人工智能

中国科学院计算技术研究所
Institute Of Computing Technology Chinese Academy Of Sciences

罗平 luop@ict.ac.cn

Knowledge 4

First-Order Logic: Inference

A brief history of reasoning

450 B.C.	Stoics	propositional logic, inference (maybe)
322 B.C.	Aristotle	"syllogisms" (inference rules), quantifiers
1565	Cardano	probability theory (propositional logic + uncertainty)
1847	Boole	propositional logic (again)
1879	Frege	first-order logic
1922	Wittgenstein	proof by truth tables
1930	Gödel	$\exists$ complete algorithm for FOL
1930	Herbrand	complete algorithm for FOL (reduce to propositional)
1931	Gödel	$\neg\exists$ complete algorithm for arithmetic
1960	Davis/Putnam	"practical" algorithm for propositional logic
1965	Robinson	"practical" algorithm for FOL—resolution

First-Order Logic: Inference

Resolution in FOL

Unification

- We can get the inference immediately if we can find a substitution θ such that $King(x)$ and $Greedy(x)$ match $King(John)$ and $Greedy(y)$
- $\theta = \{x/John\,, y/John\}$ works
- $UNIFY(\alpha, \beta) = \theta$ if $\alpha\theta\ equals\ to\ \beta\theta$

p	q	θ
$Knows(John, x)$	$Knows(John, Jane)$	$\{x/Jane\}$
$Knows(John, x)$	$Knows(y, OJ)$	$\{x/OJ, y/John\}$
$Knows(John, x)$	$Knows(y, Mother(y))$	$\{y/John, x/Mother(John)\}$
$Knows(John, x)$	$Knows(x, OJ)$	$fail$

Standardizing apart eliminates overlap of variables, e.*g.*, $Knows(z_{17}, OJ)$

Resolution: brief summary

- Full first-order version

$$\frac{l_1 \vee \cdots \vee l_k, \qquad m_1 \vee \cdots \vee m_n}{(l_1 \vee \cdots \vee l_{i-1} \vee l_{i+1} \vee \cdots \vee l_k \vee m_1 \vee \cdots \vee m_{j-1} \vee m_{j+1} \vee \cdots \vee m_n)\theta}$$

- Where $UNIFY\left(l_i, \neg m_j\right) = \theta$
- For example,

$$\frac{\neg Rich(x) \vee Unhappy(x)\ , \qquad Rich(Ken)}{Unhappy(Ken)}$$

- With $\theta = \{x/Ken\}$
- Apply resolution steps to CNF(KB$\wedge \neg\alpha$); complete for FOL

Conversion to CNF

- Everyone who loves animals is loved by someone:
  $\forall x\ \ [\forall y\ Animal(y) \Rightarrow Loves(x,y)] \Rightarrow [\exists y\ \ Loves(y,x)]$
- Eliminate biconditionals and implications $\forall x \neg [\forall y \neg Animal(y) \lor Loves(x,y)] \lor [\exists y\ \ Loves(y,x)]$
- Move $\neg$ inwards: $\neg\forall\ x,p\ \ \equiv \exists x\ \neg p,\ \neg\exists x,p\ \ \equiv \forall x\ \neg p$:

$$\forall x\ \ [\exists y\ \neg(\neg Animal(y) \lor Loves(x,y))] \lor [\exists y\ \ Loves(y,x)]$$
$$\forall x\ \ [\exists y\ \neg\neg Animal(y) \land \neg Loves(x,y))] \lor [\exists y\ \ Loves(y,x)]$$
$$\forall x\ \ [\exists y\ Animal(y) \land \neg Loves(x,y))] \lor [\exists y\ \ Loves(y,x)]$$

- Standardize variables: each quantifier should use a different one

$$\forall x\ \ [\exists y\ Animal(y) \land \neg Loves(x,y))] \lor [\exists z\ \ Loves(z,x)]$$

- Skolemize: a more general form of existential instantiation.
  Each existential variable is replaced by a Skolem function of the enclosing universally quantified variables:

$$\forall x\ \ [\ Animal(F(x)) \land \neg Loves(x,F(x)))] \lor [\ Loves(G(x),x)]$$

Conversion to CNF

- Drop universal quantifiers:

$$[\,Animal(F(x)) \land \neg Loves(x, F(x))] \lor [\,Loves(G(x), x)]$$

- Distribute $\land$ over $\lor$:

$$[\,Animal(F(x)) \lor Loves(G(x), x)] \land [\neg Loves(x, F(x)) \lor Loves(G(x), x)]$$

Example knowledge base

The law says that it is a crime for an American to sell weapons to hostile nations. The country Nono, an enemy of America, has some missiles, and all of its missiles were sold to it by Colonel West, who is American.

Prove that Col. West is a criminal

… it is a crime for an American to sell weapons to hostile nations:

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

Nono … has some missiles, i.e., $\exists x\ Owns(Nono, x) \wedge Missile(x)$:

$Owns(Nono, M_1)$ and $Missile(M_1)$

... all of its missiles were sold to it by Colonel West

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

Missiles are weapons:

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

An enemy of America counts as "hostile":

