ü (a 戓 b) 戓 (c 戓 d)
  - = (a 戓 (c 戓 d)) 戓 (b 戓 (c 戓 d))
  - = (a 戓 c) 戓 (a 戓 d) 戓 (b 戓 c) 戓 (b 戓 d)
- **step 8: Call each conjunct a separate clause**
- **step 9: Standardize the variables apart again.**

  Variables are renamed so that no variable symbol appears in more than one clause.

  (宓X)(a(X) 戓 b(X))=(宓X)a(X) 戓 (宓 Y)b(Y)

12.2.2 Converting to Clause Form(4)

- **Skolemization**
  - Skolem constant
    - (∃X)(dog(X)) may be replaced by dog(fido) where the name fido is picked from the domain of definition of X to represent that individual X.
  - Skolem function
    - If the predicate has more than one argument and the existentially quantified variable is within the scope of universally quantified variables, the existential variable must be a function of those other variables.
    - (∀X)(∃Y)(mother(X,Y)) ⇒ (∀X)mother(X,m(X))
    - (∀X)(∀Y)(∃Z)(∀W)(foo (X,Y,Z,W)) ⇒ (∀X)(∀Y)(∀W)(foo(X,Y,f(X,Y),w))

12.2.2 Converting to Clause Form(5)

- **Example of Converting Clause Form**

  (宓X)([a(X) 幷 b(X)] 宅 [c(X,I) 幷 (宝Y)((宝Z)[C(Y,Z)] 宅 d(X,Y))])
  幷 (宓 X)(e(X))

  - step 1: (宓X)(¬[a(X) 幷 b(X)] 幷 [c(X,I) 幷 (宝Y)(¬(宝Z)[c(Y,Z)]
    幷 d(X,Y))]) 幷 (宓x)(e(X))
  - step 2: (宓X)([¬a(X) 幷 ¬b(X)] 幷 [c(X,I) 幷 (宝Y)((宓Z)[¬c(Y,Z)]
    幷 d(X,Y))]) 幷 (宓x)(e(X))
  - step 3: (宓X)([¬a(X) 幷 ¬b(X)] 幷 [c(X,I) 幷 (宝Y)((宓Z)[¬c(Y,Z)]
    幷 d(X,Y))]) 幷 (宓W)(e(W))
  - step 4: (宓X)(宝Y)(宓Z)(宓W)([¬a(X) 幷 ¬b(X)] 幷 [c(X,I) 幷 (¬c(Y,Z)
    幷 d(X,Y))]) 幷 (e(W))
  - step 5: (宓X)(宓Z)(宓W)([¬a(X) 幷 ¬b(X)] 幷 [c(X,I) 幷 (¬c(f(X),Z)
    幷 d(X,f(X)))]) 幷 (e(W))
  - step 6: [¬a(X) 幷 ¬b(X)] 幷 [c(X,I) 幷 (¬c(f(X),Z) 幷 d(X,f(X)))]) 幷 e(W)

12.2.2 Converting to Clause Form(6)

- **Example of Converting Clause Form(continued)**
  - step 7: [가 ∨ 나] ∨ [다 ∨ (라 ∨ 마)] ∨ 바
    = [가 ∨나 ∨ 다 ∨ 바] ∨ [가 ∨ 나 ∨ 라 ∨ 마 ∨ 바]
    [¬a(X) ∨ ¬b(X) ∨ c(X,I) ∨ e(W)] ∨
    [¬a(X) ∨ ¬b(X) ∨ ¬c(f(X),Z) ∨ d(X,f(X)) ∨ e(W)]
  - step 8: (i) ¬a(X) ∨ ¬b(X) ∨ c(X,I) ∨ e(W)
    (ii) ¬a(X) ∨ ¬b(X) ∨ ¬c(f(X),Z) ∨ d(X,f(X)) ∨ e(W)
  - step 9: (i) ¬a(X) ∨ ¬b(X) ∨ c(X,I) ∨ e(W)
    (ii) ¬a(U) ∨ ¬b(U) ∨ ¬c(f(U),Z) ∨ d(U,f(U)) ∨ e(V)

