Previous lectures: Inference in FOL

- Universal instantiation
- Existantial instantiation
- Reduction to propositional logic
- Unification

$$\frac{\forall v \;\; \alpha}{\textsc{Subst}(\{v/g\}, \alpha)} \qquad \frac{\exists v \;\; \alpha}{\textsc{Subst}(\{v/k\}, \alpha)}$$

... 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

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

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 ...

$Enemy(Nono, America)$

Quiz

Write down logical representations for the following sentences in First-Order Logic:

a. Everyone who loves all animals is loved by someone.

b. Anyone who kills an animal is loved by no one.

c. Jack loves all animals.

d. Either Jack or Curiosity killed the cat, who is named Tuna.

e. Cat is an animal.

f. Curiosity killed the cat.

a. Everyone who loves all animals is loved by someone.

- If all animals are loved by somebody

  then that somebody will be loved by someone

- If there is somebody who loves all **y and all y are animals**

  then that somebody will be loved by someone

b. Anyone who kills an animal is loved by no one.

- **X everyone who kills all animals is loved by no one.**
-

Unification

Definition

The process of finding a substitution that makes two literals complementary is called unification.

Two literals for which a unifying substitution exists are called unifiable.

Importance of Unification

- It is the basis for FOL resolution.
- It is the main way rule-based systems determine which rules apply in a situation.
- It is the way variables are treated in logic

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

$\text{UNIFY}(\alpha, \beta) = \theta$ if $\alpha\theta = \beta\theta$

p	q	θ
$Knows(John, x)$	$Knows(John, Jane)$	
$Knows(John, x)$	$Knows(y, OJ)$	
$Knows(John, x)$	$Knows(y, Mother(y))$	
$Knows(John, x)$	$Knows(x, OJ)$	

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

$\textsc{Unify}(\alpha, \beta) = \theta$ if $\alpha\theta = \beta\theta$

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

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

$\textsc{Unify}(\alpha, \beta) = \theta$ if $\alpha\theta = \beta\theta$

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

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

$\textsc{Unify}(\alpha, \beta) = \theta$ if $\alpha\theta = \beta\theta$

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

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

$\textsc{Unify}(\alpha, \beta) = \theta$ if $\alpha\theta = \beta\theta$

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., $Knows(z_{17}, OJ)$

Generalized Modus Ponens (GMP)

$$\frac{p_1{}',\ \ p_2{}',\ \ldots,\ p_n{}',\ \ (p_1 \wedge p_2 \wedge \ldots \wedge p_n \Rightarrow q)}{q\theta} \qquad \text{where } p_i{}'\theta = p_i\theta \text{ for all } i$$

$p_1{}'$ is $King(John)$ p_1 is $King(x)$
$p_2{}'$ is $Greedy(y)$ p_2 is $Greedy(x)$
θ is $\{x/John, y/John\}$ 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

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**
      $(p_1 \land \ldots \land p_n \Rightarrow q) \leftarrow$ STANDARDIZE-APART(r)
      for each θ such that $(p_1 \land \ldots \land p_n)\theta = (p'_1 \land \ldots \land p'_n)\theta$
          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*

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

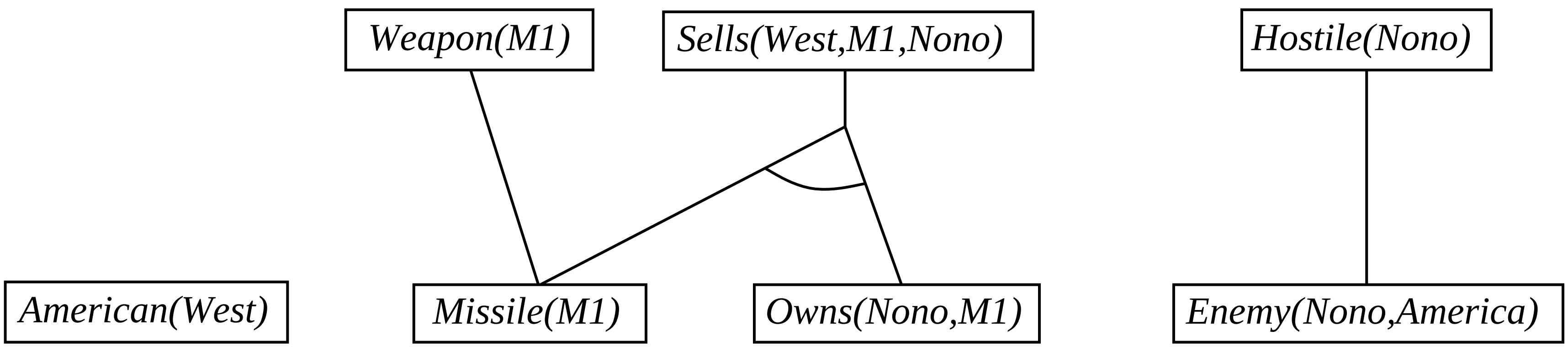
repeat until new is empty
$new \leftarrow \{\}$
for each $rule$ **in** KB **do**
$(p_1 \wedge \ldots \wedge p_n \Rightarrow q) \leftarrow$ STANDARDIZE-VARIABLES($rule$)
for each θ such that SUBST($\theta, p_1 \wedge \ldots \wedge p_n$) = SUBST($\theta, p'_1 \wedge \ldots \wedge p'_n$)
for some $p'_1, \ldots, p'_n$ 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*

... 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
$\forall\, x\ \ Missile(x) \wedge Owns(Nono, x) \Rightarrow Sells(West, x, Nono)$
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 ...
$Enemy(Nono, America)$