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

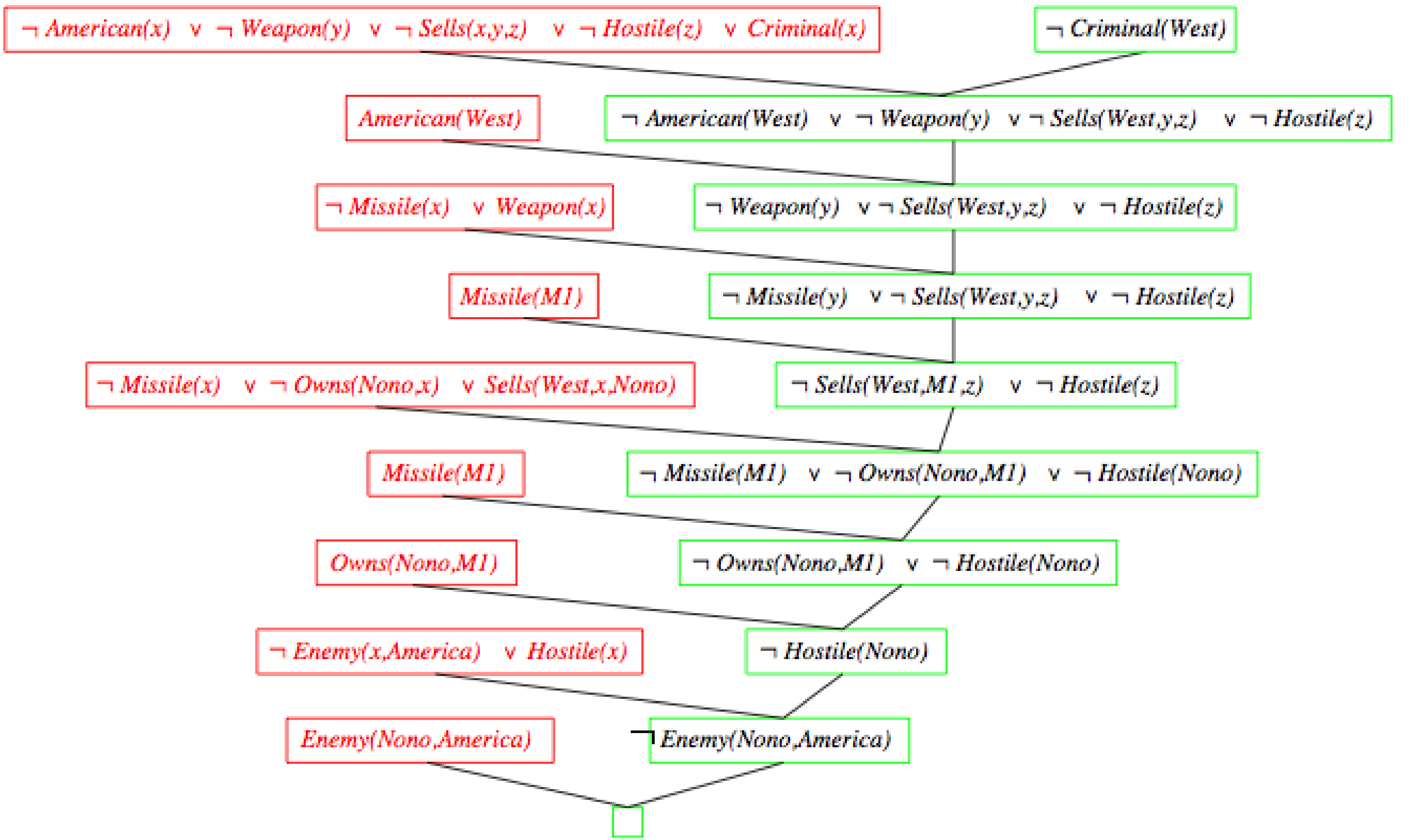
West, who is American …

$American(West)$

The country Nono, an enemy of America ...

$Enemy(Nono, America)$

Resolution proof



Completeness of FOL resolution

- Resolution is refutation-complete. If a set of sentences is unsatisfiable, resolution always derives a contradiction.
- It can find all answers of a given question, $Q(x)$, by proving that $KB \wedge \neg Q(x)$ is unsatisfiable
- Check out AIMA for the (brief) proof:

> If *S* is an unsatisfiable set of clauses, then the application of a finite number of resolution steps to *S* will yield a contradiction

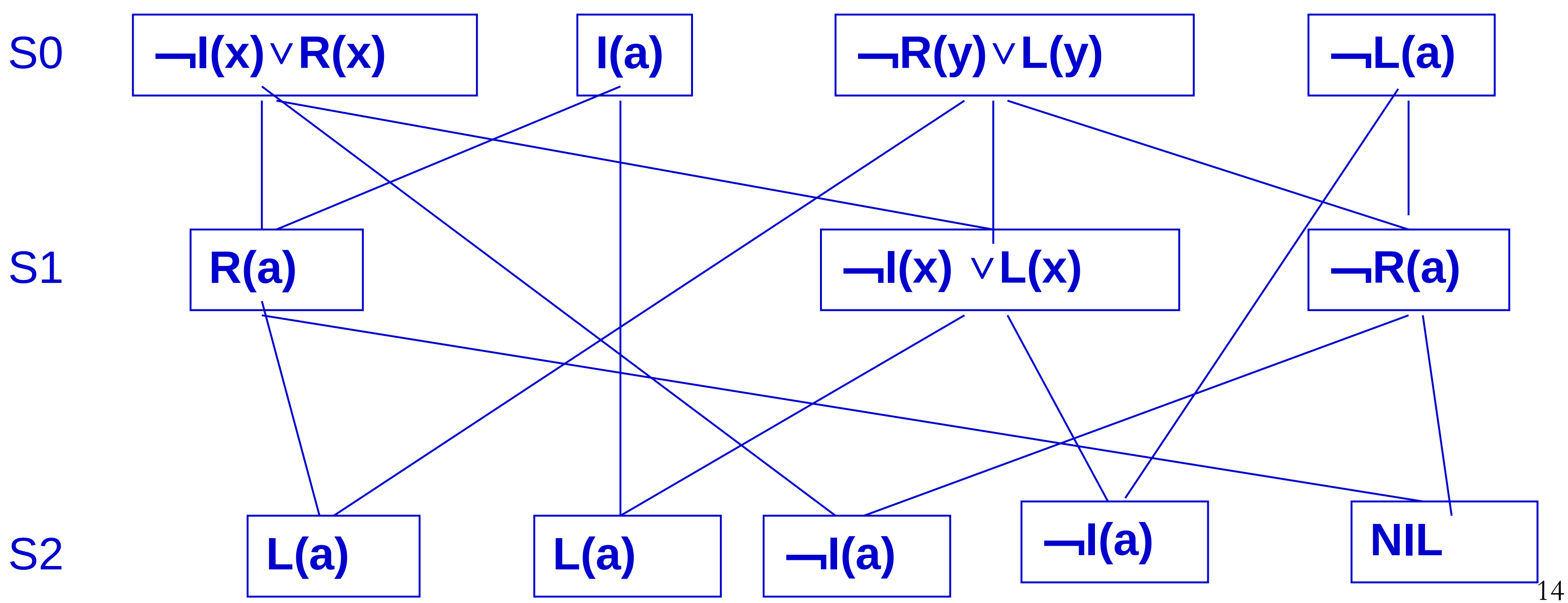
归结策略

删除和限制

归结策略：广度优先

- 策略定义：状态、目标状态
- 例设有如下子句集：S={¬I(x)∨R(x)，　I(a)，¬R(y)∨L(y)，¬L(a)}
- 用广度优先策略证明S为不可满足。

- 广度优先策略的归结树如下：



归结策略：广度优先

- **广度优先策略的优点：**
  - 当问题有解时保证能找到最短归结路径。
  - 是一种完备的归结策略。
- **广度优先策略的缺点：**
  - 归结出了许多无用的子句
  - 既浪费时间，又浪费空间

- **广度优先对大问题的归结容易产生组合爆炸，但对小问题却仍是一种比较好的归结策略。**

归结策略

- **常用的归结策略可分为两大类：**
  - **删除策略**是通过删除某些无用的子句来缩小归结范围
  - **限制策略**是通过对参加归结的子句进行某些限制，来减少归结的盲目性，以尽快得到空子句。

删除策略：删除纯文字

- **删除法主要想法是：把子句集中无用的子句删除掉，这就会缩小搜索范围，减少比较次数，从而提高归结效率。**
- **纯文字删除法**
  - 如果某文字L在子句集中不存在可与其互补的文字¬L，则称该文字为纯文字。
  - 在归结过程中，纯文字不可能被消除，用包含纯文字的子句进行归结也不可能得到空子句
  - 对子句集而言，删除包含纯文字的子句，是不影响其不可满足性的。例如，对子句集 S={P∨Q∨R，¬Q∨R，Q，¬R}，其中P是纯文字，因此可以将子句P∨Q∨R从子句集S中删除。

删除策略：删除重言式

- **重言式删除法**
  - 如果一个子句中包含有互补的文字对，则称该子句为重言式。

    例如P(x)∨¬P(x)，P(x)∨Q(x)∨¬P(x) 都是重言式，不管P(x)的真值为真还是为假，P(x)∨¬P(x)和P(x)∨Q(x)∨¬P(x)都均为真。
  - **重言式(valid sentences)是真值为真的子句。对一个子句集来说，不管是增加还是删除一个真值为真的子句，都不会影响该子句集的不可满足性。因此，可从子句集中删去重言式。**