12.2.3 Binary Resolution Proof Procedure(1)

- **Binary Resolution Step**
  - For any two clauses C_1 and C_2, if there is a literal L_1 in C_1 that is complementary to a literal L_2 in C_2, then delete L_1 and L_2 from C_1 and C_2 respectively, and construct the disjunction of the remaining clauses. The constructed clause is a resolvent of C_1 and C_2.
- **Examples of Resolution Step**
  - C_1=a ∨ ¬b, C_2=b ∨ c
    - Complementary literals : ¬b,b
    - Resolvent: a ∨ c
  - C_1=¬a ∨ b ∨ c, C_2=¬b ∨ d
    - Complementary literals : b, ¬b
    - Resolvent : ¬a ∨ c ∨ d

12.2.3 Binary Resolution Proof Procedure(2)

- **Justification of Resolution Step**
  - Theorem
    - Given two clause C_1 and C_2, a resolvent C of C_1 and C_2 is a logical consequence of C_1 and C_2.
  - Proof
    - Let $C_1 = L \vee C_1'$, $C_2 = \neg L \vee C_2'$, and $C = C_1' \vee C_2'$, where C_1' and C_2' are disjunction of literals.
    - Suppose C_1 and C_2 are true in an interpretation I.
    - We want to prove that the resolvent C of C_1 and C_2 is also true in I.
    - Case 1: L is true in I
      - Then since $C_2 = \neg L \vee C_2'$ is true in I, C_2' must be true in I, and thus $C = C_1' \vee C_2'$ is true in I.
    - Case 2: L is false in I
      - Then since $C_1 = L \vee C_1'$ is true in I, C_1' must be true in I. Thus, $C = C_1' \vee C_2'$ must be true in I.

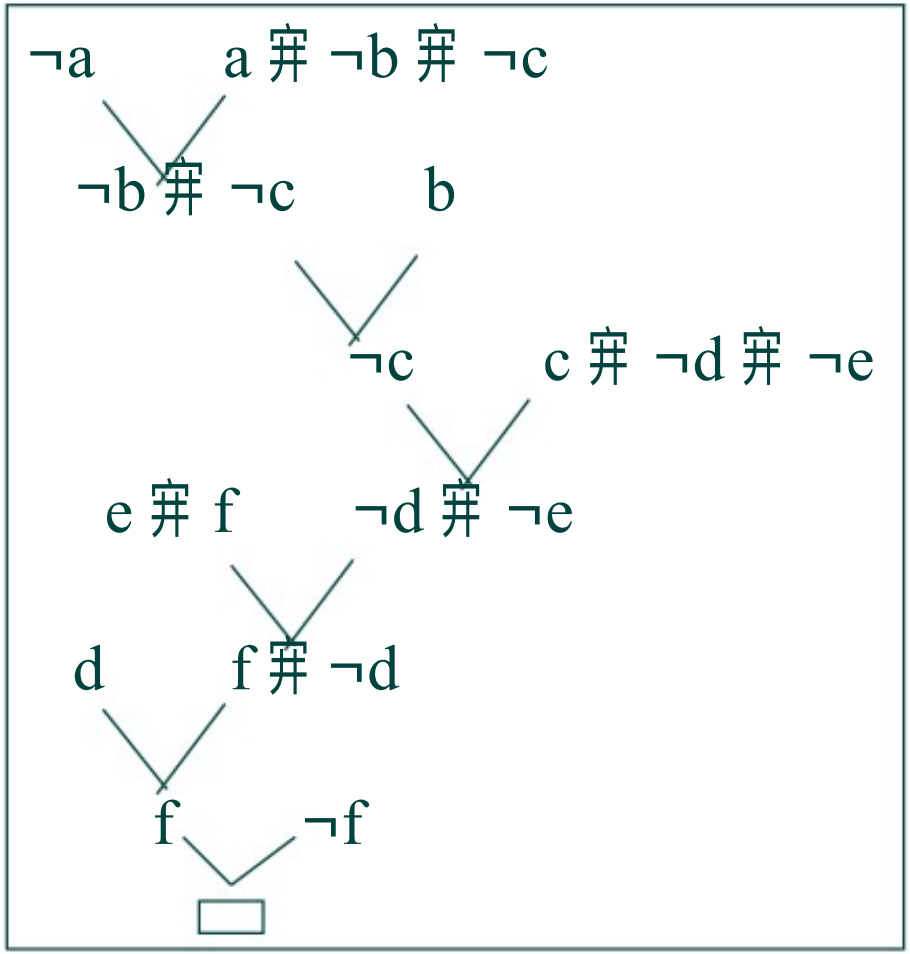
12.2.3 Binary Resolution Proof Procedure(3)

- **Resolution in Propositional Logic**

1. a ← b 并 c	a 并 ¬b 并 ¬c
2. b	b
3. c ← d 并 e	c 并 ¬d 并 ¬e
4. e 并 f	e 并 f
5. d 并 ¬ f	d
	¬ f

12.2.3 Binary Resolution Proof Procedure(4)

- **Resolution in Propositional Logic (continued)**
  - First, the goal to be proved, a , is negated and added to the clause set.
  - The derivation of □ indicates that the database of clauses is inconsistent.