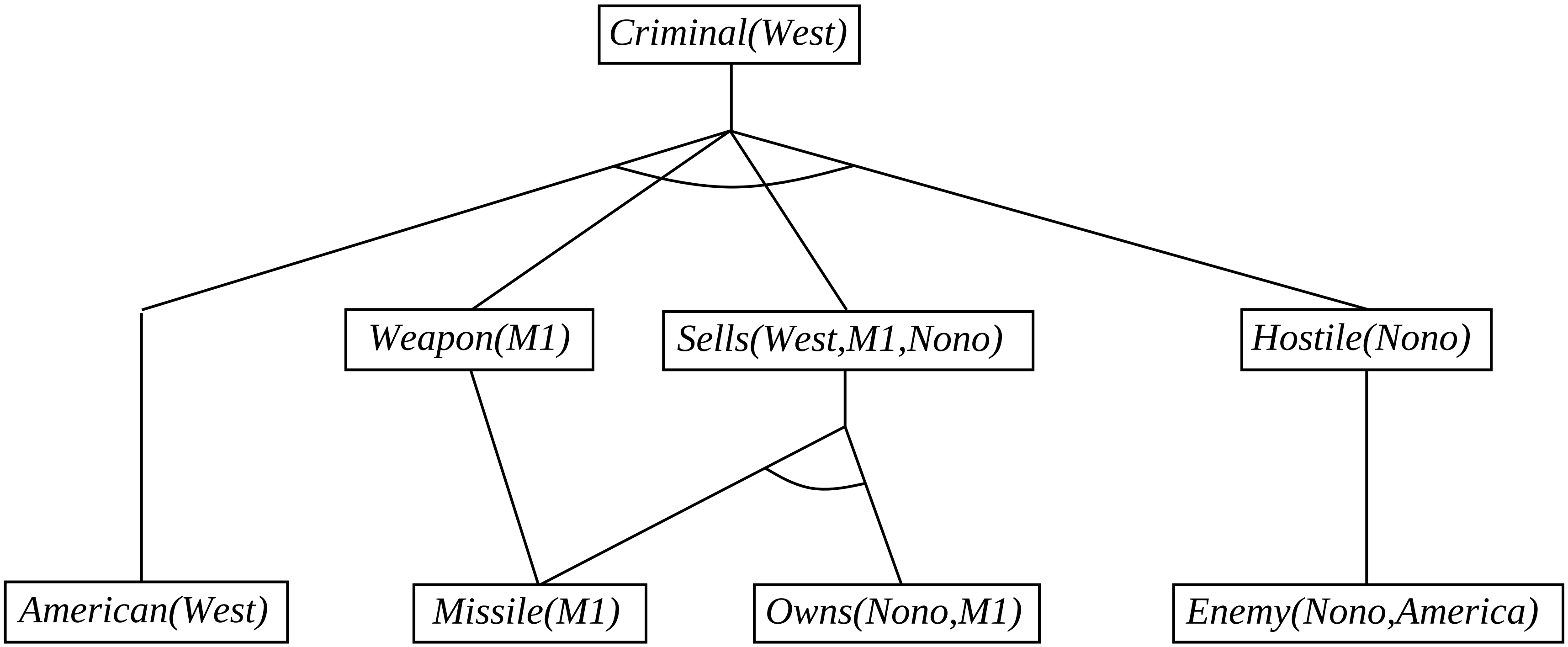
Forward chaining proof

American(West) | *Missile(M1)* | *Owns(Nono,M1)* | *Enemy(Nono,America)*

Forward chaining proof



Forward chaining proof



Backward chaining algorithm

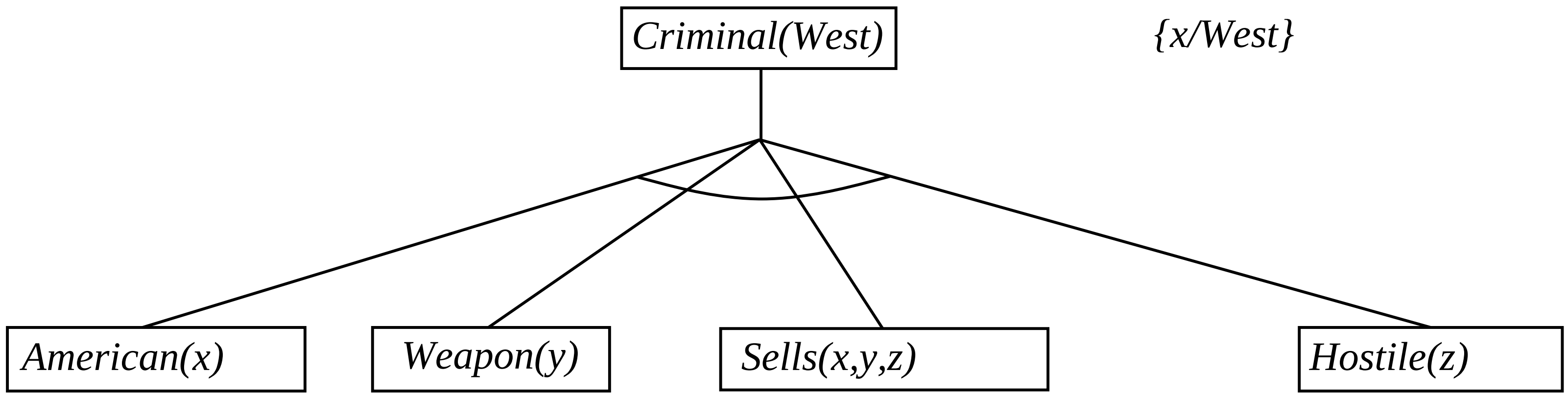
function FOL-BC-ASK($KB, goals, \theta$) **returns** a set of substitutions
  inputs: KB, a knowledge base
    $goals$, a list of conjuncts forming a query (θ already applied)
    θ, the current substitution, initially the empty substitution $\{\,\}$
  local variables: $answers$, a set of substitutions, 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) $= (\, p_1 \ \wedge \ \ldots \ \wedge \ p_n \ \Rightarrow \ q)$
    and $\theta' \leftarrow$ UNIFY(q, q') succeeds
   $new_goals \leftarrow [\, p_1, \ldots, p_n |$REST($goals$)$]$
   $answers \leftarrow$ FOL-BC-ASK(KB, new_goals, COMPOSE(θ', θ)) $\cup$ $answers$
  return $answers$