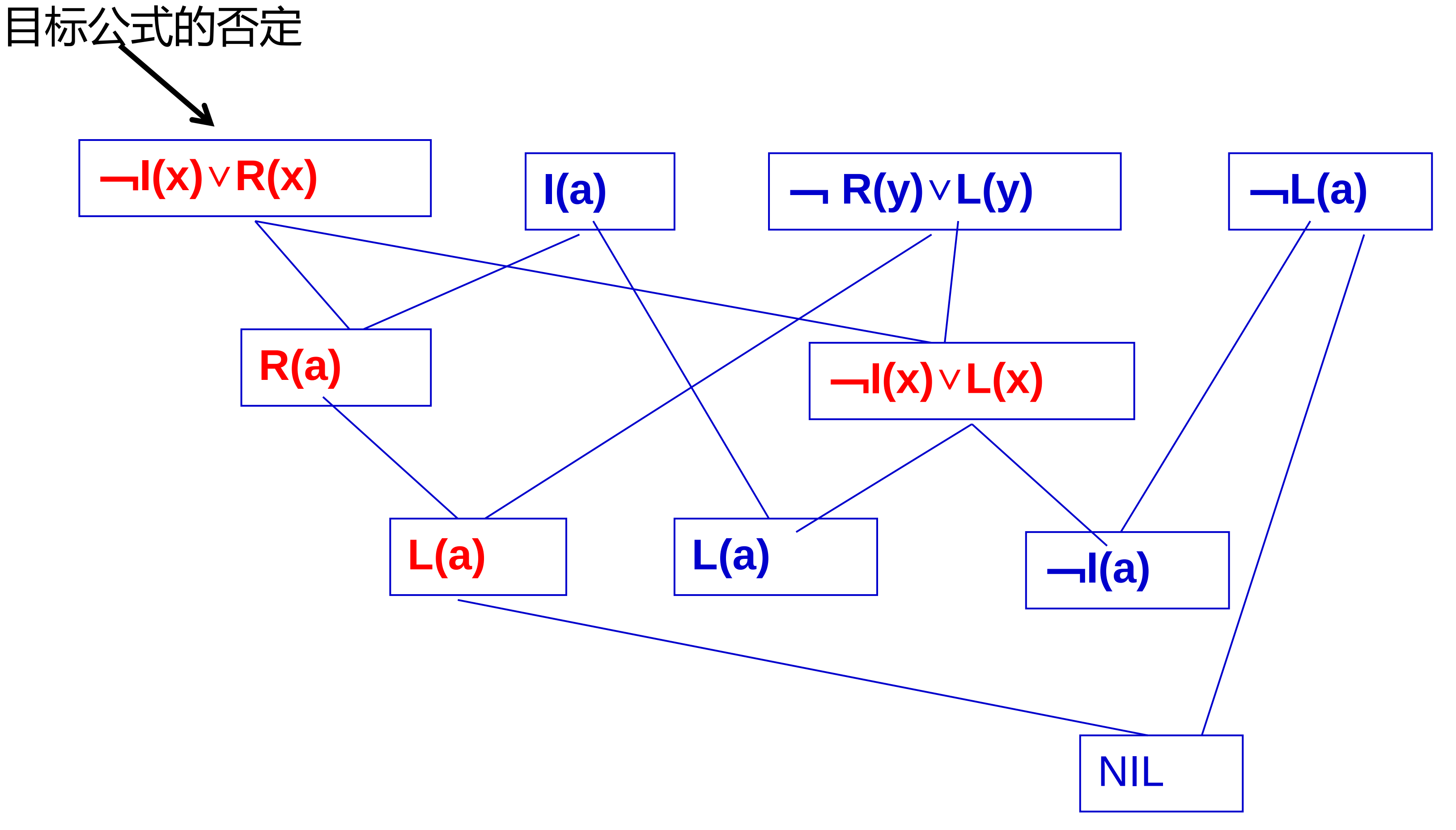
限制策略：支持集策略

- **支持集策略（Set of support）：**

  每一次参加归结的两个亲本子句中，至少应该有一个是由目标公式的否定所得到的子句或它们的后裔。

  - 支持集策略是**完备的**(?)，即当子句集为不可满足时，则由支持集策略一定能够归结出一个空子句。
  - 也可以把支持集策略看成是在广度优先策略中引入了**某种限制条件**，这种限制条件代表一种启发信息，因而有较高的效率

限制策略：支持集策略



限制策略：支持集策略

- 支持集策略**限制了子句集元素的剧增**，但会**增加空子句所在的深度（结果可能不是最优）**。
- 支持集策略具有**逆向推理的含义**，由于进行归结的亲本子句中至少有一个与目标子句有关，因此推理过程可以看作是沿目标、子目标的方向前进的。

限制策略：单文字子句策略

- 如果一个子句只包含一个文字，则称此子句为单文字子句。单文字子句策略是对支持集策略的进一步改进，它要求每次参加归结的两个亲本子句中至少有一个子句是单文字子句。

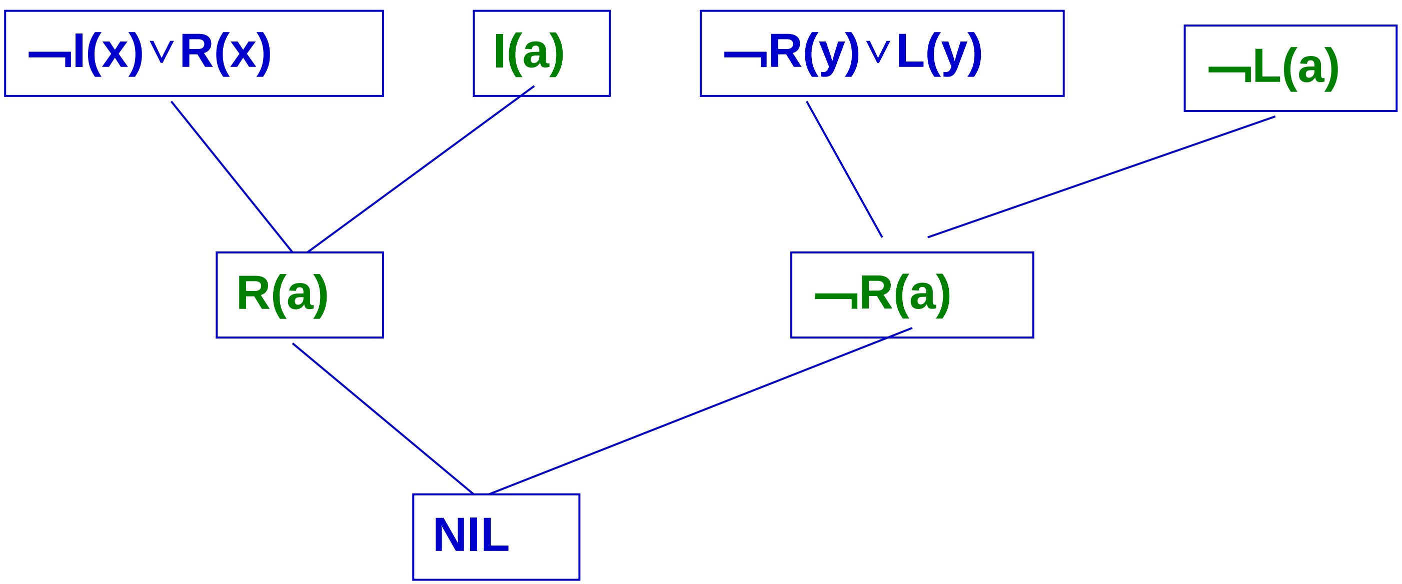
- **采用单文字子句策略，归结式包含的文字数将少于其非单文字亲本子句中的文字数，这将有利于向空子句的方向发展，因此会有较高的归结效率。**

