EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

MATHEMATISCH-
NATURWISSENSCHAFTLICHE FAKULTÄT

Artificial Intelligence
Chapter 9: Inference in First-Order Logic

Andreas Zell

After the Textbook: Artificial Intelligence,
A Modern Approach
by Stuart Russell and Peter Norvig (3rd Edition)

9 Inference in First-Order Logic
Outline

EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

- We discuss 3 approaches for inference in first-order logic (FOL): forward-chaining, backward-chaining, & resolution
- *Entailment in first-order logic is semidecidable*
  - There are algorithms that can correctly identify every entailed sentence
  - No algorithm can correctly identify every non-entailed sentence

9.1 Propositional vs 1st-Order Inference

- We know how to do inference in propositional logic, but not in first-order logic.
- Idea: convert first-order logic to proposit. logic
  - Resulting knowledge base is *not* logically equivalent
  - It is inferentially equivalent; it is satisfiable exactly when the original knowledge base is satisfiable
- We use simple inference rules to convert quantified statements to propositional sentences
- We use SUBST(θ,α) to denote the result of applying the substitution θ to sentence α:

$$\text{SUBST}(\{x/\mathit{John}\}, \mathit{King}(x)) \mapsto \mathit{King}(\mathit{John})$$

9.1 Propositional vs 1st-Order Inference
Inference Rules for Quantifiers

EBERHARD KARLS UNIVERSITÄT TÜBINGEN

- Universal Instantiation – substitute variable of universally quantified statement with ground term
  - Applied many times, producing several consequences

$$\frac{\forall v \quad \alpha}{\mathrm{SUBST}\left(\{v/g\}, \alpha\right)}$$

  - I.e., all universally quantified sentences α can be grounded by replacing universal quantifier v with any ground term g which has no variables
  - Ex. $\forall x \quad King(x) \wedge Greedy(x) \Rightarrow Evil(x)$ becomes

$$King(John) \wedge Greedy(John) \Rightarrow Evil(John) \qquad \{x/John\}$$
$$King(Richard) \wedge Greedy(Richard) \Rightarrow Evil(Richard) \qquad \{x/Richard\}$$
$$King(Father(John)) \wedge Greedy(Father(John)) \Rightarrow Evil(Father(John)) \qquad \{x/Father(John)\}$$
$$\ldots \qquad \ldots$$

9.1 Propositional vs 1st-Order Inference Inference Rules for Quantifiers

EBERHARD KARLS UNIVERSITÄT TÜBINGEN

- Existential Instantiation: substituting a ground term, Skolem constant, for the variable in an existential statement – this process is called skolemization
  - Skolem constant is new symbol k not previously in KB
  - Only applied once & then sentence α is discarded

$$\frac{\exists v \quad \alpha}{\text{SUBST}\left(\{v/k\}, \alpha\right)}$$

  - If existential quantifier is embedded in a universally quantified statement, use a skolem function - a new generic function of the universally quantified variable
  - Ex. $\exists x \quad Crown(x) \wedge Gold(x)$ becomes $Crown(C_1) \wedge Gold(C_1)$
  - Ex. $\forall y \quad King(y) \Rightarrow \exists x \quad Crown(x)$ becomes
    $\forall y \quad King(y) \Rightarrow Crown(C_2(y))$

9.1 Propositional vs 1st-Order Inference
Reduction to Propositional Inference

EBERHARD KARLS UNIVERSITÄT TÜBINGEN

- Propositionalization: the process of converting a first-order sentence into propositional logic
  - Idea: existential sentence replaced by one instantiation & universal sentence replaced by *all* possible instantiations
  - A propositionalized knowledge base is *inferentially equivalent* to the original KB but not *logically equivalent*
  - Entailment is preserved thus resolution is *complete*
  - Functions cause the number of possible ground terms to be infinite; ex: *Father(John), Father(Father(John)), …*
  - However, if a sentence is entailed by KB, there is a proof requiring only a finite subset of the propositions (Herbrand)
- Entailment for FOL is semidecidable: any entailed sentence can be proved

9.2 Unification and Lifting
A First-Order Inference Rule

EBERHARD KARLS UNIVERSITÄT TÜBINGEN

- Preceding approach very inefficient (many unnecessary rules are created); instead we use inference rules design for FOL:
- Generalized Modus Ponens: A lifted version of Modus Ponens for FOL. For atomic sentences p_i, p_i', and q, where $\exists\theta\left(\forall i \quad \text{SUBST}\left(\theta, p_i'\right) = \text{SUBST}\left(\theta, p_i\right)\right)$

$$\frac{p_1', p_2', \ldots, p_n' \qquad \left(p_1 \wedge p_2 \wedge \ldots \wedge p_n \Rightarrow q\right)}{SUBST\left(\theta, q\right)}$$

  - That is, there is a substitution θ that allows us to match each literal p_i in the clause's body with a fact p'_i in our KB, and thus, we can conclude that $SUBST(\theta, q)$ follows.

9.2 Unification and Lifting
Ex: Generalized Modus Ponens

$\forall x \quad King(x) \wedge Greedy(x) \Rightarrow Evil(x)$

$King(John)$

$Greedy(John)$

Matching for Modus Ponens	
p_1' is $King(John)$	p_1 is $King(x)$
p_2' is $Greedy(John)$	p_2 is $Greedy(x)$
θ is $\{x/John\}$	q is $Evil(x)$
SUBST(θ,q) is $Evil(John)$	

$$\frac{King(John) \quad Greedy(John) \qquad King(x) \wedge Greedy(x) \Rightarrow Evil(x)}{p_1', p_2', \ldots, p_n' \qquad \left(p_1 \wedge p_2 \wedge \ldots \wedge p_n \Rightarrow q\right)}$$
$$SUBST\left(\theta, q\right)$$
$$Evil(John)$$

Thus, we can conclude $Evil(John)$

9.2 Unification and Lifting
Unification

EBERHARD KARLS UNIVERSITÄT TÜBINGEN

- Unification – the process of finding substitutions that make different logical expressions look identical.
  - Given two sentences, p and q, UNIFY returns the most general unifier θ, if a unifier exists.

$$\text{UNIFY}(p, q) = \theta \quad \text{where} \quad \text{SUBST}(\theta, p) = \text{SUBST}(\theta, q)$$

  - standardizing apart: rename variables to avoid name clashes
    - Ex. To unify $Knows(John,x)$ & $Knows(x,Jim)$, replace 2nd x with x_2
  - most general unifier (MGU): for any unifiable pair of expressions there is a unique unifier more general than any other
    - A unifier is *more general* than another unifier if the first places fewer restrictions on the values of the variables
    - Ex: $\text{UNIFY}(Knows(John,x),Knows(y,z)) = \{y/John, x/z\}$
  - occur check: when matching variable against a complex term, must check whether the variable itself occurs in the term, in which case the unification fails; ex: $S(x)$ can't unify with $S(S(x))$

9.2 Unification and Lifting
Unification Algorithm

EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

function UNIFY(x, y, θ) **returns** a substitution to make x and y identical
  inputs: x, a variable, constant, list, or compound expression
    y, a variable, constant, list, or compound expression
    θ, the substitution built up so far (optional, defaults to empty)

  if θ = failure **then return** failure
  else if $x = y$ **then return** θ
  else if VARIABLE?(x) **then return** UNIFY-VAR(x, y, θ)
  else if VARIABLE?(y) **then return** UNIFY-VAR(y, x, θ)
  else if COMPOUND?(x) **and** COMPOUND?(y) **then**
    return UNIFY(x.ARGS, y.ARGS, UNIFY(x.OP, y.OP, θ))
  else if LIST?(x) **and** LIST?(y) **then**
    return UNIFY(x.REST, y.REST, UNIFY(x.FIRST, y.FIRST, θ))
  else return failure

function UNIFY-VAR(var, x, θ) **returns** a substitution

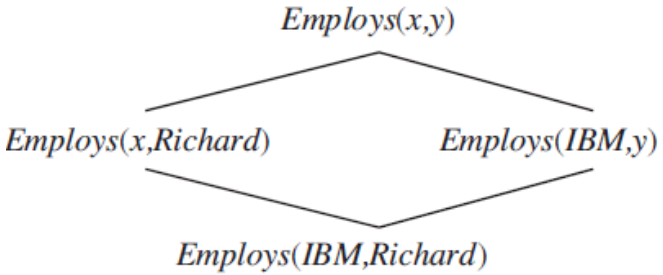
  if $\{var/val\} \in \theta$ **then return** UNIFY(val, x, θ)
  else if $\{x/val\} \in \theta$ **then return** UNIFY(var, val, θ)
  else if OCCUR-CHECK?(var, x) **then return** failure
  else return add $\{var/x\}$ to θ