12.2.3 Binary Resolution Proof Procedure(5)

- **Resolution on the predicate calculus**
  - A literal and its negation in parent clauses produce a resolvent only if they unify under some substitution σ. σ Is then applied to the resolvent before adding it to the clause set.
  - C_1 = ¬dog(X) ∨ animal(X)

    C_2 = ¬animal(Y) ∨ die(Y)

    Resolvent : ¬dog(Y) ∨ die(Y) {Y/X}

12.2.3 Binary Resolution Proof Procedure(6)

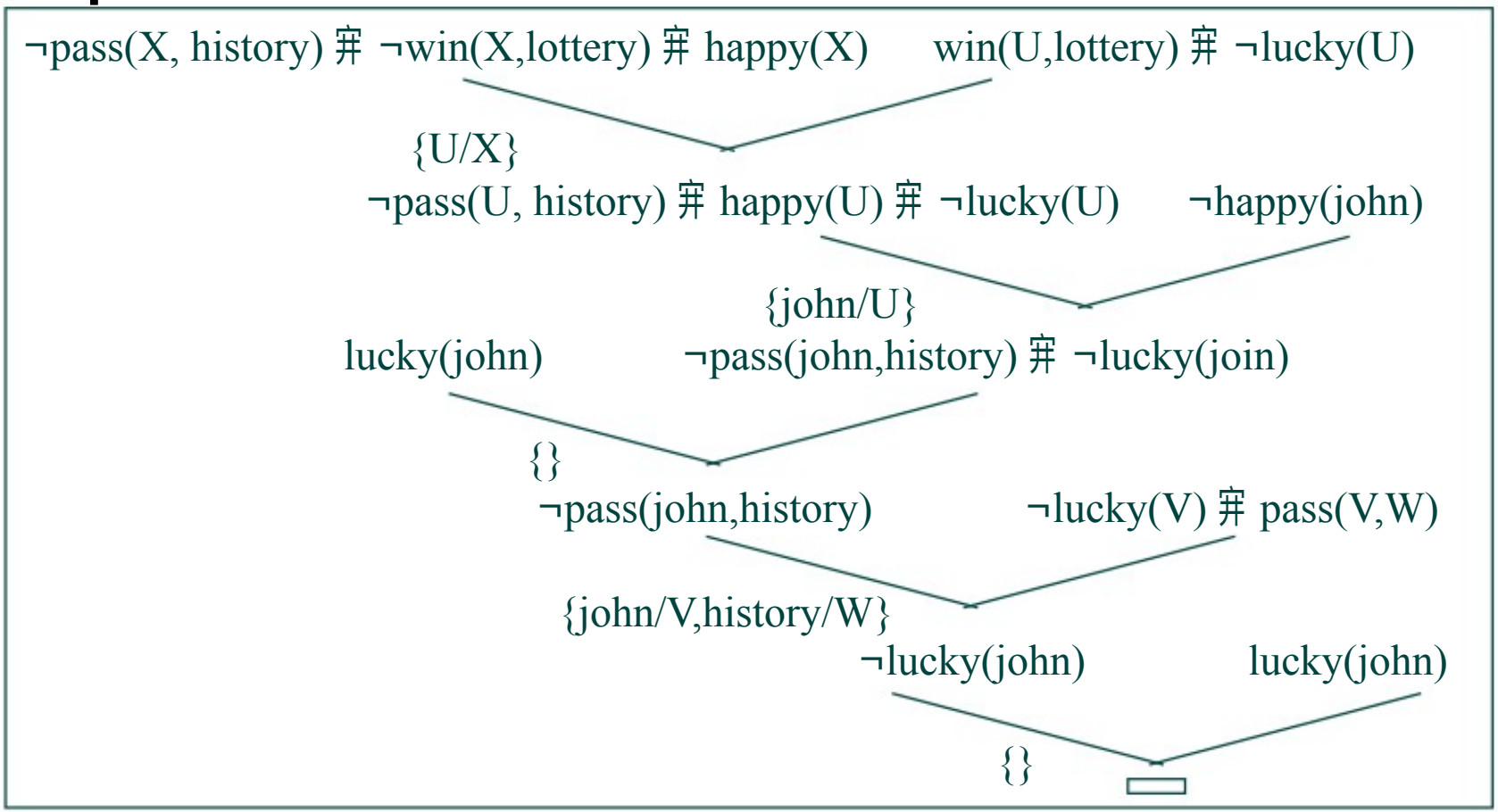
- **"Lucky student"**
  1. Anyone passing his history exams and winning the lottery is happy
     - ∀X(pass(X,history) ∧ win(X,lottery) → happy(X))
  2. Anyone who studies or is lucky can pass all his exams.
     - ∀X∀Y(study(X) ∨ lucky(X) → pass(X,Y))
  3. John did not study but he is lucky
     - ¬study(john) ∨ lucky(john)
  4. Anyone who is lucky wins the lottery.
     - ∀X(lucky(X) → win(X,lottery))