限制策略：单文字子句策略

- 例: 设有如下子句集：

  S={¬I(x)∨R(x), I(a), ¬R(y)∨L(y), ¬L(a) }

- 用单文字子句策略证明S为不可满足。

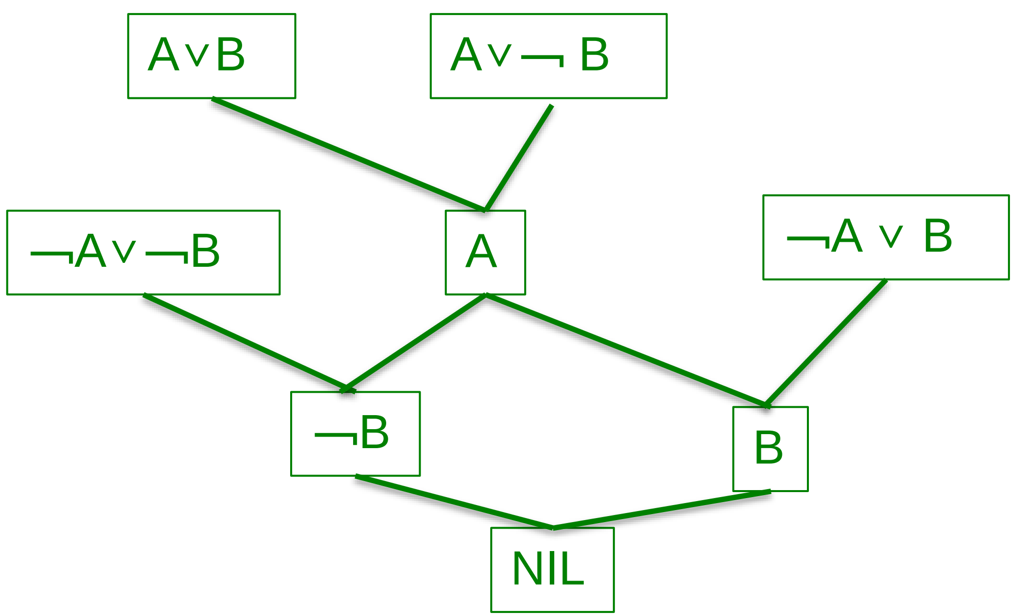


限制策略：单文字子句策略

- 单文字子句策略是不完备的，即当子句集为不可满足时，用这种策略不一定能归结出空子句。
- 原因：　没有可用的单文字字句

例如：已知：$A \vee B$，$A \vee \neg B$，$\neg A \vee B$，　求证：$A \wedge B$

化为字句集后为：　$A \vee B$，$A \vee \neg B$，$\neg A \vee B$，$\neg A \vee \neg B$，不存在单文字的字句。但是可以消解出空。



限制策略：祖先过滤策略

- **祖先过滤策略(Ancestry Filtering)**：满足以下两个条件中的任意一个就可进行归结：
  - 两个亲本子句中至少有一个是初始子句集中的子句。
  - 如果两个亲本子句都不是初始子句集中的子句，则一个子句应该是另一个子句的先辈子句。

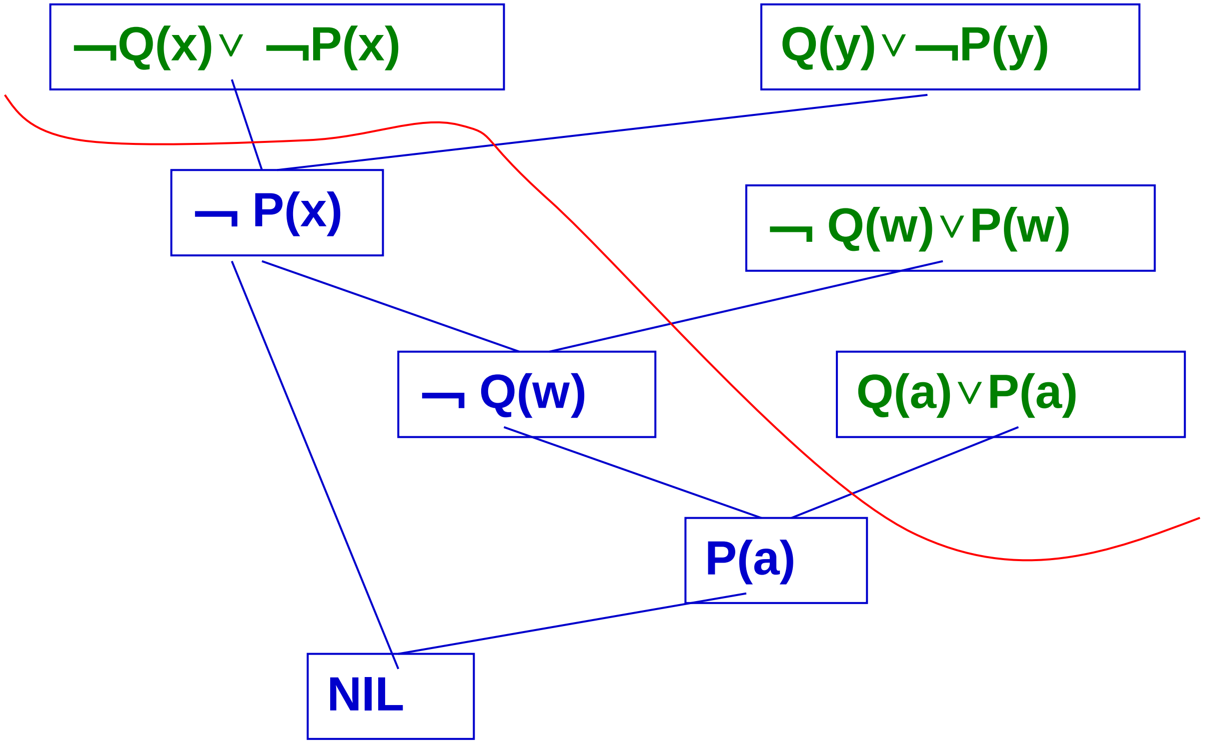
- **祖先过滤策略是完备的**

限制策略：祖先过滤策略

- 例：设有如下子句集：

  S={¬Q(x)∨¬P(x)，Q(y)∨¬P(y)，¬Q(w)∨P(w)，Q(a)∨P(a)}

- 用祖先过滤策略证明S为不可满足



First-Order Logic: Inference

Generalized Modus Ponens

Generalized Modus Ponens (GMP) 前见推理

$$\frac{p_1', p_2', \ldots, p_n', (p_1 \wedge p_2 \wedge \cdots \wedge p_n \Rightarrow q)}{q\theta}$$

where $p_i'\theta$ $equals\ to\ p_i\theta$ for all i

p_1' is $King(John)$ $\quad$ p_1 is $King(x)$

p_2' is $\text{Greedy}(y)$ $\quad$ p_2 is Greedy(x)