9.2 Unification and Lifting
Efficient Storage & Retrieval

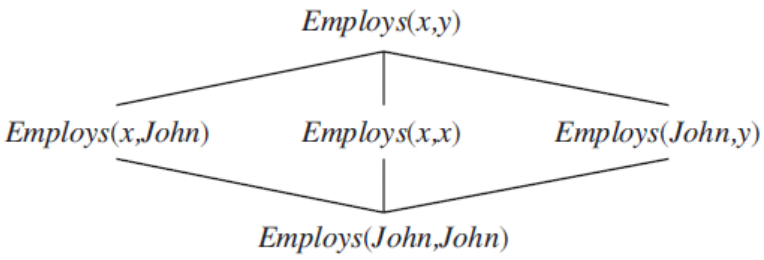
- To implement *TELL* & *ASK* for a KB we need primatives *STORE* & *FETCH*
  - *STORE(s)* stores s in the KB
  - *FETCH(q)* returns all unifiers to make q unify with s in KB
  - Special data structures are used to make these efficient
- predicate indexing: store all sentences with a predicate together for fast lookup
  - Works well with many predicates each used in few clauses
  - May have additional indices by arguments to predicate
    - E.g. A company may have many clauses with *Employs*
    - We can create 2nd index to its first argument *Employs(IBM,x)* or second argument *Employs(y,Richard)*

9.2 Unification and Lifting
Efficient Storage & Retrieval

- Storing α: index all possible queries that unify with it
  - Ex: *Employs(IBM,Jim)* unifies with *Employs(x,Jim)*, *Employs(IBM,y)* & *Employs(x,y)*
- subsumption lattice: lattice based on the MGU
  - Child of a node is obtained by a single substitution
  - The least common ancestor of any 2 nodes is their MGU
  - A predicate with n arguments requires $O(2^n)$ nodes
  - Function storage is exponential in size of terms in sentence to be stored



Lattice for *Employs(IBM,Richard)*



Lattice for *Employs(John,John)*

9.3 Forward Chaining
A Forward-Chaining Algorithm

EBERHARD KARLS UNIVERSITÄT TÜBINGEN

- Here we extend forward-chaining (FC) to FOL
- FC uses generalized Modus Ponens to infer atomic sentences in KB until no new inference is possible
  - FC terminates when either goal reached or no new inference is possible; i.e., identical upto names of variables
    - fixed point: a state of the knowledge base for which no new sentences can be inferred
  - FC is *sound* & for definite clause KBs it is also *complete*
  - However, FOL inference with definite clauses is only *semidecidable* due to functions – FC may not terminate if query is unanswerable
    - Datalog KB: set of first-order definite clauses (disjunctions with exactly one positive literal) with no function symbols

9.3 Forward Chaining
First-Order Definite Clauses

- First-order definite clause: a disjunction of literals for which exactly one is positive
  - Either an atomic clause or an implication whose body is conjunction of positive literals & head is positive literal
  - May also include (universally quantified) variables
- **Ex:** Americans who sell weapons to hostiles are criminals. Nono is an enemy. An American, Colonel West, sold them missiles.

$$American(x) \wedge Weapon(y) \wedge Sells(x, y, z) \wedge Hostile(z) \Rightarrow Criminal(x)$$

$$Owns(Nono, M1)$$

$$Missile(M1)$$

$$Missile(x) \wedge Owns(Nono, x) \Rightarrow Sells(West, x, Nono)$$

$$Missile(x) \Rightarrow Weapon(x)$$

$$Enemy(x, America) \Rightarrow Hostile(x)$$

$$American(West)$$

$$Enemy(Nono, America)$$

9.3 Forward Chaining
Forward-Chaining Algorithm

EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

function FOL-FC-ASK(KB, α) **returns** a substitution or *false*
  inputs: KB, the knowledge base, a set of first-order definite clauses
    α, the query, an atomic sentence
  local variables: *new*, the new sentences inferred on each iteration

  repeat until *new* is empty
    new $\leftarrow \{\}$
    for each *rule* **in** KB **do**
      $(p_1 \wedge \ldots \wedge p_n \Rightarrow q) \leftarrow$ STANDARDIZE-VARIABLES(*rule*)
      for each θ such that SUBST($\theta, p_1 \wedge \ldots \wedge p_n$) = SUBST($\theta, p'_1 \wedge \ldots \wedge p'_n$)
          for some $p'_1, \ldots, p'_n$ in KB
        $q' \leftarrow$ SUBST(θ, q)
        if q' does not unify with some sentence already in KB or *new* **then**
          add q' to *new*
          $\phi \leftarrow$ UNIFY(q', α)
          if ϕ is not *fail* **then return** ϕ
    add *new* to KB
  return *false*

9.3 Forward Chaining
Example: Prove West is Criminal

$American(x) \wedge Weapon(y) \wedge Sells(x, y, z) \wedge Hostile(z) \Rightarrow Criminal(x)$

$Owns(Nono, M1)$

$Missile(M1)$

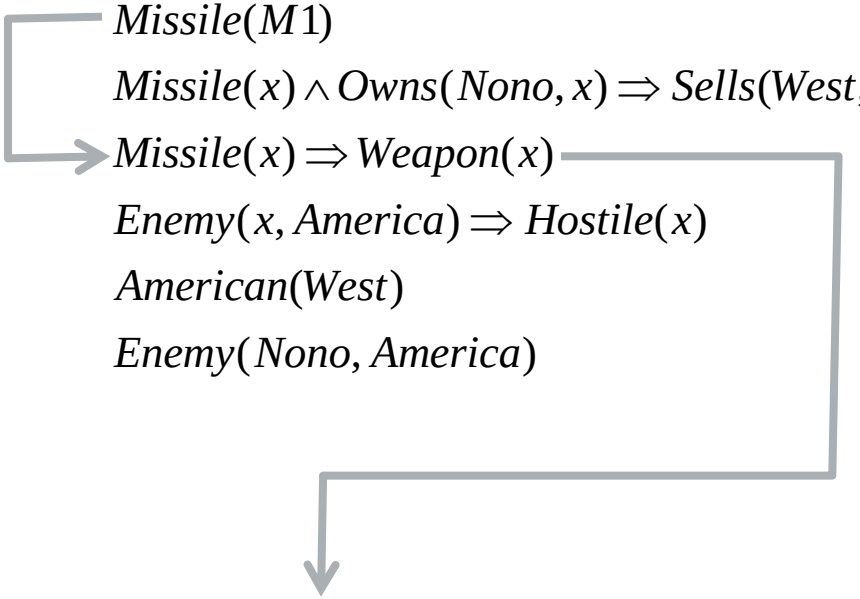
$Missile(x) \wedge Owns(Nono, x) \Rightarrow Sells(West, x, Nono)$

$Missile(x) \Rightarrow Weapon(x)$

$Enemy(x, America) \Rightarrow Hostile(x)$

$American(West)$

$Enemy(Nono, America)$



$Weapon(M1)$

9.3 Forward Chaining
Example: Prove West is Criminal

$$American(x) \wedge Weapon(y) \wedge Sells(x, y, z) \wedge Hostile(z) \Rightarrow Criminal(x)$$