12.2.3 Binary Resolution Proof Procedure(7)

- **Clause forms of "Lucky student"**
  1. ¬pass(X,history) ∨ ¬win(X,lottery) ∨ happy(X)
  2. ¬study(X) ∨ pass(Y,Z)
     ¬lucky(W) ∨ pass(W,V)
  3. ¬study(john)
     lucky(john)
  4. ¬lucky(V) ∨ win(V,lottery)
  5. Negate the conclusion "John is happy"
     ¬happy(john)

12.2.3 Binary Resolution Proof Procedure(8)

- **Resolution refutation for the "Lucky Student" problem**



12.2.3 Binary Resolution Proof Procedure(9)

- **"Exciting Life"**
  1. All people who are not poor and are smart are happy.
     - ∀X(¬poor(X) ∧ smart(X) → happy(X))
  2. Those people who read are not stupid.
     - ∀Y(read(Y) → smart(Y))
     - {assume ∀X(smart(X) ∨ ¬stupid(X))}
  3. John can read and is wealthy.
     - read(john) ∧ ¬poor(john)
     - {assume ∀Y(wealthy(Y) ∨ ¬poor(Y))}
  4. Happy people have exciting lives.
     - ∀Z(Happy(Z) → exciting(Z))
  5. Negate the conclusion.
     - "Can anyone be found with an exciting life?"
     - ∃X(exciting(W))