Θ is $\{x/John, y/John\}$ $\quad$ q is $Evil(x)$

$q\theta$ is $Evil(John)$

- GMP used with KB of definite clauses (exactly one positive literal)
  All variables assumed universally quantified

Soundness of GMP

- Need to show that

$$p_1', \dots, p_n', \qquad (p_1 \wedge \cdots \wedge p_n \Rightarrow q) \vDash q\theta$$

- Provided that $p_i'\theta = p_i\theta$ for all i
- Lemma: For any definite clause p, we have $p \vDash p\theta$ by UI
- 1. $(p_1 \wedge \cdots \wedge p_n \Rightarrow q) \vDash (p_1 \wedge \cdots \wedge p_n \Rightarrow q)\theta = (p_1\theta \wedge \cdots \wedge p_n\theta \Rightarrow q\theta)$
- 2. $p_1', \dots, p_n' \vDash p_1' \wedge \cdots \wedge p_n' \vDash p_1'\theta \wedge \cdots \wedge p_n'\theta$
- 3. From 1 and 2, $q\theta$ follows by ordinary Modus Ponens

the soundness of ordinary Modus Ponens

Forward chaining algorithm

function FOL-FC-ASK(KB, α) **returns** a substitution or *false*

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

Forward chaining proof

Facts

R_1 Owns(Nono, M_1)

R_2 Missile (M_1)

R_3 $American(West)$

R_4 $Enemy(Nono, America)$

Implications

R_5 $American(x) \wedge Weapon(y) \wedge Sells(x, y, z) \wedge Hostile(z) \Rightarrow Criminal(x)$

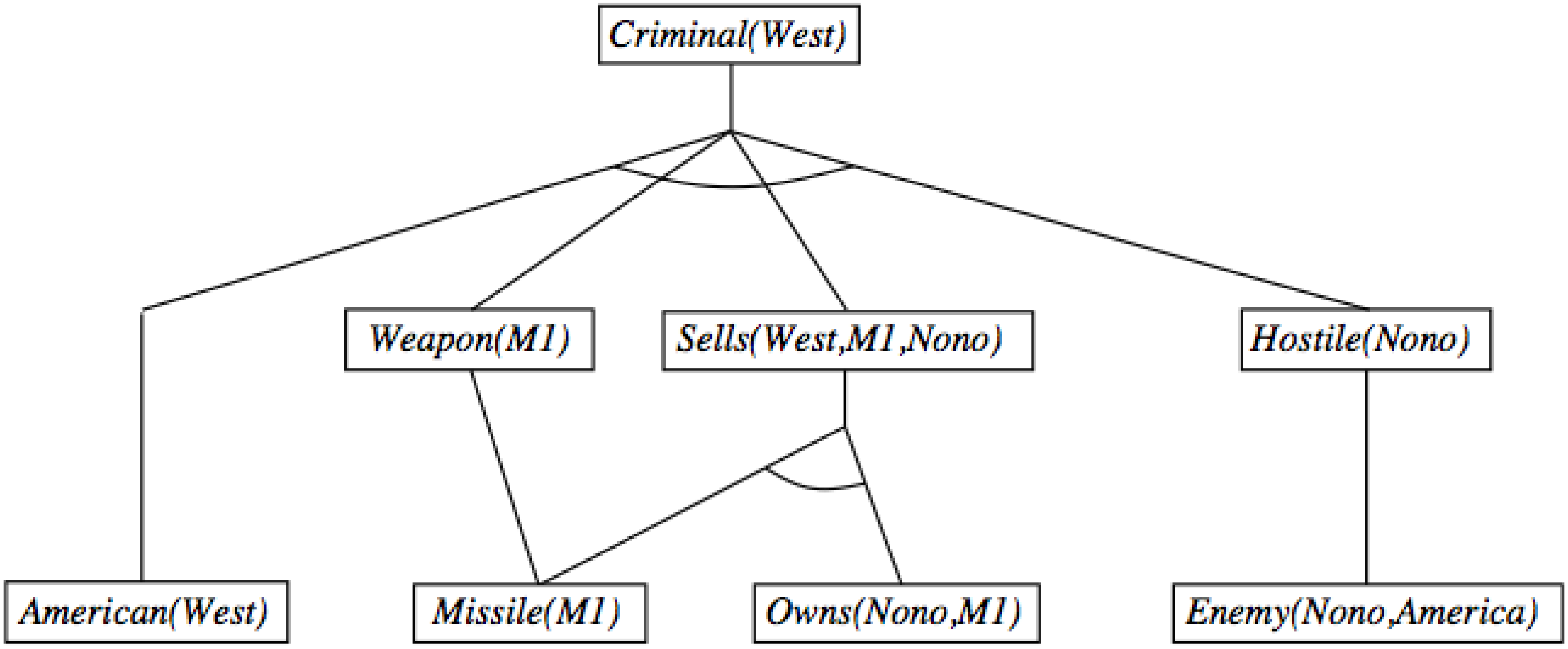
R_6 $Missile(x) \wedge Owns(Nono, x) \Rightarrow Sells(West, x, Nono)$

R_7 $Missile(x) \Rightarrow Weapon(x)$

R_8 $Enemy(x, America) \Rightarrow Hostile(x)$

- 1^{st} iteration, R_5 has unsatisfied premises
  R_6 is satisfied with $\{x/M_1\}, Sells\ (West, M_1, Nono)$ is added.
  R_7 is satisfied with $\{x/M_1\}, Weapon(M_1)$ is added.
  R_8 is satisfied with $\{x/Nono\}, Hostile(Nono)$ is added.
- 2^{nd} iteration, R_5 is satisfied with $\{x/West, y/M_1, z/Nono\},\ Criminal(West)$ is added

Forward chaining proof



Properties of forward chaining

- Sound and complete for first-order definite clauses

  (proof similar to propositional proof)
- May not terminate in general if α is not entailed
- This is unavoidable: entailment with definite clauses is semi-decidable

- Datalog = first-order definite clauses + no functions(e.g., crime KB)
- FC terminates for Datalog in poly iterations

Efficiency of forward chaining

- Simple observation: no need to match a rule on iteration k if a premise wasn't added on iteration $k-1$

  ⇒ match each rule whose premise contains a newly added literal
- Matching itself can be expensive
- Database indexing allows $O(1)$ retrieval of known facts