$Owns(Nono, M1)$

$Missile(M1)$

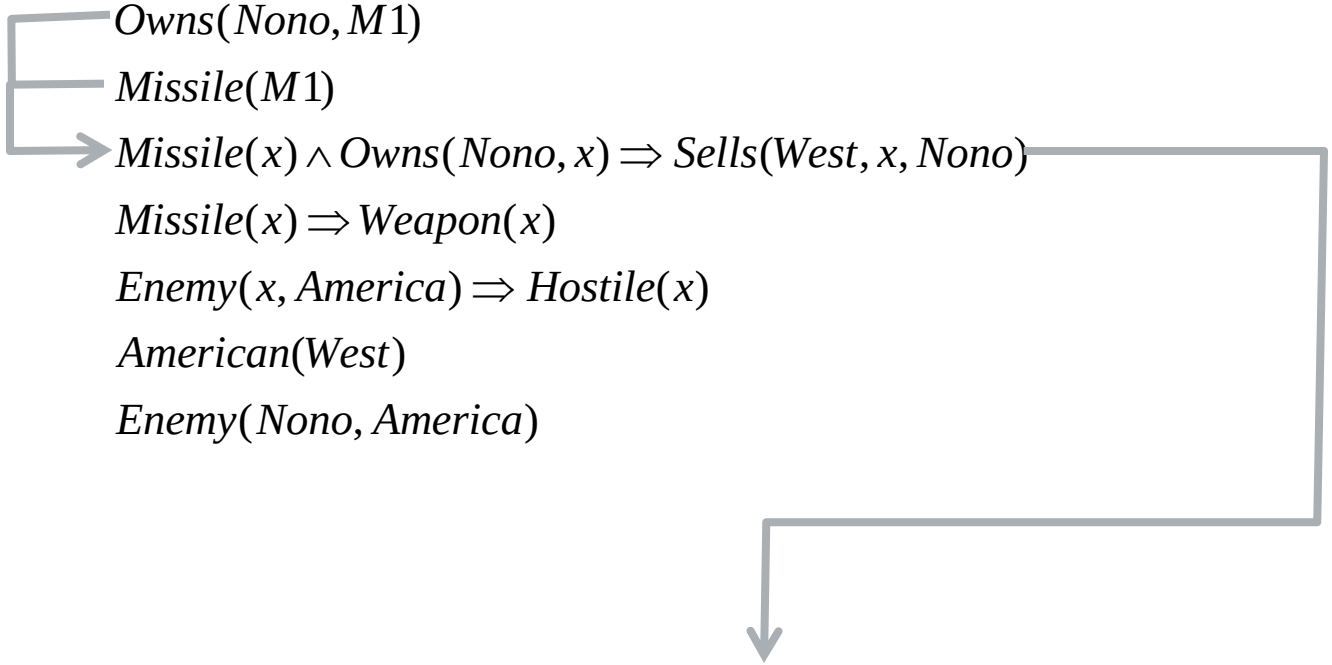
$Missile(x) \wedge Owns(Nono, x) \Rightarrow Sells(West, x, Nono)$

$Missile(x) \Rightarrow Weapon(x)$

$Enemy(x, America) \Rightarrow Hostile(x)$

$American(West)$

$Enemy(Nono, America)$



$Weapon(M1)$ $Sells(West, M1, Nono)$

9.3 Forward Chaining
Example: Prove West is Criminal

$American(x) \wedge Weapon(y) \wedge Sells(x, y, z) \wedge Hostile(z) \Rightarrow Criminal(x)$

$Owns(Nono, M1)$

$Missile(M1)$

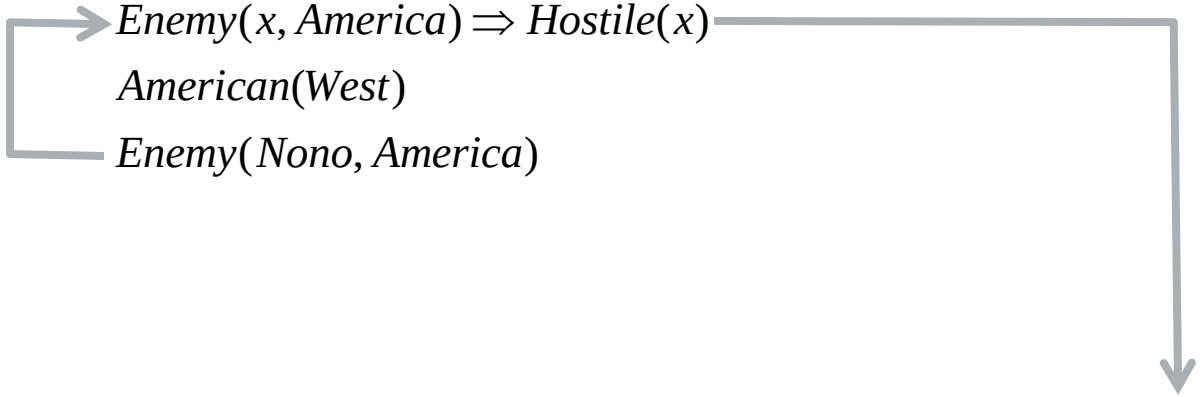
$Missile(x) \wedge Owns(Nono, x) \Rightarrow Sells(West, x, Nono)$

$Missile(x) \Rightarrow Weapon(x)$

$Enemy(x, America) \Rightarrow Hostile(x)$

$American(West)$

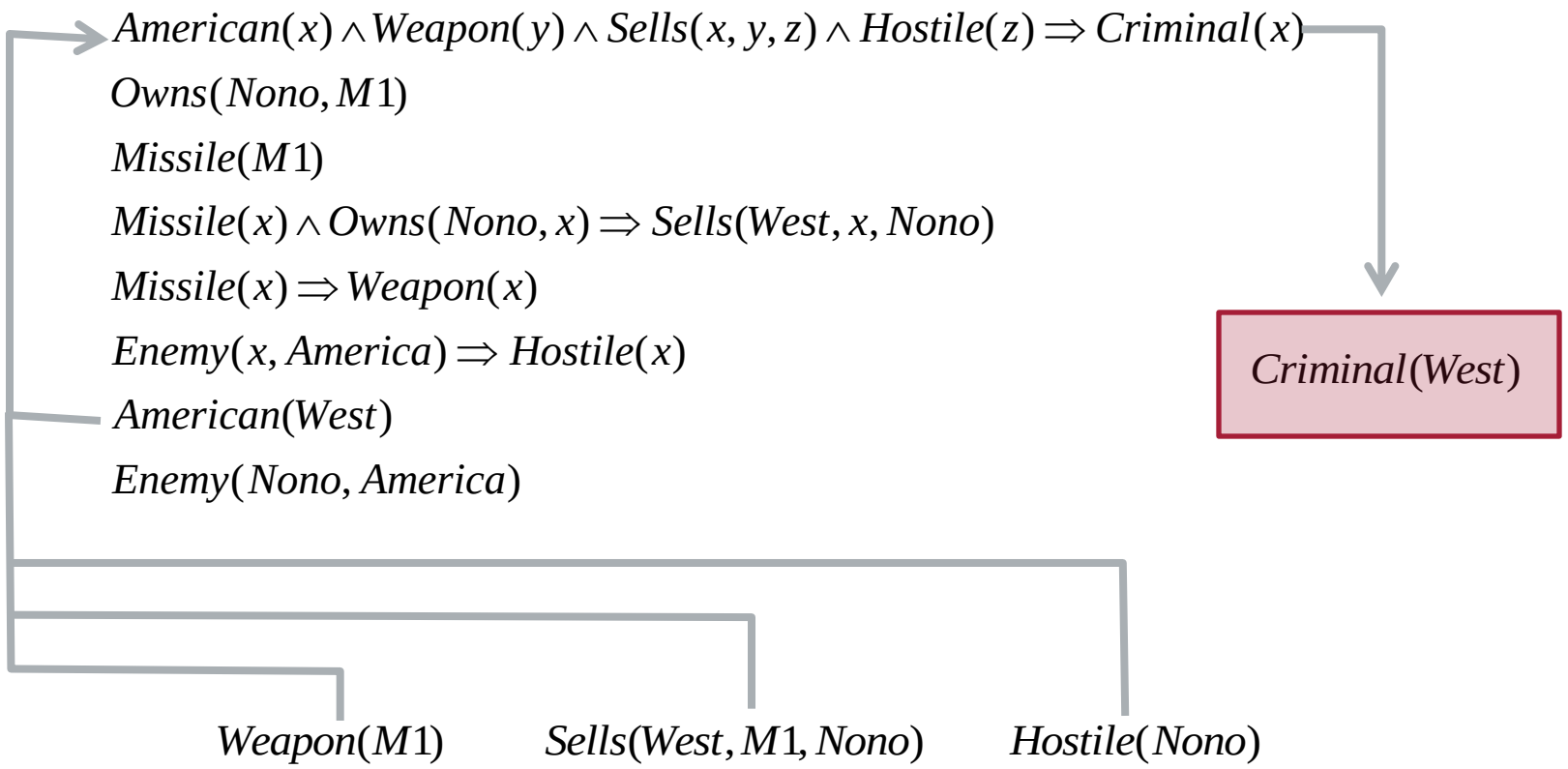
$Enemy(Nono, America)$



$Weapon(M1)$ $Sells(West, M1, Nono)$ $Hostile(Nono)$

$Criminal(West)$

9.3 Forward Chaining
Example: Prove West is Criminal



9.3 Forward Chaining
Forward Chaining Inefficiencies

EBERHARD KARLS UNIVERSITÄT TÜBINGEN

1st Inefficiency: inner loop requires finding all unifiers that match to the premises of a rule