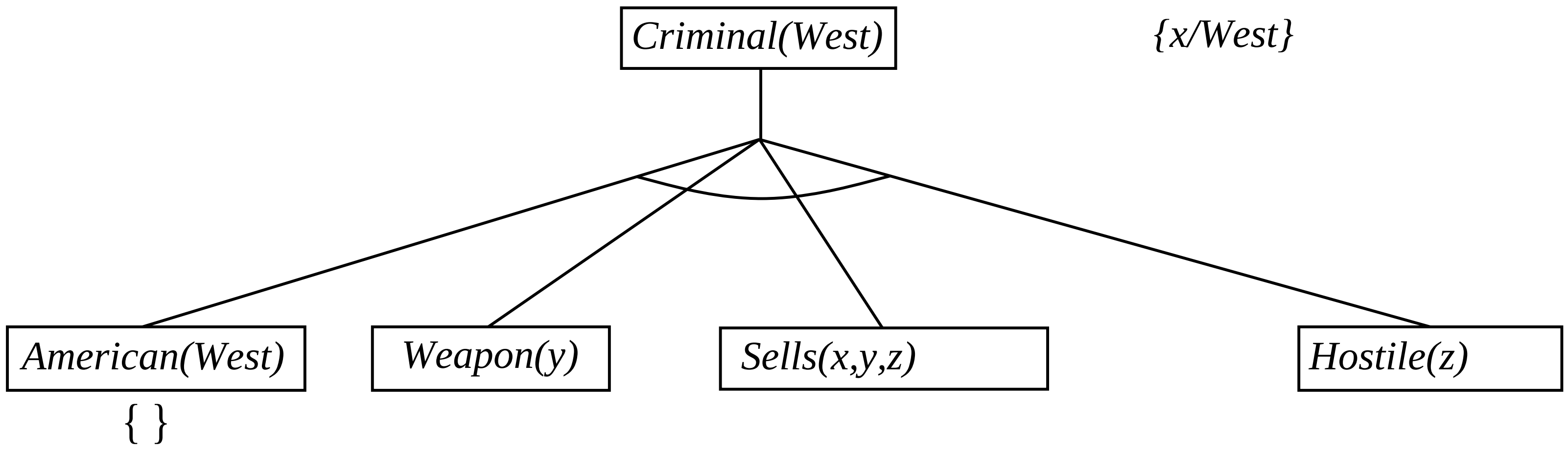
Backward chaining example

$\boxed{Criminal(West)}$

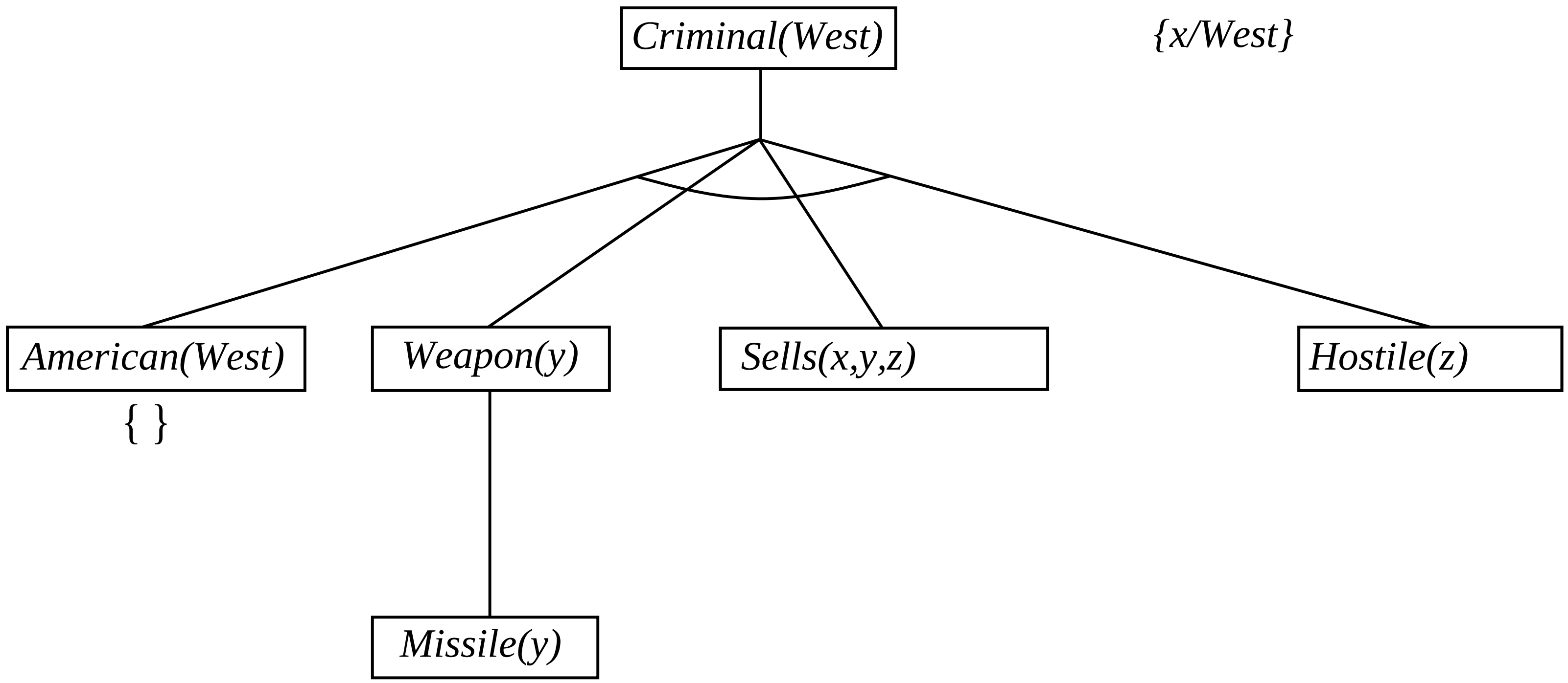
Backward chaining example



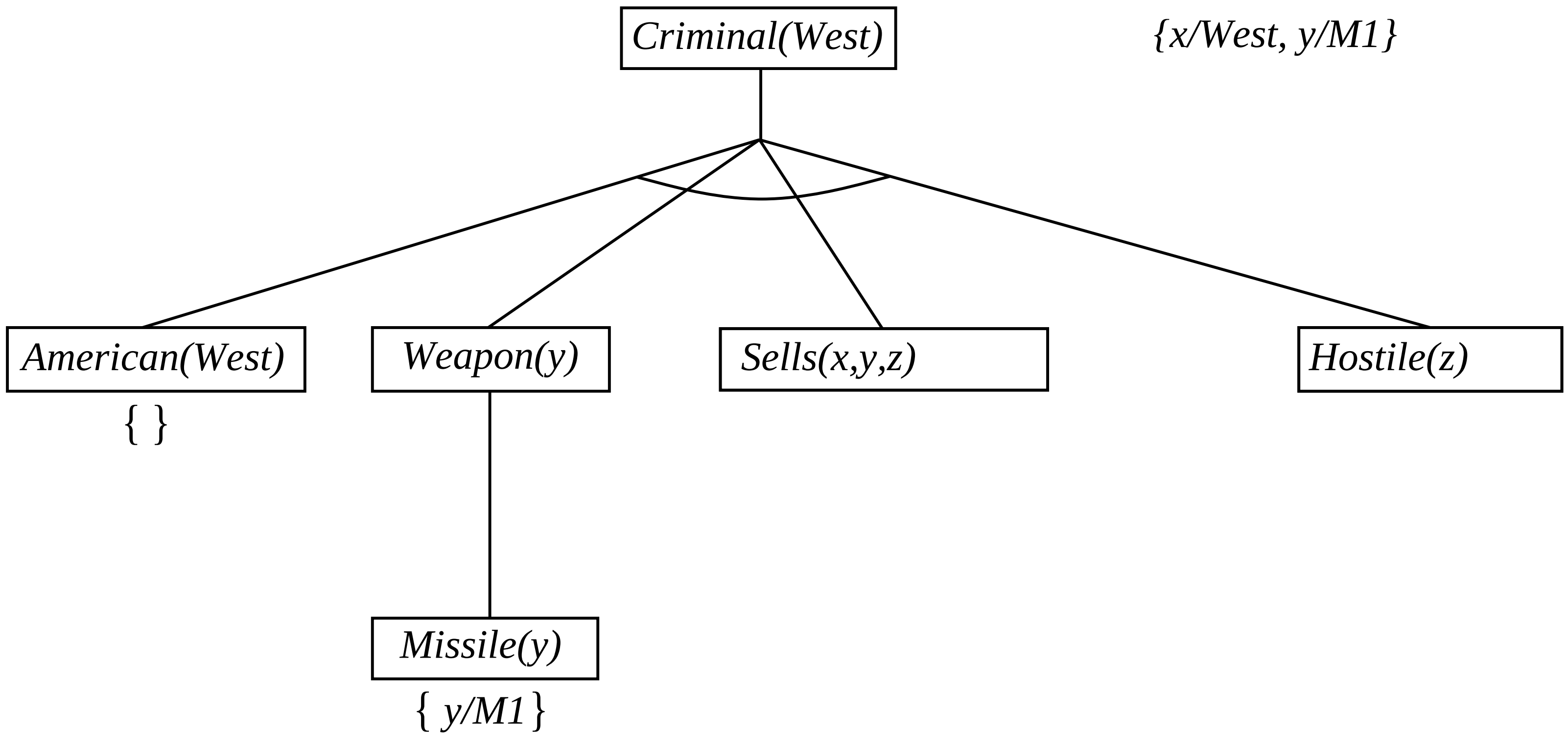
Backward chaining example



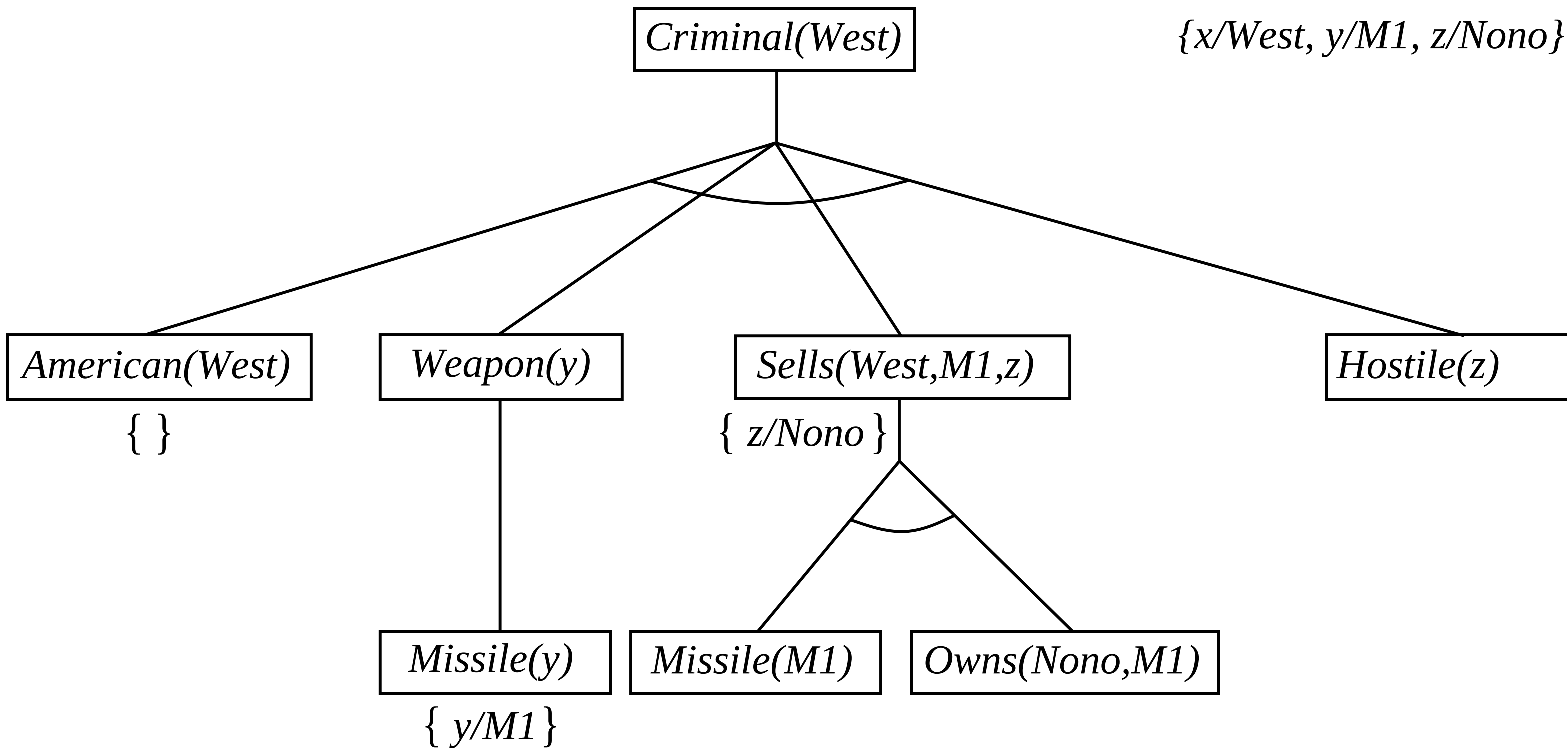
Backward chaining example



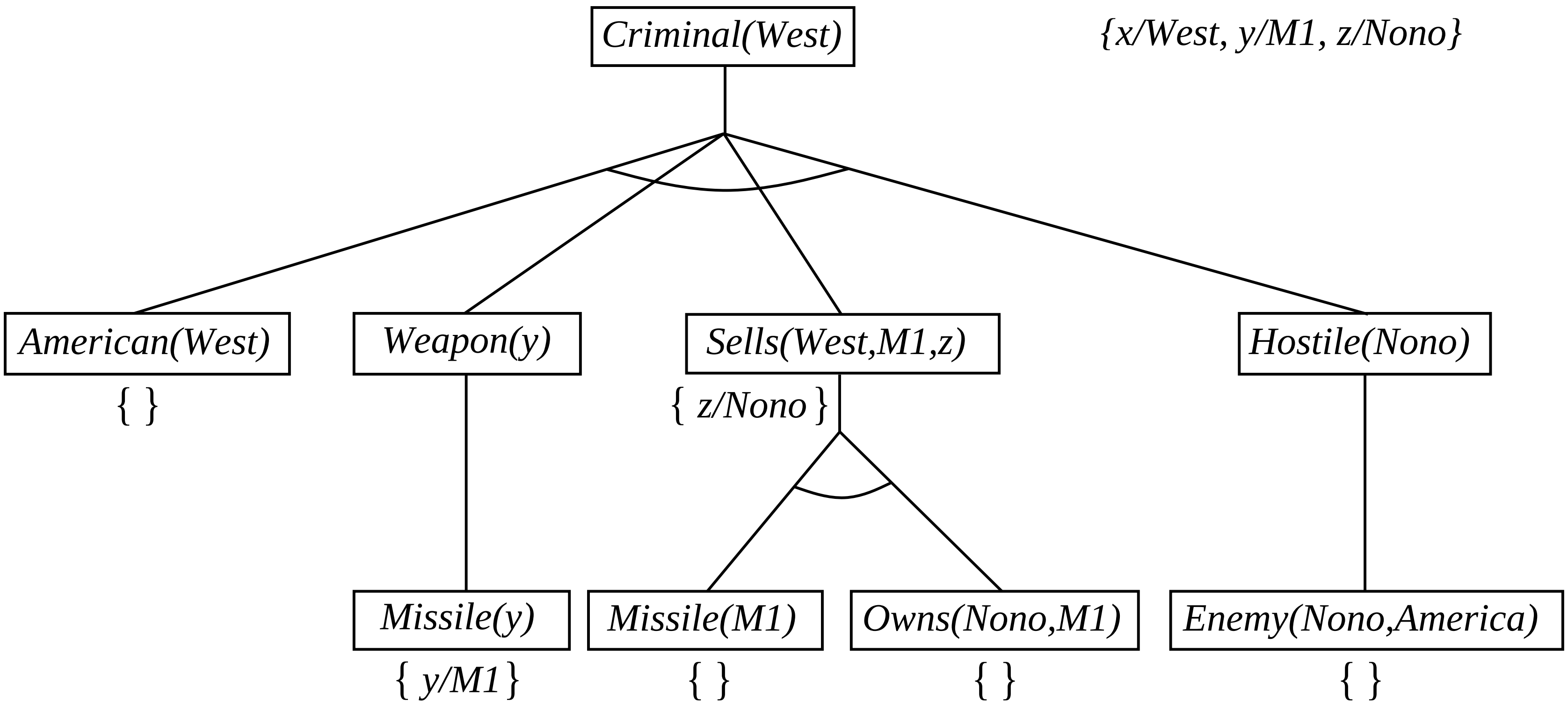
Backward chaining example



Backward chaining example



Backward chaining example



Properties of backward chaining

Depth-first recursive proof search: space is linear in size of proof

Incomplete due to infinite loops