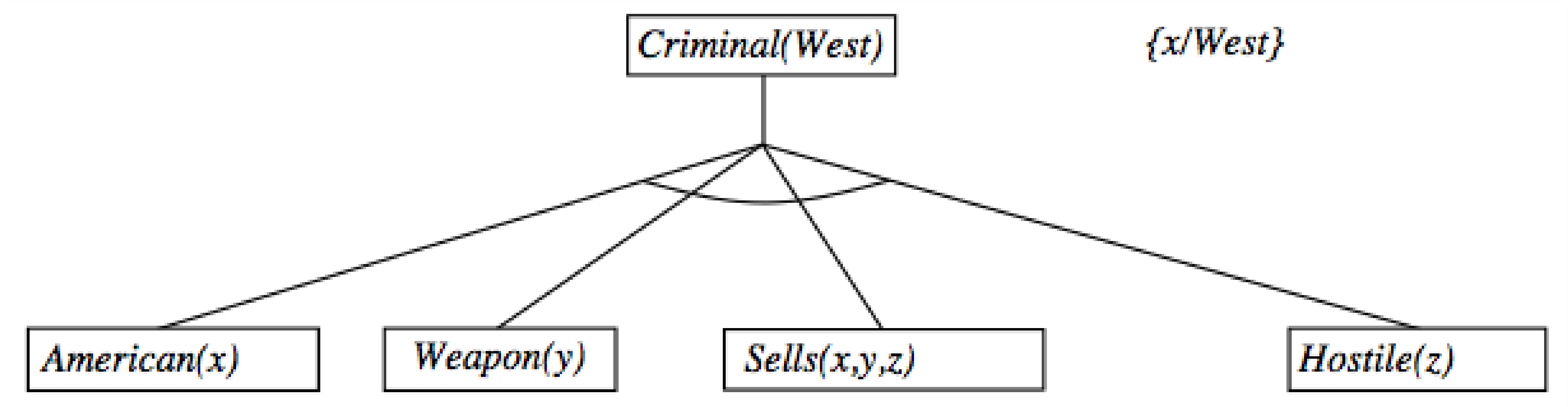
  e.g., query $Missile(x)$ retrives $Missile(M_1)$
- Matching conjunctive premise against known facts is NP-hard
- Forward chaining is widely used in deductive databases

Backward chaining algorithm

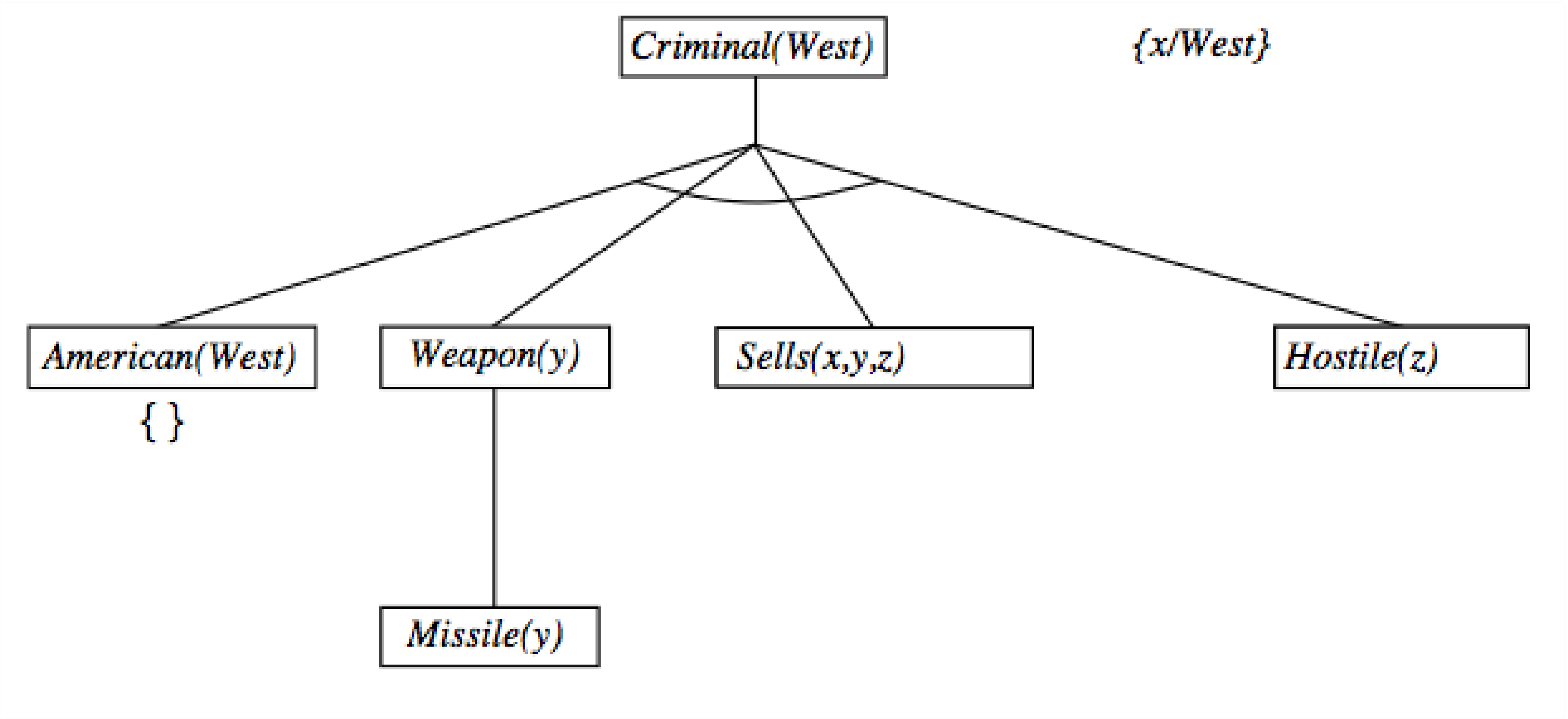
function FOL-BC-ASK(KB, $goals$, θ) **returns** a set of substitutions
  inputs: KB, a knowledge base
    $goals$, a list of conjuncts forming a query (θ already applied)
    θ, the current substitution, initially the empty substitution $\{ \}$
  local variables: $answers$, a set of substitutions, initially empty

  if $goals$ is empty **then return** $\{\theta\}$
  $q' \leftarrow$ SUBST(θ, FIRST($goals$))
  for each sentence r **in** KB
    where STANDARDIZE-APART(r) $= (p_1 \land \ldots \land p_n \Rightarrow q)$
    and $\theta' \leftarrow$ UNIFY(q, q') succeeds
   $new\_goals \leftarrow [p_1, \ldots, p_n | \text{REST}(goals)]$
   $answers \leftarrow$ FOL-BC-ASK(KB, new_goals, COMPOSE(θ', θ)) $\cup$ $answers$
  return $answers$