- Pattern Matching: find all unifiers for a rule
  - Conjunct ordering: finding best order to match premises
    - Each conjunct is a constraint, so pattern matching is a CSP
    - Any finite-domain CSP can be expressed as a single definite clause together with associated ground facts
    - Matching can thus use heuristic to find ordering
  - Matching definite clause to facts is NP-hard but,
    - Data complexity: complexity of inference as a function of number of ground facts
      - data complexity of forward chaining is polynomial
    - We can consider sets of rules for which matching is efficient
    - We can eliminate redundant rule matching attempts

9.3 Forward Chaining
Forward Chaining Inefficiencies

EBERHARD KARLS UNIVERSITÄT TÜBINGEN

2nd Inefficiency: forward chaining rechecks every rule in every iteration

- Preventing Redundant Checking of rules
  - *Every new fact inferred on iteration t must be derived from at least one new fact inferred on iteration t-1*
  - Indexing identifies only the rules that a fact can trigger
  - We can also retain & complete partial matches ➔ avoid recomputation by incremental forward chaining
  - The *rete algorithm* uses this process to make efficient inference by constructing a dataflow network where each node is a literal from a rule premise… this network captures all partial matches

Forward Chaining Inefficiencies

3rd Inefficiency: many facts are irrelevant to goal

- Preventing facts irrelevant to the goal
  1. Backward chaining is an alternative
  2. Restrict forward chaining to a selected subset of rules
  3. Rewrite rule set using info from the goal, so that only relevant variable bindings, the magic set, are considered
     - Predicate Magic(…) added to rules that can derive goal
     - Ex: Add fact Magic(West) to our KB & make rule now

$$Magic(x) \wedge American(x) \wedge Weapon(x) \wedge Sells(x, y, z) \wedge Hostile(z) \Rightarrow Criminal(x)$$

     - This avoids exploring many irrelevant matches to the rule
     - Computing the magic set is a hybrid of forward & backward inference

9.4 Backward Chaining

EBERHARD KARLS UNIVERSITÄT TÜBINGEN

- Backward Chaining Algorithm: works backwards from goal, matching the effects of rules to support desired proof based on *Modus Ponens*.
  - It is implemented as generator: a function that returns multiple times, each time with one possible result
  - Acts like an AND/OR search:
    - OR search: the goal can be proved by any rule in the KB
    - AND search: all the conjuncts in the rule's body must be proved to use the rule
  - It is a DFS search ➔ linear space complexity
  - Problems: redundant states & incompleteness

9.4 Backward Chaining

- Algorithm outline:
  - Goals are kept in stacks; 1 stack for each branch of proof
  - If all goals are satisfied by initial state: success
  - Goals are popped off the stack and if unsatisfied:
    - Every clause whose *head* unifies with the goal makes a separate branch of the proof.
    - The *body* of the unified clause is added to the stack

9.4 Backward Chaining
Backward Chaining Algorithm

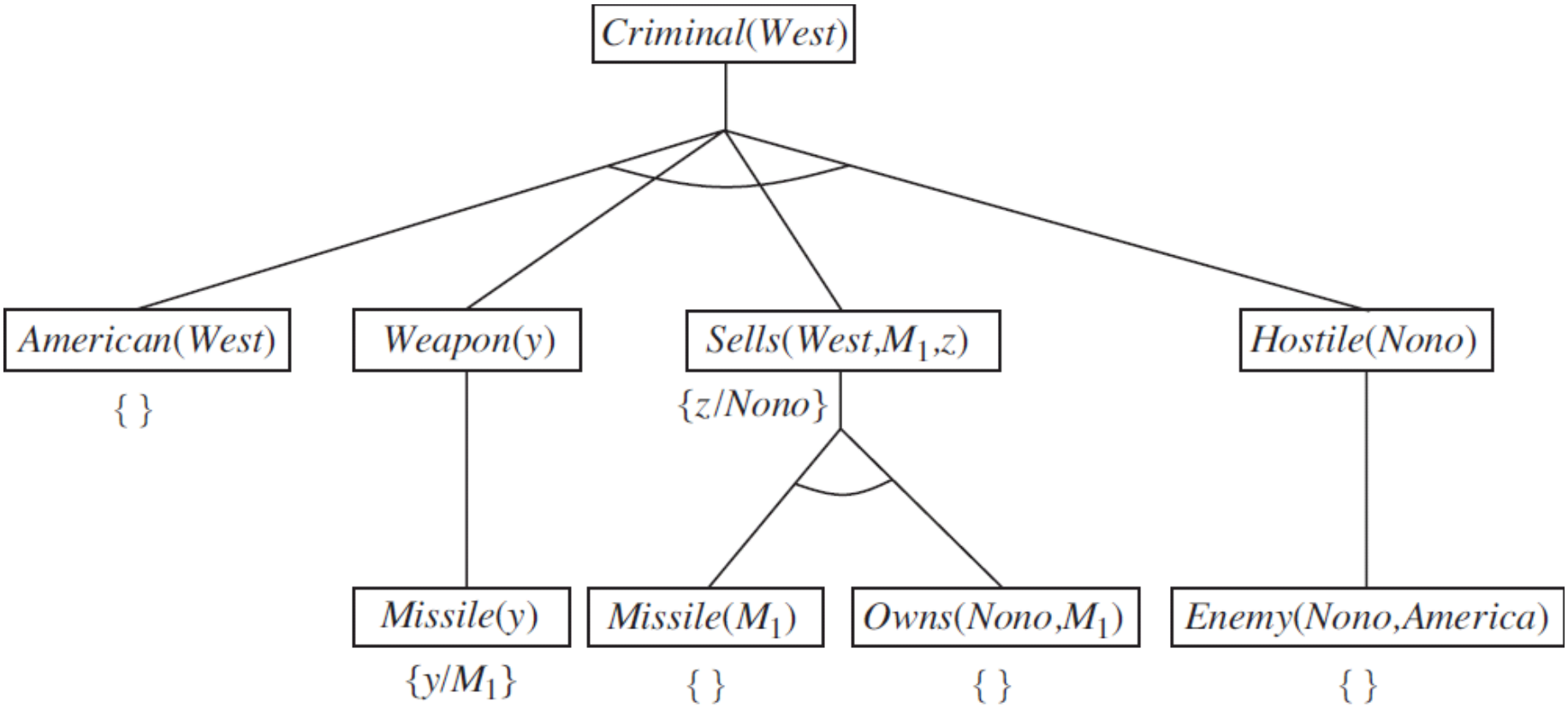
EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

function FOL-BC-ASK(KB, $query$) **returns** a generator of substitutions
  return FOL-BC-OR(KB, $query$, $\{\}$)

generator FOL-BC-OR(KB, $goal$, θ) **yields** a substitution
  for each rule $(lhs \Rightarrow rhs)$ in FETCH-RULES-FOR-GOAL(KB, $goal$) **do**
    $(lhs, rhs) \leftarrow$ STANDARDIZE-VARIABLES((lhs, rhs))
    for each θ' **in** FOL-BC-AND(KB, lhs, UNIFY(rhs, $goal$, θ)) **do**
      yield θ'

generator FOL-BC-AND(KB, $goals$, θ) **yields** a substitution
  if $\theta = failure$ **then return**
  else if LENGTH($goals$) $= 0$ **then yield** θ
  else do
    $first, rest \leftarrow$ FIRST($goals$), REST($goals$)
    for each θ' **in** FOL-BC-OR(KB, SUBST(θ, $first$), θ) **do**
      for each θ'' **in** FOL-BC-AND(KB, $rest$, θ') **do**
        yield θ''