$\Rightarrow$ fix by checking current goal against every goal on stack

Inefficient due to repeated subgoals (both success and failure)
$\Rightarrow$ fix using caching of previous results (extra space!)

Widely used (without improvements!) for logic programming

Previous lecture: Resolution algorithm

- Proof by contradiction.

To prove $(KB \models \alpha)$, show $(KB \wedge \neg\alpha)$ is unsatisfiable.

Resolution algorithm

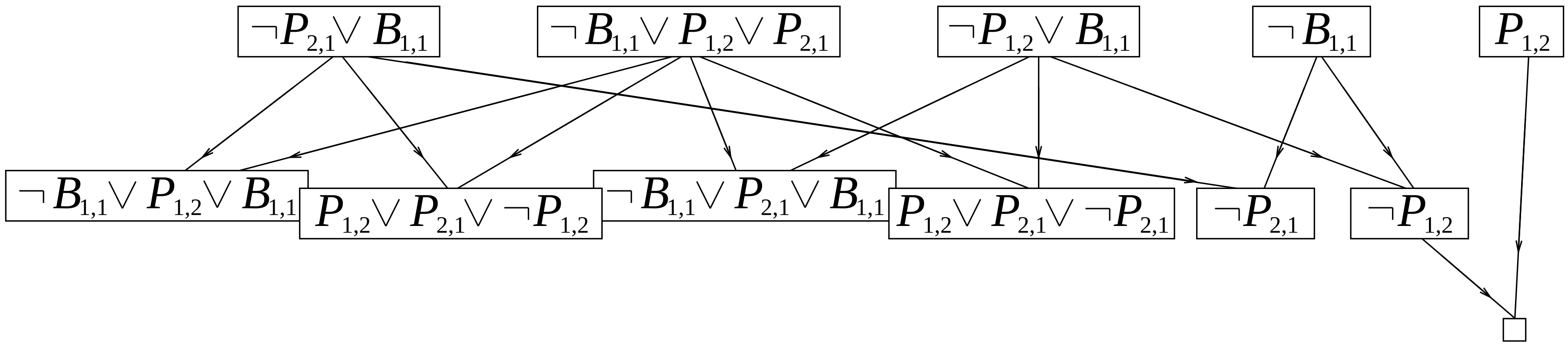
Proof by contradiction, i.e., show $KB \wedge \neg\alpha$ unsatisfiable

function PL-RESOLUTION(KB, α) **returns** *true* or *false*
  inputs: KB, the knowledge base, a sentence in propositional logic
    α, the query, a sentence in propositional logic

  clauses ← the set of clauses in the CNF representation of $KB \wedge \neg\alpha$
  new ← { }
  loop do
    for each C_i, C_j **in** *clauses* **do**
      resolvents ← PL-RESOLVE(C_i, C_j)
      if *resolvents* contains the empty clause **then return** *true*
      new ← *new* ∪ *resolvents*
    if *new* ⊆ *clauses* **then return** *false*
    clauses ← *clauses* ∪ *new*