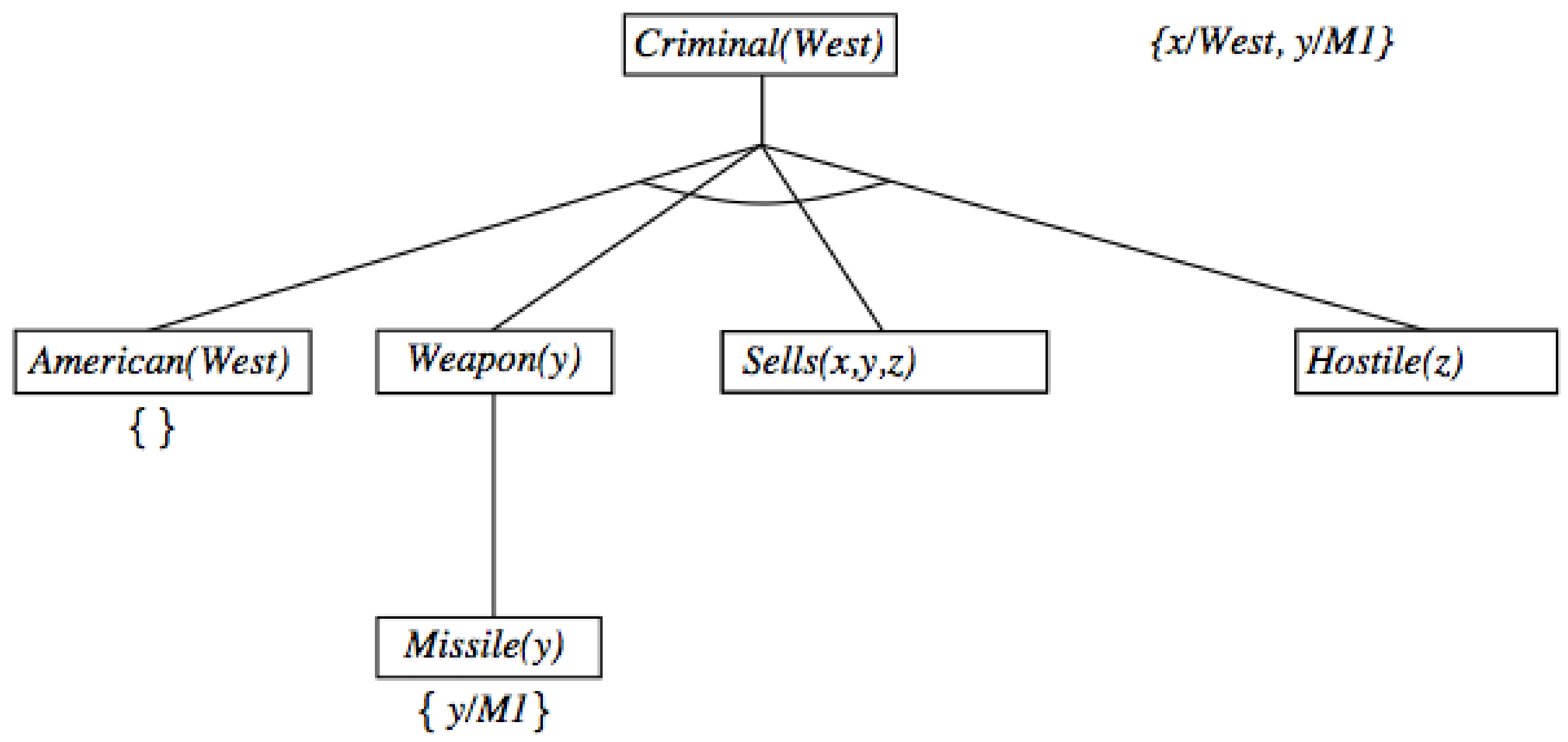
Backward chaining example



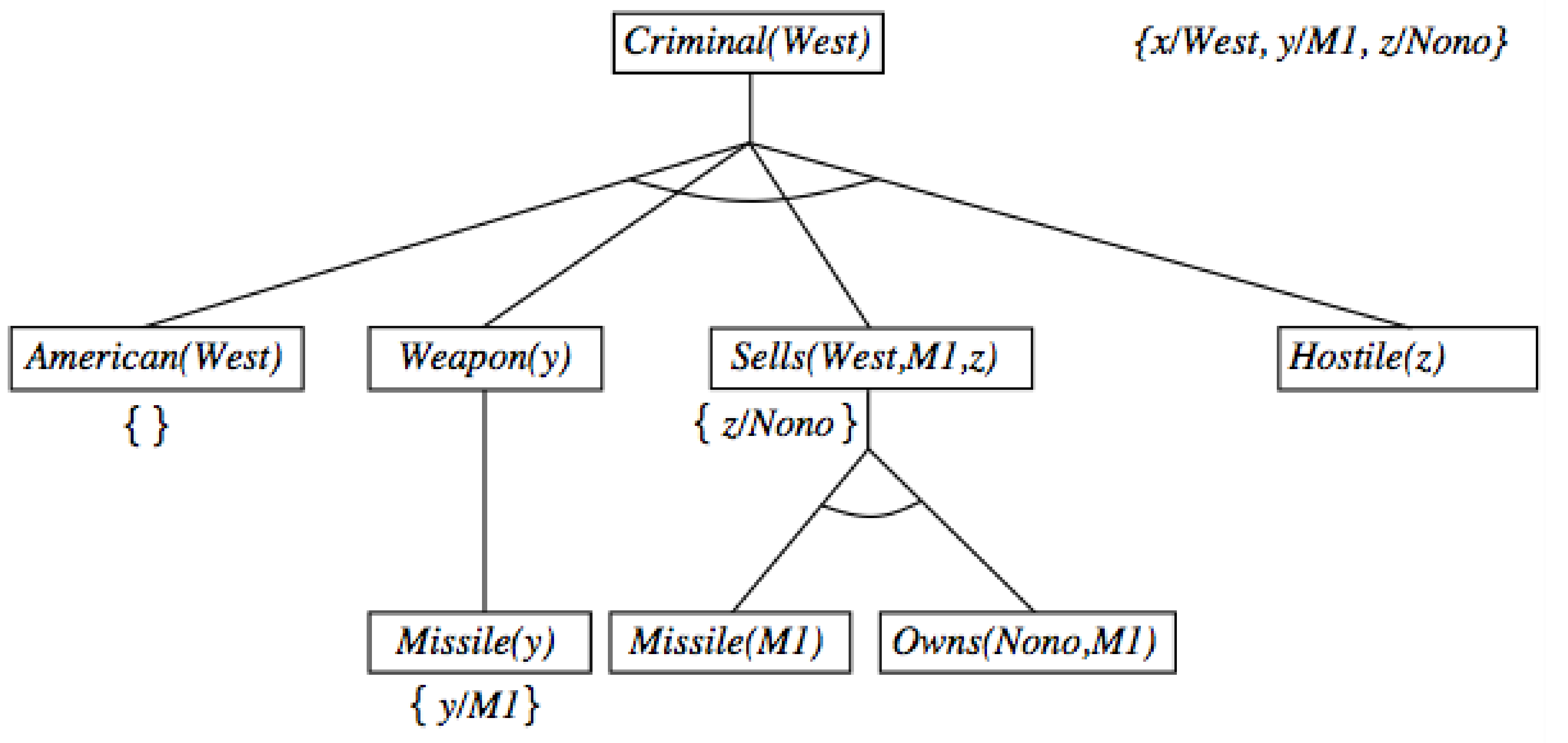
Backward chaining example



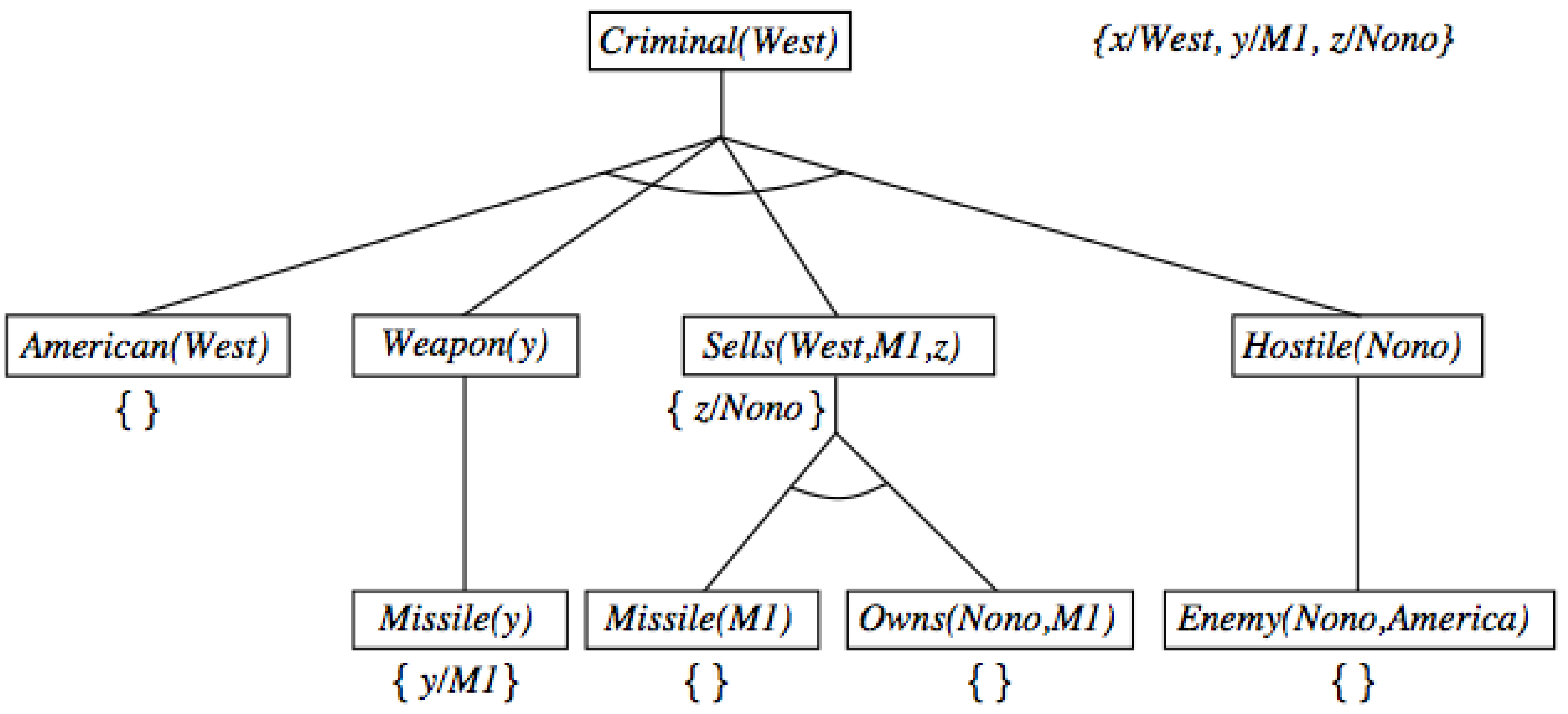
Backward chaining example



Backward chaining example



Backward chaining example



Properties of backward chaining

AND-OR search: AND for all premises; OR since the goal query can be proved by any rules

- Depth-first recursive proof search: space is linear in size of proof
- Incomplete due to infinite loops
  - ⇒ fix by checking current goal against every goal on stack
- Inefficient due to repeated sub-goals (both success and failure)
  - ⇒ fix using caching of previous results (extra space!)
- Widely used (without improvements !) for logic programming

Homework

- Reading Chapter 8.4

9.9 Suppose you are given the following axioms:

1. $0 \leq 3$.
2. $7 \leq 9$.
3. $\forall x \quad x \leq x$.
4. $\forall x \quad x \leq x + 0$.
5. $\forall x \quad x + 0 \leq x$.
6. $\forall x, y \quad x + y \leq y + x$.
7. $\forall w, x, y, z \quad w \leq y \wedge x \leq z \quad \Rightarrow \quad w + x \leq y + z$.
8. $\forall x, y, z \quad x \leq y \wedge y \leq z \quad \Rightarrow \quad x \leq z$

a. Give a backward-chaining proof of the sentence $7 \leq 3 + 9$. (Be sure, of course, to use only the axioms given here, not anything else you may know about arithmetic.) Show only the steps that leads to success, not the irrelevant steps.

b. Give a forward-chaining proof of the sentence $7 \leq 3 + 9$. Again, show only the steps that lead to success.

Logic Programming: Prolog

Logic programming

- Sound bite: computation as inference on logical KBs

	Logic programming	Ordinary programming
1.	Identify problem	Identify problem
2.	Assemble information	Assemble information
3.	Tea break	Figure out solution
4.	Encode information in KB	Program solution
5.	Encode problem instance as facts	Encode problem instance as data
6.	Ask queries	Apply program to data
7.	Find false facts	Debug procedural errors

- Should be easier to debug $Capital(NewYork, US)$ than $x \coloneqq x + 2$

Prolog systems

Basis: backward chaining with Horn clauses + bells & whistles
Widely used in Europe, Japan (basis of 5th Generation project)
Compilation techniques ⇒ approaching a billion LIPS

Program = set of clauses = `head :- literal`$_1$`, ... literal`$_n$`.`

```
criminal(X) :- american(X), weapon(Y), sells(X,Y,Z), hostile(Z).