9.3 Backward Chaining
Example: Prove West is Criminal



9.4 Backward Chaining
Logic Programming

EBERHARD KARLS
UNIVERSITÄT
TÜBINGEN

- Logic Programming: declarative language expresses logic & solves problems via inference
  - Kowalski's Equation: *Algorithm* = *Logic* + *Control*
- Prolog is the most-widely used logic programming language
  - Also used for rapid-prototyping, symbolic-manipulation, & parsing natural language
  - Many expert systems are written in Prolog

9.4 Backward Chaining
Logic Programming

EBERHARD KARLS UNIVERSITÄT TÜBINGEN

- Prolog: logic PL with depth-first backward chaining
  - Programs are sets of definite clauses
  - Definite clause syntax: $A \wedge B \Rightarrow C$ written C :- A , B .
  - List syntax: [E | L] denotes list with 1st element E & rest L
  - Built-in arithmetic functions
  - Some predicates (ex: input/output) have side-effects
  - Database Semantics:
    - Unique Names: every constant & ground term is unique obj.
    - Closed World: only true sentences are those entailed by KB
      - Not possible to assert that a sentence is false
    - Completion: in FOL, a sentence to add unique names or closed world assumption or other domain-specific info to the KB
  - Occur check is omitted from unification, so unification is unsound
  - Prolog can be compiled or interpreted

9.4 Backward Chaining
Logic Programming: Example

EBERHARD KARLS UNIVERSITÄT TÜBINGEN

- Append relation takes 3 arguments & succeeds if the 1st two arguments are appended to make the 3rd
  - Prolog program for *append*:

$$append([], Y, Y) \quad .$$
$$append([A|X], Y, [A|Z]) \quad :- \quad append(X, Y, Z) \quad .$$

  - This is not only a function to append lists, it is a relationship between 3 arguments.
    - We can ask questions like: $append([], X, [A|X])$
    - We can also ask questions like what 2 lists will generate [1,2]: $append(X, Y, [1,2])$ generates $X = []$ $Y = [1,2]$;
      $X = [1]$ $Y = [2]$;
      $Y = [1,2]$ $Y = []$;

9.4 Backward Chaining
Efficient Logic Programs

- choice point: structure for remembering previous choices in the DFS for FOL-BC-ASK; a global stack
  - Continuation: a choice point that contains a procedure call that defines what to do when goal succeeds
- Every variable is either unbound or bound to a value
  - Path extended by binding but must unbind on backtracking
  - Trail: list of variables that have been bound on the stack
- OR-parallelism: possibility of goal unifying with any clause leads to solving each branch in parallel
- AND-parallelism: possibility parallelism in solving each conjunct in the body of an implication
  - Problem: each conjunctive branch must communicate bindings made with other branches to be consistent

9.4 Backward Chaining
Problems with Backward Chaining

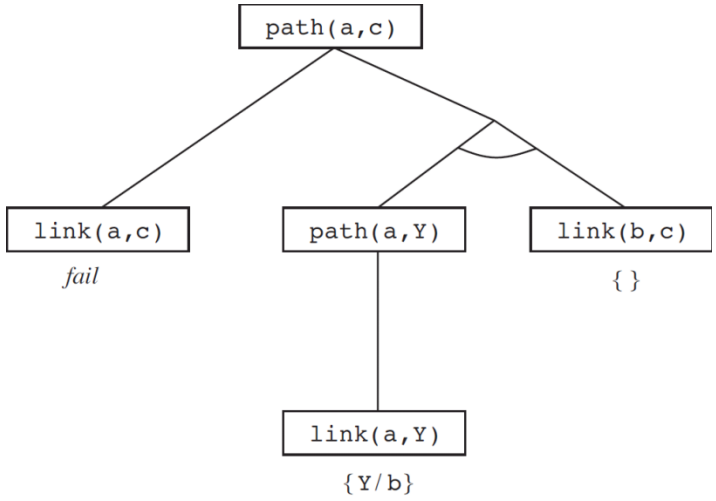
- Infinite Paths: possibility of infinite paths makes backward chaining incomplete
  - Ex. Program defines connectivity 2 nodes in a DAG

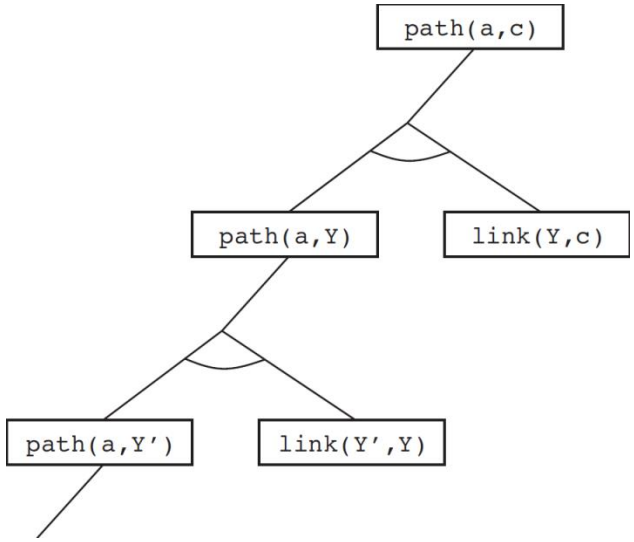
$path(X,Z) :- link(X,Z).$
$path(X,Z) :- path(X,Y), path(Y,Z).$

OR

$path(X,Z) :- path(X,Y), path(Y,Z).$
$path(X,Z) :- link(X,Z).$

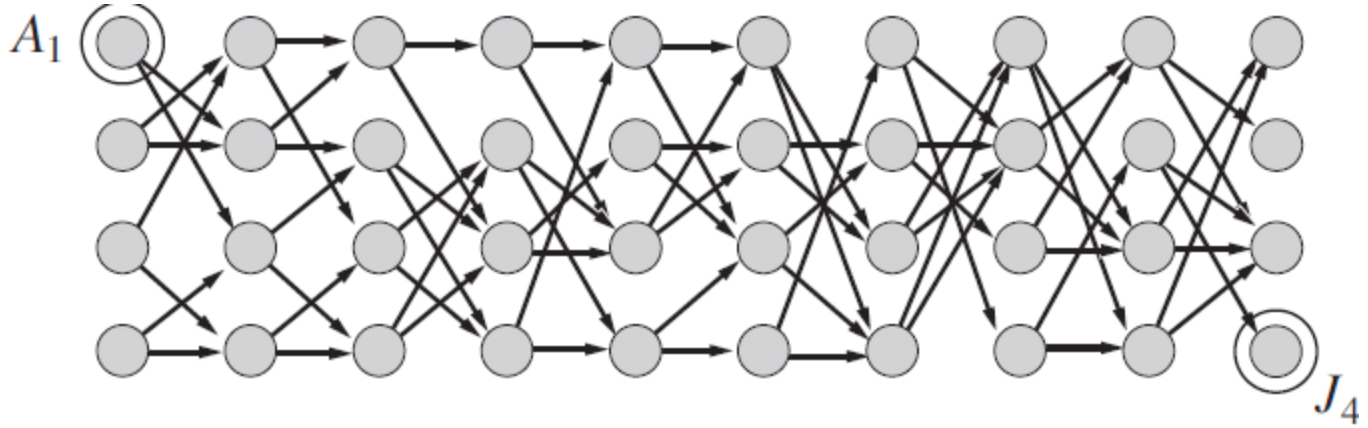
  - The 2nd way generates infinite paths: incomplete





9.4 Backward Chaining
Problems with Backward Chaining

- Repeated States*:* inference can be exponential in number of ground facts
  - Ex: In following graph Prolog uses 877 inferences to find path from A_1 to J_4; forward chaining only uses 62:



  - FC acts like dynamic programming for these graph problems: also possible for BC using memoization
  - memoization: caching solutions to subgoals & reusing the solutions if they reoccur (tabled logic programming)