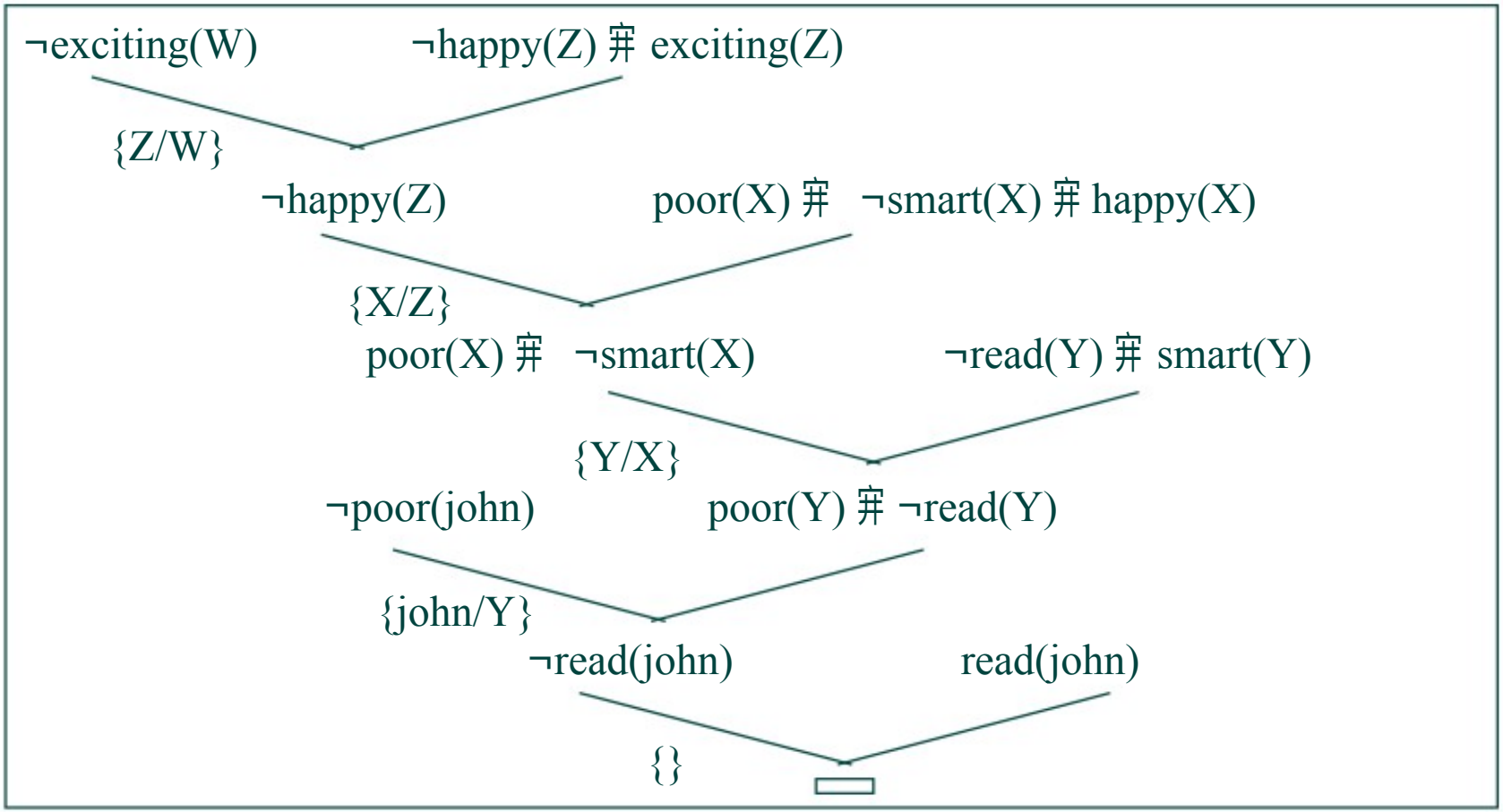
12.2.3 Binary Resolution Proof Procedure(10)

- **Clause forms of "exciting life"**
  1. poor(X) ∨ ¬smart(X) ∨ happy(X)
  2. ¬read(Y) ∨ smart(Y)
  3. read(john)

     ¬poor(john)
  4. ¬happy(Z) ∨ exciting(Z)
  5. ¬exciting(W)