Resolution example

$KB = (B_{1,1} \Leftrightarrow (P_{1,2} \vee P_{2,1})) \wedge \neg B_{1,1}$ $\alpha = \neg P_{1,2}$



Resolution: brief summary

Full first-order version:

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

where $\textsc{Unify}(\ell_i, \neg m_j) = \theta$.

$[Animal(F(x)) \vee Loves(G(x), x)]$ and $[\neg Loves(u, v) \vee \neg Kills(u, v)]$

unifier?
resolvent clause?

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 all animals is loved by someone:

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

1. Eliminate biconditionals and implications

2. Move $\neg$ inwards: $\neg\forall x, p \;\equiv \exists x \;\; \neg p$, $\neg\exists x, p \;\equiv \forall x \;\; \neg p$:

Conversion to CNF contd.

3. Standardize variables: each quantifier should use a different one

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

5. Drop universal quantifiers:

6. Distribute $\wedge$ over $\vee$:

Conversion to CNF contd.

3. Standardize variables: each quantifier should use a different one

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

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

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

5. Drop universal quantifiers:

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

6. Distribute $\land$ over $\lor$:

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

Conversion to CNF contd.

3. Standardize variables: each quantifier should use a different one

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

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

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

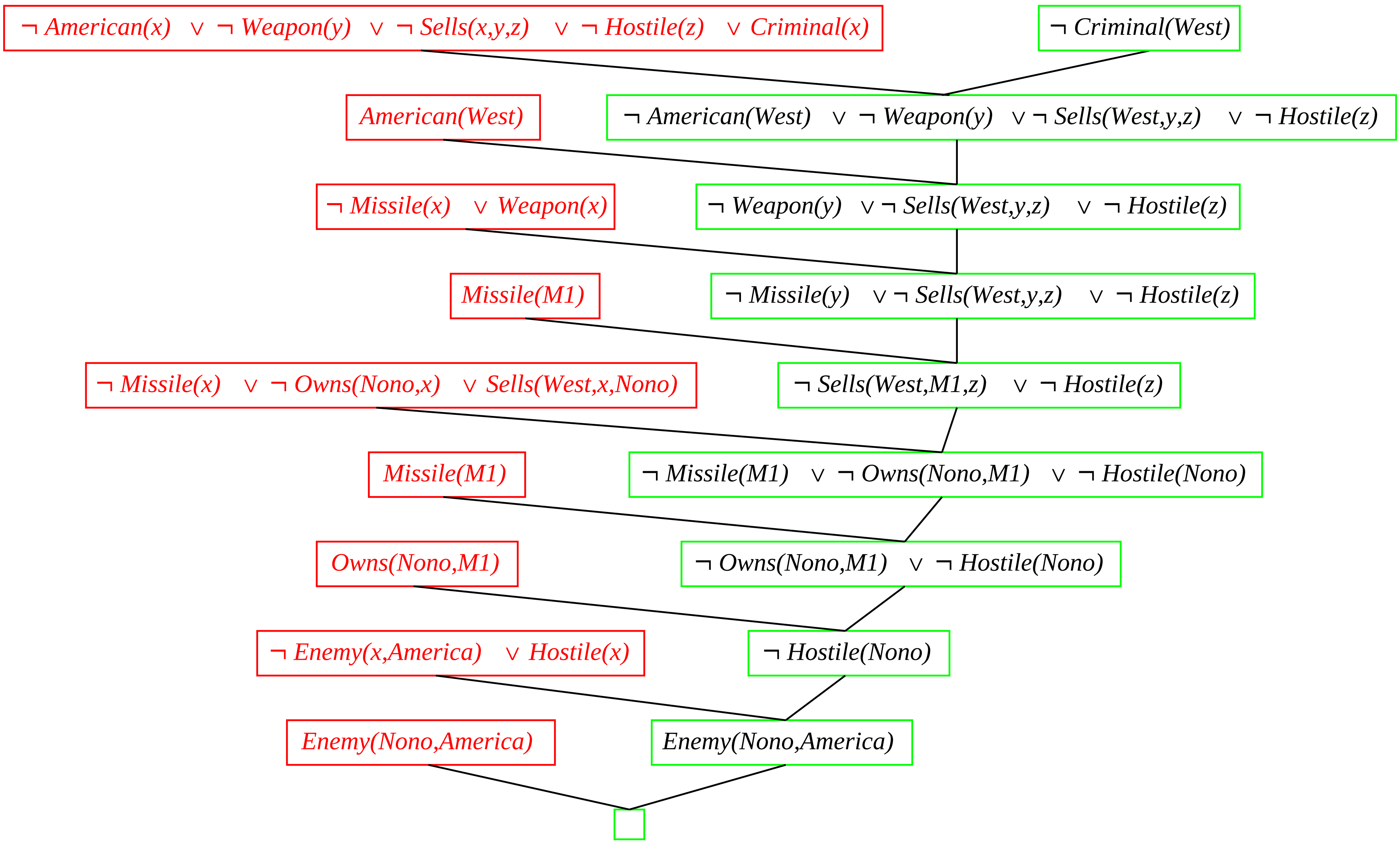
5. Drop universal quantifiers:

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

6. Distribute $\land$ over $\lor$:

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

Resolution proof: definite clauses



Resolution: Another example

Write down logical representations for the following sentences in First-Order Logic:

a. Everyone who loves all animals is loved by someone.

b. Anyone who kills an animal is loved by no one.

c. Jack loves all animals.

d. Either Jack or Curiosity killed the cat, who is named Tuna.

e. Cat is an animal.

f. Did curiosity kill the cat?