9.4 Backward Chaining Constraint Logic Programming

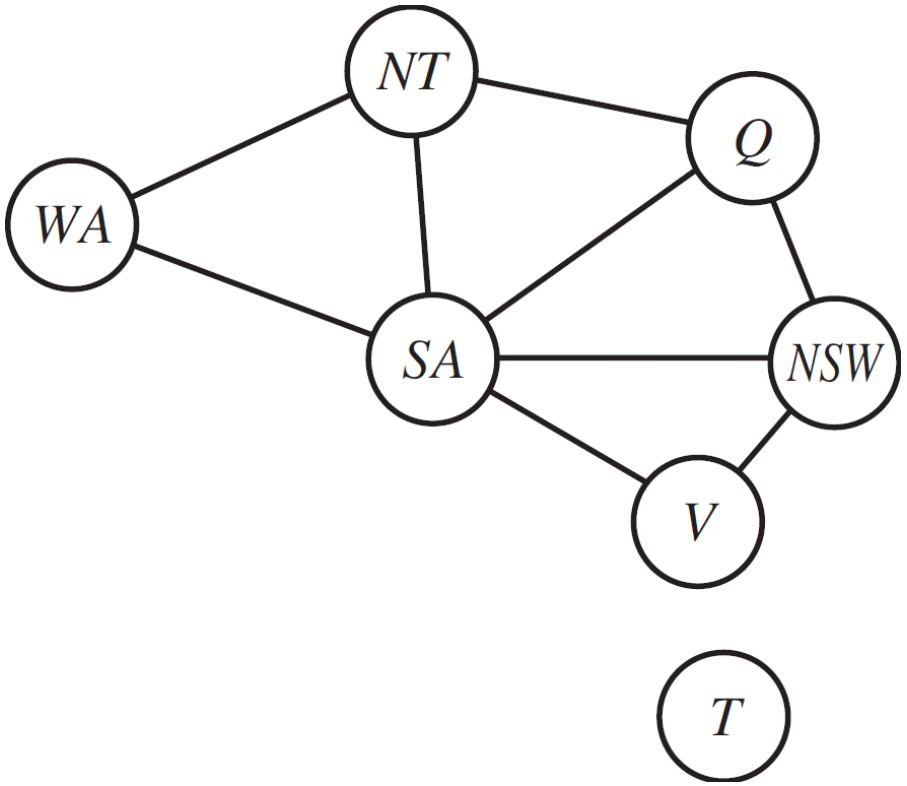
EBERHARD KARLS UNIVERSITÄT TÜBINGEN

- Constraint Logic Programming: CSP's can be encoded as definite clauses
  - Backward chaining can be used for finite domain CSPs
    - Ex: consider triangle inequality

      $$triangle(X,Y,Z) \;:-\; X \geq 0,\quad Y \geq 0,\quad Z \geq 0,\quad X+Y \geq Z,\quad Y+Z \geq X,\quad X+Z \geq Y.$$

    - Prolog can't answer $triangle(3,4,Z)$ because of subgoal $Z \geq 0$
  - Idea: allow variables to be constrained (restrict values) rather than bound (fixed to a single value)
    - Standard logic programs are special case with only equality
    - Ex: answer to $triangle(3,4,Z)$ is constraint $7 \geq Z \geq 1$
    - CLP incorporate special constraint-solving algorithms for their allowable constraints; ex: linear programming

- Example: Australia Coloring



$$Diff(WA, NT) \wedge Diff(WA, SA) \wedge$$
$$Diff(NT, Q) \wedge Diff(NT, SA) \wedge$$
$$Diff(Q, NSW) \wedge Diff(Q, SA) \wedge$$
$$Diff(NSW, V) \wedge Diff(NSW, SA) \wedge$$
$$Diff(V, SA) \Rightarrow Colorable()$$
$$Diff(red, blue) \quad Diff(red, green)$$
$$Diff(green, red) \quad Diff(green, blue)$$
$$Diff(blue, red) \quad Diff(blue, green)$$

9.5 Resolution

EBERHARD KARLS UNIVERSITÄT TÜBINGEN

- Generalized Resolution: an extension of propositional resolution to FOL
- Proof by contradiction: *negated goal is added to KB. If {} is derived, the goal is proved*
  - These proofs are *non-constructive*: they only show if the query is true or false. Alternatives:
    - Restrict query variables to a single binding & backtrack
    - Add answer literal as a disjunction with the negated goal. The resulting proof will have a disjunction of possible answers instead of an empty clause → multiple answers.

Reduction to Propositional Inference

- Conjunctive Normal Form – conjunction of clauses; each clause is a disjunction of literals.
  - *Every sentence in FOL can be converted into an inferentially equivalent CNF sentence.*
  - Conversion to CNF:
    1. Eliminate implications using: $p \Rightarrow q \equiv \neg p \vee q$
    2. Move $\neg$ inwards in quantified statements
    3. Standardize Variables: eliminate reused names
    4. Skolemization: removal of existential quantifiers with Skolem functions (within universal quantifier) or Skolem constants
    5. Drop universal quantifiers
    6. Distribute $\wedge$ over $\vee$

9.5 Resolution
Example of CNF Conversion

EBERHARD KARLS UNIVERSITÄT TÜBINGEN

- Example: "Everyone who loves all animals is loved by somebody"

$$\forall x \quad [\forall y \quad Animal(y) \Rightarrow Loves(x,y)] \Rightarrow [\exists y \quad Loves(y,x)]$$

Eliminate implications: $\forall x \quad \neg[\forall y \quad \neg Animal(y) \vee Loves(x,y)] \vee [\exists y \quad Loves(y,x)]$

Move $\neg$ inwards: $\forall x \quad [\exists y \quad Animal(y) \wedge \neg Loves(x,y)] \vee [\exists y \quad Loves(y,x)]$

Standardize variables: $\forall x \quad [\exists y \quad Animal(y) \wedge \neg Loves(x,y)] \vee [\exists z \quad Loves(z,x)]$

Skolemization: $\forall x \quad [Animal(F(x)) \wedge \neg Loves(x,F(x))] \vee [Loves(G(x),x)]$

Drop Universal Quantifiers: $[Animal(F(x)) \wedge \neg Loves(x,F(x))] \vee [Loves(G(x),x)]$

Distribute $\wedge$ over $\vee$:

$$[Animal(F(x)) \vee Loves(G(x),x)] \wedge [\neg Loves(x,F(x)) \vee Loves(G(x),x)]$$

9.5 Resolution
The Resolution Inference Rule

- Propositional literals are complementary if one is the negation of the other
- FOL literals if one unifies with negation of other
- Binary First-Order Resolution when $\text{UNIFY}(\ell_i, \neg m_j) = \theta$

$$\frac{\ell_1 \vee \ldots \vee \ell_k \qquad m_1 \vee \ldots \vee m_n}{\text{SUBST}(\theta, \ell_1 \vee \ldots \vee \ell_{i-1} \vee \ell_{i+1} \vee \ldots \vee \ell_k \vee m_1 \vee \ldots \vee m_{j-1} \vee m_{j+1} \vee \ldots \vee m_n)}$$

- Ex: Consider the following clauses:
  - $Animal(F(x)) \vee Loves(G(x),x)$
  - $\neg\, Loves(u,v) \vee \neg\, Kills(u,v)$
  - $Loves(G(x),x)$ and $\neg\, Loves(u,v)$ have unifier $\theta = \{u/G(x), v/x\}$
  - Resolvent clause: $Animal(F(x)) \vee \neg\, Kills(G(x),x)$

9.5 Resolution: Factoring

EBERHARD KARLS UNIVERSITÄT TÜBINGEN

- Binary resolution only resolves a pair of complementary literals: not complete
- Full Resolution resolves subsets of literals
- Alternative: use binary resolution with factoring
- Factoring: removal of redundant literals
  - Propositional factoring: reduce identical literals
  - First-order factoring: reduce literals if they are unifiable, then apply the unifier to the entire clause
- Together, the *binary resolution rule* and *factoring* are complete