```

Efficient unification by open coding
Efficient retrieval of matching clauses by direct linking
Depth-first, left-to-right backward chaining
Built-in predicates for arithmetic etc., e.g., `X is Y*Z+3`
Closed-world assumption ("negation as failure")
  e.g., given `alive(X) :- not dead(X).`
  `alive(joe)` succeeds if `dead(joe)` fails

Prolog examples

- Depth-first search from a start state X:

```
dfs(X)  :— goal(x).
dfs(X)  :— successor(X,S),dfs(S).
```

- No need to loop over S: successor succeeds for each

- Appending two lists to produce a third:

```
append({}, Y, Y).
append( [X|L], Y, [X|Z])   :—  append(L, Y, Z).
```

第二个input放到第一个的右边

- Query: append(A, B, [1:2]) ?
- Answers: A=[] B=[1, 2]
  A=[1] B=[2]
  A=[1, 2] B=[]

Prolog systems

- Unification without the occur check, may results in unsound inferences. But almost never a problem in practice.
- Depth-first, left-to-right backward chaining search with no checks for infinite recursion.
- Built-in predicates for arithmetic etc., e.g., $X\ is\ Y * Z + 3$; no arbitrary equation solving. e.g., $5\ is\ X + Y$ fails
- Database semantics instead of first-order semantics
  - Closed-world assumption — anything not known to be true is false.
  - Unique-names assumption— different names refer to distinct objects.
  - Domain closure— only those mentioned exist in the domain.

Summary

- For small domains, we can use UI and EI to propositionalize the problem
- Unification identifies proper substitutions, more efficient than instantiation.
- Forward and backward chaining uses the generalized Modus Ponens on a sets of definite clauses.
- GMP is complete for definite clauses, where the entailment is semi-decidable; for Datalog KB (function-less definite clauses), entailment can be decided in $\mathcal{P}$-time (with forward-chaining)
- Backward chaining is used in logic programming systems; inferences are fast but may be unsound or incomplete.
- Resolution is sound and (refutation-) complete for FOL, using CNF KB

Homework

member(1,[1,2,3,4,5])
member(3,[1,2,3,4,5])　　要求：给出一个集合，列出其所有元素

subset([2,4],[1,2,3,4,5])　　要求：给出一个集合，列出其所有子集

Homework

事实的集合：有三种事实

is_relation (fact_ID, company_name, time, index-name, value)

supplier (fact_ID, time, company_A, company_B, k, value)

client (fact_ID, time, company_A, company_B, k, value)

检测冲突的事实（facts）

给定一个事实的集合，找出所有“冲突”的fact ID对

思考：什么情形“发生冲突”

fact2.pl