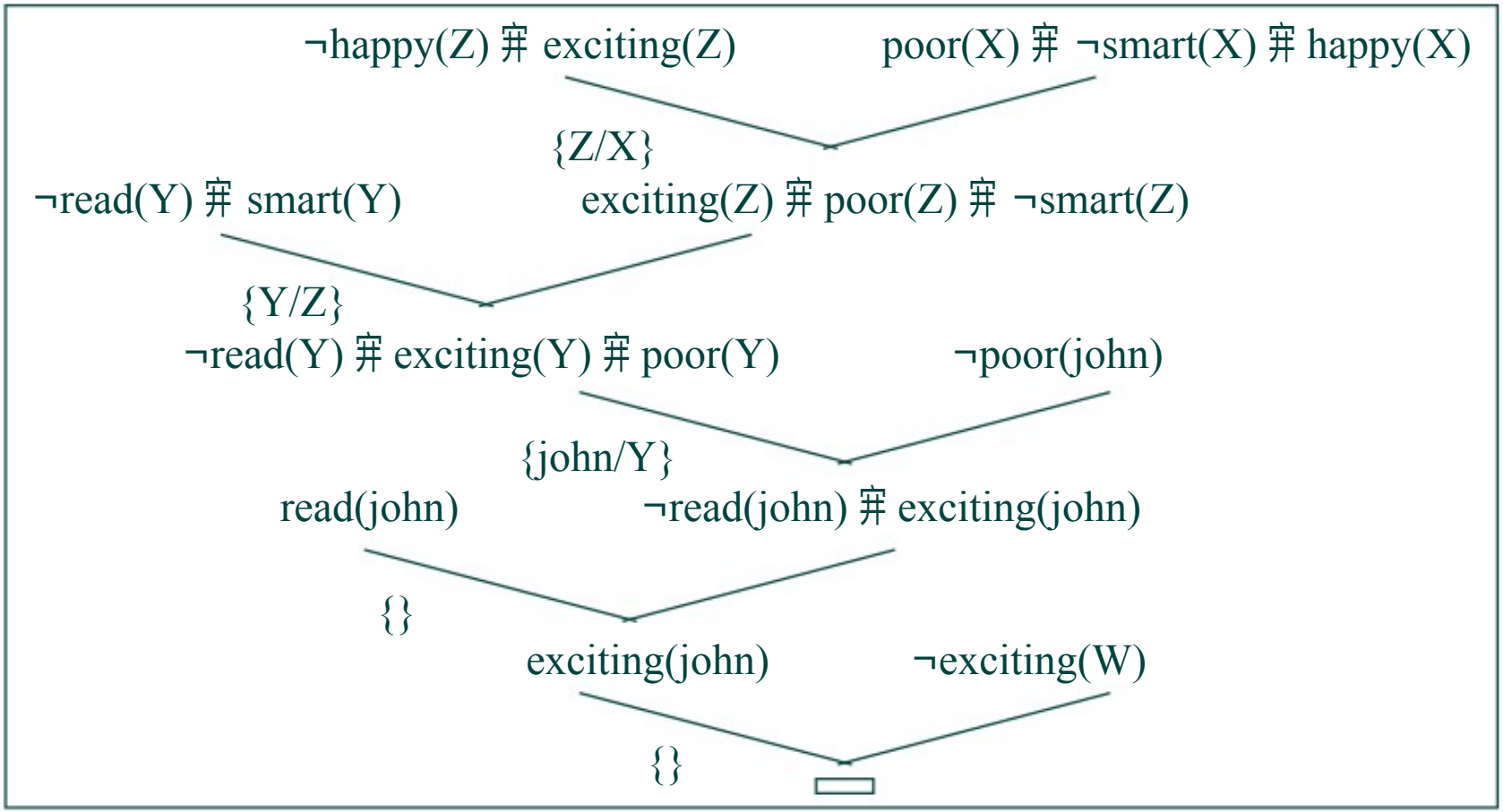
12.2.3 Binary Resolution Proof Procedure(11)

- **Resolution refutation for the "exciting life"**



12.2.3 Binary Resolution Proof Procedure(12)

- **Another resolution refutation for "exciting life"**



12.2.4 Strategies for Resolution(1)

- **Order of clause combination is important**
  - N clauses → N_2 ways of combinations or checking to see whether they can be combined
  - Search heuristics are very important in resolution proof procedures
- **Strategies**
  - Breadth-First Strategy
  - Set of Support Strategy
  - Unit Preference Strategy
  - Linear Input Form Strategy

12.2.4 Strategies for Resolution(2)

- **Breadth-First Strategy**
  - Ü First round: each clause is compared for resolution with every other clause in the clause space.
  - Ü Second round: generate the clauses by resolving the clauses produced at the first round with all the original clauses.
  - Ü Nth round: generate the clauses by resolving all clauses at level n-1 against the elements of the original clause set and all clauses previously produced.
  - Ü Characteristics
    - Ø it guarantees finding the shortest solution path because it generates every search state for each level before going any deeper.
    - Ø It is a complete strategy in that it is guaranteed to find a refutation if one exists.
  - Ü Figure 12.8 (p. 530)

12.2.4 Strategies for Resolution(3)

- **Set of Support Strategy**
  - Specify a subset(T) of a set of input clauses(S) called the set of support such that S-T is satisfiable.
  - Require that resolvents in each resolution have an ancestor in the set of support(T).
  - Based on the insight that the negation of what we want to prove true is going to be responsible for causing the clause space to be contradictory.
  - Forces resolutions between clauses of which at least one is either the negated goal clause or a clause produced by resolutions on the negated goal.
  - Figure 12.6 (p. 528)

12.2.4 Strategies for Resolution(4)

- **Unit Preference Strategy**
  - Try to produce a resultant clause that has fewer literals than the parent clauses.
  - Resolving with a clause of one literal, called a unit clause, will guarantee that the resolvent is smaller than the largest parent clause.
  - Figure 12.9 (p. 531)