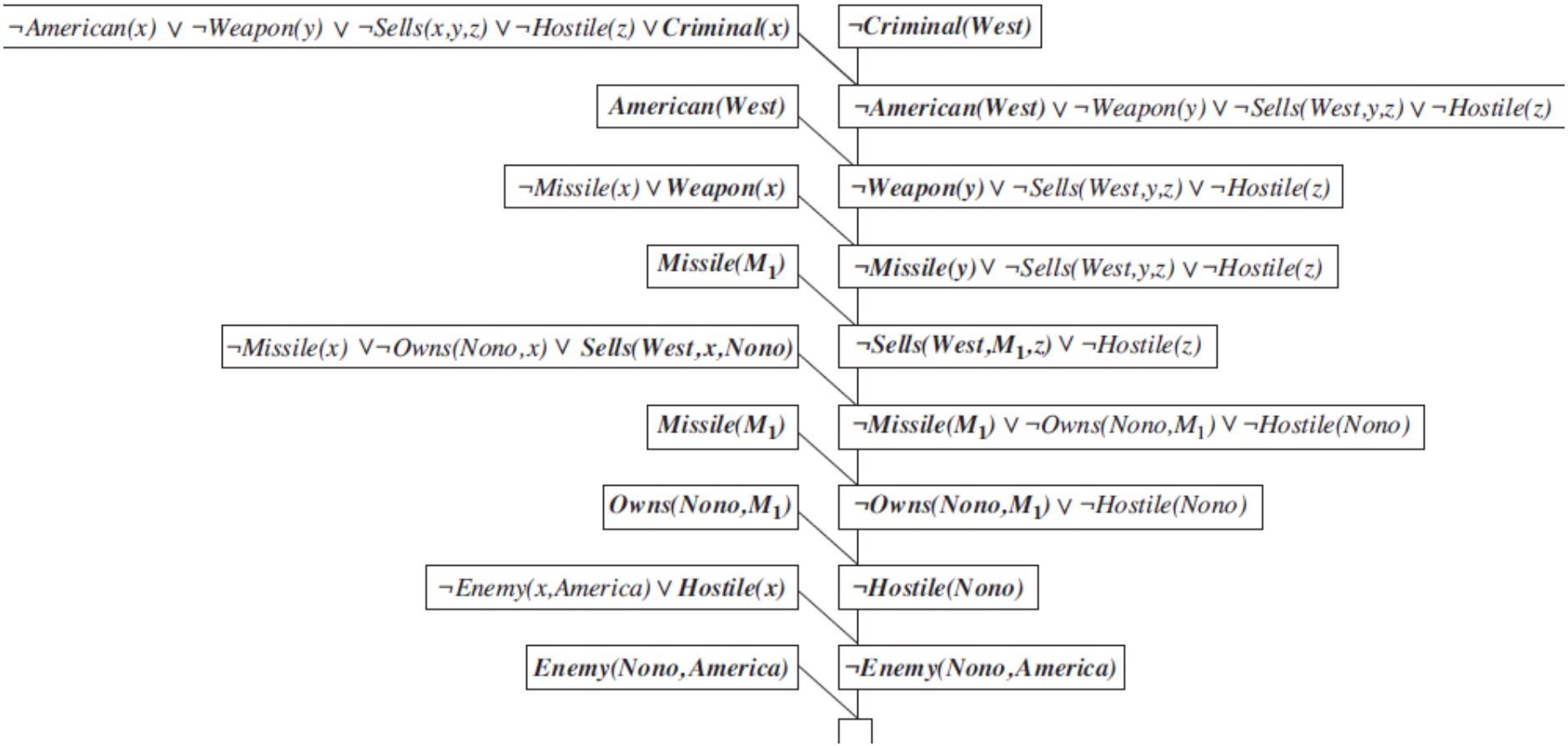
First-order Resolution Procedure

- Algorithm outline:
  1. Begin with propositionalization & conversion to CNF
  2. Apply the binary Resolution Inference Rule
     - The binary resolution rule can be applied to pairs of clauses (assumingly standardized apart) if they contain complementary literals.
     - *First-order literals are complementary if one unifies with the negation of the other.*
  3. Use factoring to unify redundant literals

9.5 Resolution
Example: Prove West is Criminal

EBERHARD KARLS UNIVERSITÄT TÜBINGEN

- Resolution proof that Col. West is a criminal



Example: Curiosity killed the Cat

- Example
  - Everyone who loves all animals is loved by someone
  - Anyone who kills an animal is loved by no one
  - Jack loves all animals
  - Either Jack or Curiosity killed the cat, Tuna

$A.\quad \forall x \quad [\forall y \quad Animal(y) \Rightarrow Loves(x,y)] \Rightarrow [\exists y \quad Loves(y,x)]$

$B.\quad \forall x \quad [\exists z \quad Animal(z) \Rightarrow Kill(x,z)] \Rightarrow [\forall y \quad \neg Loves(y,x)]$

$C.\quad \forall x \quad Animal(x) \Rightarrow Loves(Jack,x)$

$D.\quad Kills(Jack,Tuna) \vee Kills(Curiosity,Tuna)$

$E.\quad Cat(Tuna)$

$F.\quad \forall x \quad Cat(x) \Rightarrow Animal(x)$

  - Did Curiosity kill Tuna? Add $G.\quad \neg Kills(Curiosity,Tuna)$

Example: Curiosity killed the Cat

- Converting each sentence to CNF yields:

$A_1.\quad Animal(F(x)) \vee Loves(G(x), x)$

$A_2.\quad \neg Loves(x, F(x)) \vee Loves(G(x), x)$

$B.\quad \neg Loves(y, x) \vee \neg Animal(z) \vee \neg Kill(x, z)$

$C.\quad \neg Animal(x) \vee Loves(Jack, x)$

$D.\quad Kills(Jack, Tuna) \vee Kills(Curiosity, Tuna)$

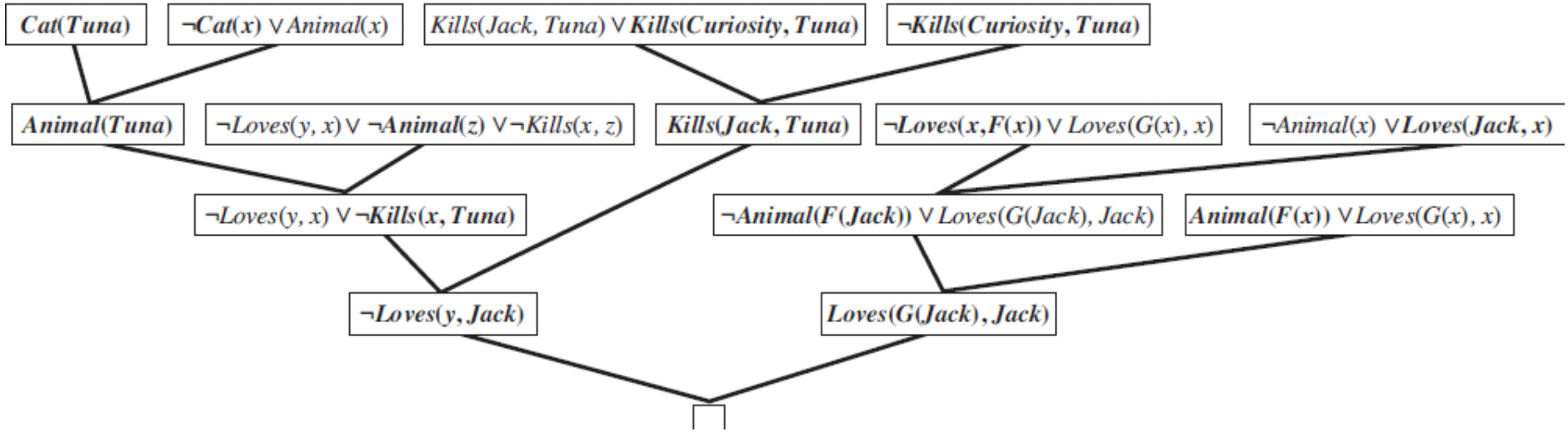
$E.\quad Cat(Tuna)$

$F.\quad \neg Cat(x) \vee Animal(x)$

$G.\quad \neg Kills(Curiosity, Tuna)$

9.5 Resolution
Example: Curiosity killed the Cat

- Resolution proof that curiosity killed the cat



- Proof (in English): Suppose Curiosity did not kill Tuna. We know either Jack or Curiosity did; thus, Jack must have. Further, Tuna is a cat & cats are animals, so Tuna is an animal. Because anybody who kills animals is loved by no one, nobody loves Jack. However, Jack also loves all animals and thus is loved by somebody, which is a contradiction. Thus, Curiosity killed Tuna.