Convert to FOL

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

$\forall x\ [\exists y\ Animal(y) \wedge Kills(x,y)] \Rightarrow [\forall z\ \neg Loves(z,x)]$

$\forall x\ Animal(x) \Rightarrow Loves(Jack,x)$

$Kills(Jack,Tuna) \vee Kills(Curiosity,Tuna)$

$Cat(Tuna)$

$\forall x\ Cat(x) \Rightarrow Animal(x)$

$\neg Kills(Curiosity,Tuna)$

FOL to CNF

$\text{Animal}(F(x)) \lor \text{Loves}(G(x),x)$

$\neg\text{Loves}(x,F(x)) \lor \text{Loves}(G(x),x)$

$\neg\text{Animal}(y) \lor \neg\text{Kills}(x,y) \lor \neg\text{Loves}(z,x)$

$\neg\text{Animal}(x) \lor \text{Loves}(\text{Jack},x)$

$\text{Kills}(\text{Jack},\text{Tuna}) \lor \text{Kills}(\text{Curiosity},\text{Tuna})$

$\text{Cat}(\text{Tuna})$

$\neg\text{Cat}(x) \lor \text{Animal}(x)$

$\neg\text{Kills}(\text{Curiosity},\text{Tuna})$

Resolution by Refutation

- Exercise

Resolution by Refutation

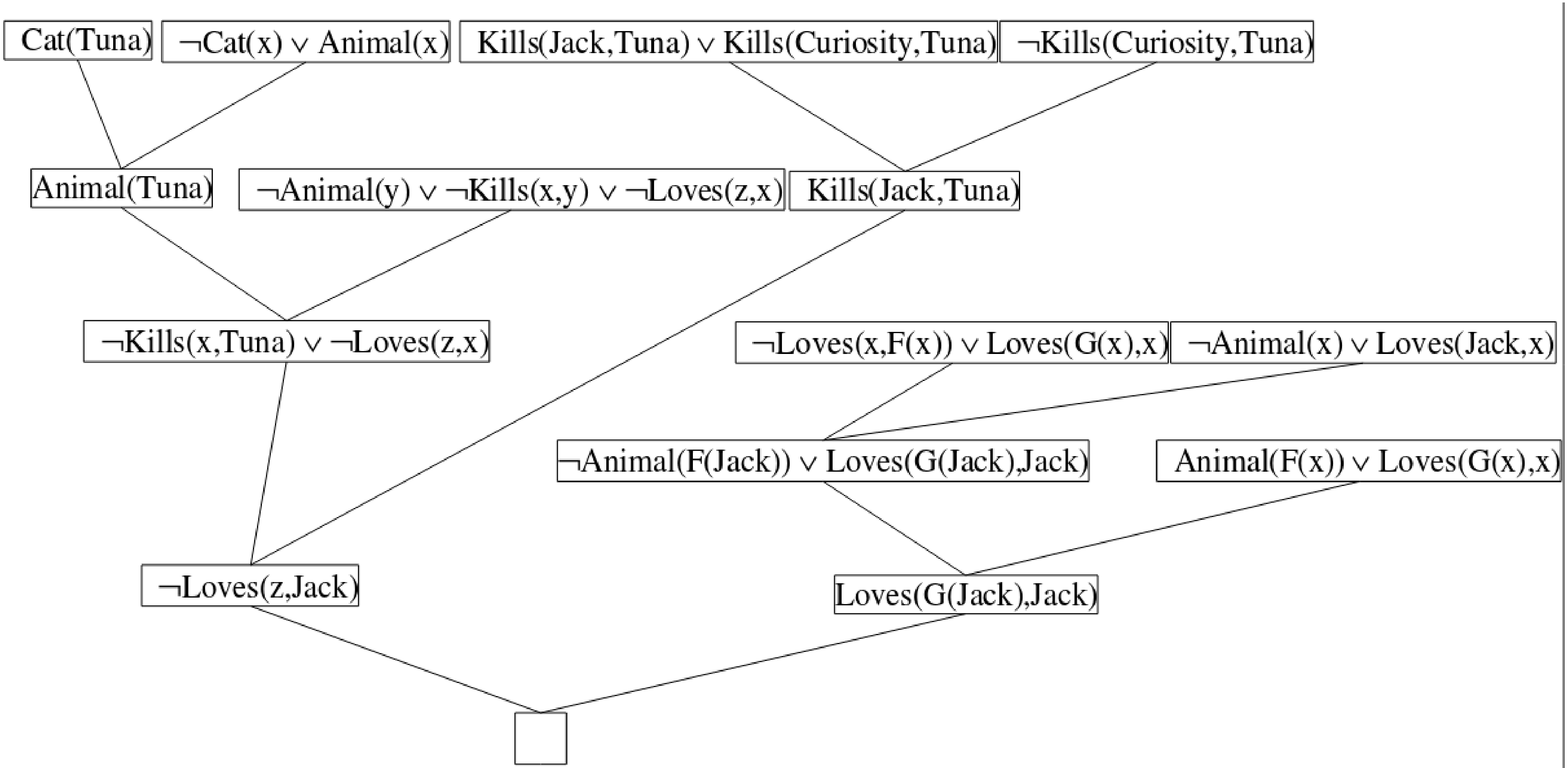
$Cat(Tuna)$ | $\neg Cat(x) \vee Animal(x)$ | $Kills(Jack,Tuna) \vee Kills(Curiosity,Tuna)$ | $\neg Kills(Curiosity,Tuna)$

$\neg Animal(y) \vee \neg Kills(x,y) \vee \neg Loves(z,x)$

$\neg Loves(x,F(x)) \vee Loves(G(x),x)$ | $\neg Animal(x) \vee Loves(Jack,x)$

$Animal(F(x)) \vee Loves(G(x),x)$

Resolution by Refutation



Properties of backward chaining

Depth-first recursive proof search: space is linear in size of proof

Incomplete due to infinite loops
  $\Rightarrow$ fix by checking current goal against every goal on stack

Inefficient due to repeated subgoals (both success and failure)
  $\Rightarrow$ fix using caching of previous results (extra space!)

Widely used (without improvements!) for logic programming

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 $x := x + 2$!