12.2.4 Strategies for Resolution(5)

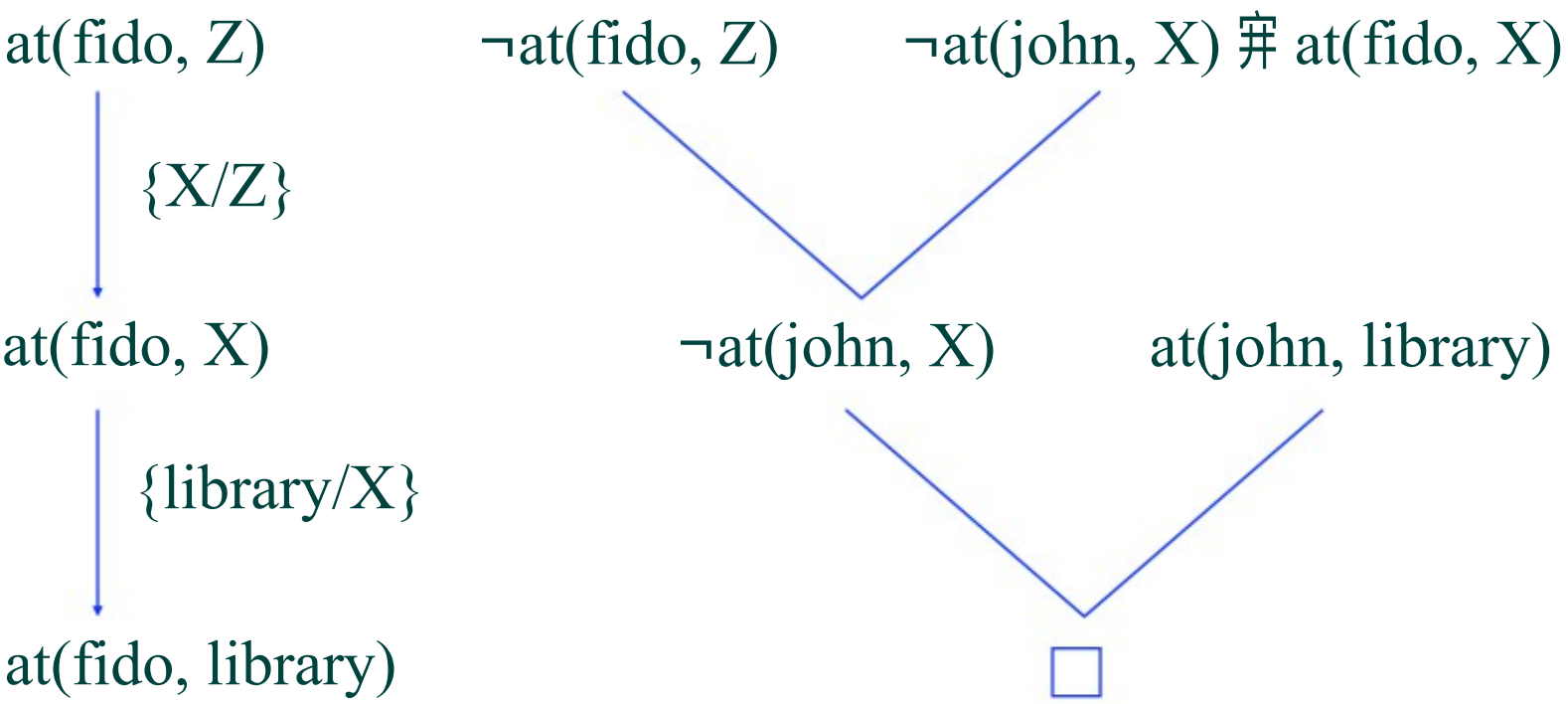
- **Linear Input Form Strategy**
  - a direct use of the negated goal and the original axioms
  - Take the negated goal and resolve it with one of the axioms. The result is then resolved with one of the axioms to get another new clause. The new clause is again resolved with one of the axioms.
  - Try to resolve the most recently obtained clause with the original axioms.

12.2.5 Answer Extraction from Resolution Refutations(1)

- **Extract the correct answer by retaining information on the unification substitutions made in the resolution refutation.**
  - Retain the original conclusion that was to be proved.
  - Introduce each unification made in the resolution process into the conclusion.
- **Example of answer extraction**
  - Fido goes wherever John goes.
    - at(john, X) → at(fido, X)
  - John is at the library.
    - at(john, library)
  - Negate the conclusion “Where is Fido?”
    - ¬at(fido, Z)

12.2.5 Answer Extraction from Resolution Refutations(2)

- **Example of answer extraction(continued)**



12.2.5 Answer Extraction from Resolution Refutations(3)

q **"Exciting Life"**