9.5 Resolution: Equality

- Handling equality *x*=*y* in inference: 3 approaches
- #1 Axiomatic Equality: add axioms for reflexivity, symmetry & transitivity
  - We need 3 basic axioms & 1 for every predicate & function

$$\forall x, \quad x = x$$

$$\forall x, y \quad x = y \Rightarrow y = x$$

$$\forall x, y, z \quad x = y \wedge y = z \Rightarrow x = z$$

$$\forall x, y \quad x = y \Rightarrow \left(P_1(x) \Leftrightarrow P_1(y)\right)$$

$$\ldots$$

$$\forall w, x, y, z \quad \left(w = y \wedge x = z\right) \Rightarrow \left(F_1(w, x) = F_1(y, z)\right)$$

  - This approach generates many spurious conclusions

9.5 Resolution: Equality

- #2 Demodulation: for any terms x, y and z, where $\text{UNIFY}(x, z) = \theta$ and m_i is a literal containing z

$$\frac{x = y \quad m_1 \vee \ldots \vee m_n}{\text{SUB}(\text{SUBST}(\theta, x), \text{SUBST}(\theta, y), m_1 \vee \ldots \vee m_n)}$$

  - Where $\text{SUB}(a,b,c)$ replaces a with b where a occurs in c
  - Demodulation is directional: for $x=y$ always replace x with y and never y with x
- Paramodulation: for any terms x, y and z, where $\text{UNIFY}(x, z) = \theta$ and m_i is a literal containing z

$$\frac{\ell_1 \vee \ldots \vee \ell_k \vee x = y \quad m_1 \vee \ldots \vee m_n}{\text{SUB}(\text{SUBST}(\theta, x), \text{SUBST}(\theta, y), \mathit{SUBST}(\theta, \ell_1 \vee \ldots \vee \ell_k \vee m_1 \vee \ldots \vee m_n))}$$

  - Inference with equality is complete with paramodulation

- #3 Equational Unification: unification designed for the particular axiom used rather than explicit inference with those axioms
  - Under this approach, terms are unifiable if they are “provably” equal under some substitution
  - Proving equality allows for equality reasoning
- Theorem Provers that use this technique are related to CLP systems

9.5 Resolution
Strategies for Efficient Resolution

- Unit preference: prefer resolution with unit clauses; always makes the resolvents shorter
  - unit clause: a sentence of a single literal.
  - unit resolution: resolution only with unit clauses… Incomplete in general, but complete for Horn KBs
- Set of Support: goal-driven strategy in which resolution is only done with a subset of sentences called the *set of support*
  - Results of the resolution are added to the support set
  - Complete if *set of support* chosen so that the remaining sentences are jointly satisfiable

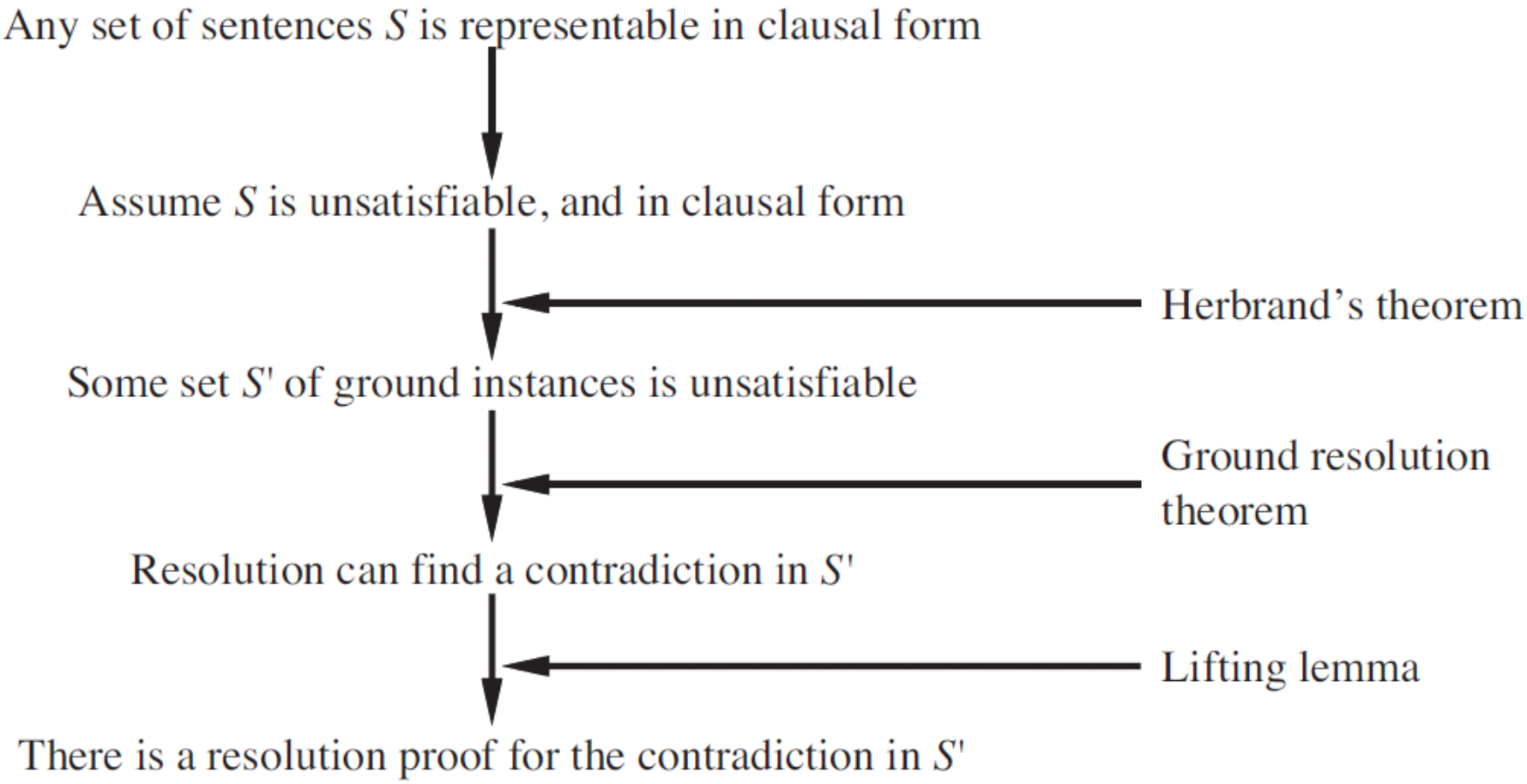
9.5 Resolution
Strategies for Efficient Resolution

- Input Resolution: every resolution combines an input sentence (KB and query) with some other sentence; ex: Modus Ponens
  - complete for Horn KBs but incomplete in general
- Linear Resolution: allows *P* and *Q* to be resolved together either if *P* is in the original KB or if *P* is an ancestor of *Q* in the proof tree
  - complete, in general
- Subsumption: elimination of all subsumed sentences from the KB to maintain small KB
  - sentence *a* is subsumed by sentence *b* if *a* is more specific than *b*

Completeness of Resolution

- Completeness of Resolution: If S is an unsatisfiable set of clauses, then the application of a finite number of resolution steps to S will yield a contradiction; that is, resolution is refutation complete
  - Refutation complete: if S is unsatisfiable, a finite number of resolution rules will yield a contradiction

Structure of the Proof:



9.5 Resolution
Hebrand's Theorem

EBERHARD KARLS UNIVERSITÄT TÜBINGEN

- Herbrand Universe H_S : the set of all *ground terms* constructible from the symbols in set of clauses S
- Saturation $P(S)$: set of all ground clauses obtained from consistent substitutions of terms in P, a set of ground terms, applied to variables in S
- Herbrand base $H_S(S)$: The saturation of set S of clauses with respect to its Herbrand Universe.
- Herbrand's Theorem: If a set S of clauses is unsatisfiable, then there exists a finite subset S' of $H_S(S)$ that is also unsatisfiable

Ground Resolution Revisited

- Ground Resolution Theorem: as presented in Chapter 7, propositional resolution is complete for ground sentences; hence, the resolution closure $RC(S')$ will contain the empty clause: a contradiction

9.5 Resolution: Lifting Lemma

EBERHARD KARLS UNIVERSITÄT TÜBINGEN

- Lifting Lemma: Let C_1 and C_2 be clauses with no shared variables, and let C_1' and C_2' be ground instances of C_1 and C_2. If C' is resolvent of C_1' and C_2', then there exists a clause C such that
  - C is resolvent of C_1 and C_2
  - C' is a ground instance of C
- From this, if $\{\}$ appears in $RC(S')$, it must also appear in $RC(S)$ since $\{\}$ cannot be the ground instance of any other clause
- This completes the proof to show that resolution is refutation complete