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).
```

```
query:   append(A,B,[1,2]) ?
answers: A=[]    B=[1,2]
         A=[1]   B=[2]
         A=[1,2] B=[]
```

Planning

- Problem solving agents
- Logical Agents
- Planners

PDDL

- a **factored representation**: a state of the world is represented by a collection of variables
- **state**: a conjunction of fluents that are ground, functionless atoms.
- the closed-world assumption
- Not used:
  - At(x, y)
  - ¬Poor,
  - At(Father (Fred), Sydney)
- Deals with frame problem: only mentions Δ
  - everything that stays the same is left unmentioned

$$
\begin{aligned}
&Action(Fly(p, from, to),\\
&\quad \text{PRECOND:} At(p, from) \land Plane(p) \land Airport(from) \land Airport(to)\\
&\quad \text{EFFECT:} \neg At(p, from) \land At(p, to))
\end{aligned}
$$

PDDL

$$\begin{aligned}&Action(Fly(p, from, to),\\&\quad \text{PRECOND:} At(p, from) \wedge Plane(p) \wedge Airport(from) \wedge Airport(to)\\&\quad \text{EFFECT:} \neg At(p, from) \wedge At(p, to))\end{aligned}$$

$$\begin{aligned}&\forall\, p, from, to \;\; (Fly(p, from, to) \in \text{ACTIONS}(s)) \;\Leftrightarrow\\&\quad s \models (At(p, from) \wedge Plane(p) \wedge Airport(from) \wedge Airport(to))\end{aligned}$$

We say that action a is **applicable** in state s if the preconditions are satisfied by s.

Propositionalize a PDDL problem
replace each action schema with a set of ground actions
then use a propositional solver such as SATPLAN to find a solution.
This is impractical when v and k are large.

$$\text{RESULT}(s, a) = (s - \text{DEL}(a)) \cup \text{ADD}(a)\;.$$

$$F^{t+1} \Leftrightarrow ActionCausesF^t \vee (F^t \wedge \neg ActionCausesNotF^t)\;.$$

In PDDL the times and states are implicit in the action schemas: the precondition always refers to time t and the effect to time $t+1$.

PDDL

- Definition of a planning **domain**: A set of action schemas
- A specific **problem**: within the domain is defined with the addition of an initial state and a goal.

$$
\begin{aligned}
& Init(At(C_1, SFO) \wedge At(C_2, JFK) \wedge At(P_1, SFO) \wedge At(P_2, JFK) \\
& \quad \wedge Cargo(C_1) \wedge Cargo(C_2) \wedge Plane(P_1) \wedge Plane(P_2) \\
& \quad \wedge Airport(JFK) \wedge Airport(SFO)) \\
& Goal(At(C_1, JFK) \wedge At(C_2, SFO)) \\
& Action(Load(c, p, a), \\
& \quad \text{PRECOND: } At(c, a) \wedge At(p, a) \wedge Cargo(c) \wedge Plane(p) \wedge Airport(a) \\
& \quad \text{EFFECT: } \neg At(c, a) \wedge In(c, p)) \\
& Action(Unload(c, p, a), \\
& \quad \text{PRECOND: } In(c, p) \wedge At(p, a) \wedge Cargo(c) \wedge Plane(p) \wedge Airport(a) \\
& \quad \text{EFFECT: } At(c, a) \wedge \neg In(c, p)) \\
& Action(Fly(p, from, to), \\
& \quad \text{PRECOND: } At(p, from) \wedge Plane(p) \wedge Airport(from) \wedge Airport(to) \\
& \quad \text{EFFECT: } \neg At(p, from) \wedge At(p, to))
\end{aligned}
$$

Figure 10.1 A PDDL description of an air cargo transportation planning problem.

PDDL example

Black(x)

On(x,y)

Clear(x)

Action: Move (b, x, y)...

Action: MoveToTable (b,x)

PDDL example

$$
\begin{aligned}
&Init(On(A, Table) \wedge On(B, Table) \wedge On(C, A) \\
&\quad \wedge Block(A) \wedge Block(B) \wedge Block(C) \wedge Clear(B) \wedge Clear(C)) \\
&Goal(On(A, B) \wedge On(B, C)) \\
&Action(Move(b, x, y), \\
&\quad \text{PRECOND: } On(b, x) \wedge Clear(b) \wedge Clear(y) \wedge Block(b) \wedge Block(y) \wedge \\
&\qquad\qquad (b \neq x) \wedge (b \neq y) \wedge (x \neq y), \\
&\quad \text{EFFECT: } On(b, y) \wedge Clear(x) \wedge \neg On(b, x) \wedge \neg Clear(y)) \\
&Action(MoveToTable(b, x), \\
&\quad \text{PRECOND: } On(b, x) \wedge Clear(b) \wedge Block(b) \wedge (b \neq x), \\
&\quad \text{EFFECT: } On(b, Table) \wedge Clear(x) \wedge \neg On(b, x))
\end{aligned}
$$

Automated Planning

Planning research has been central to AI from the beginning, partly because of practical interest but also because of the "intelligence" features of human planners.

◇ Large logistics problems, operational planning, robotics, scheduling etc.

◇ A number of international Conferences on Planning

◇ Bi-annual Planning competition

Automated Planning

The setting: a single agent in a fully observable, deterministic and static environment.

Propositional logic can express small domain planning problems, but becomes impractical if there are many actions and states (combinatorial explosion).

Example: In the wumpus world the action of a forward-step has to be written for all four directions, for all n^2 locations, and for each time step T.

The Planning Domain Definition Language (PDDL) is a subset of FOL and more expressive than propositional logic. It allows for factored representation.

Planning Domain Definition Language (PDDL)

PDDL is derived from the STRIPS planning language.

– Initial and goal states.
– A set of $\textsc{Actions}(s)$ in terms of preconditions and effects.
– Closed world assumption: Unmentioned state variables are assumed false.

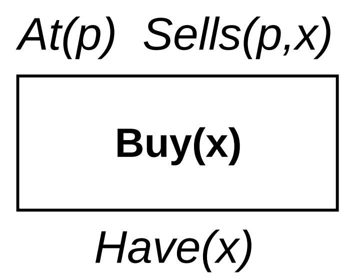
Example:

$\textsc{Action}$: Fly($from$, to)
$\textsc{Precondition}$: At(p, $from$), Plane(p), Airport($from$), Airport(to)
$\textsc{Effect}$: ¬At(p, $from$), At(p, to)

PDDL/STRIPS operators

Tidily arranged actions descriptions, restricted language

ACTION: Buy(x)
PRECONDITION: At(p), Sells(p, x)
EFFECT: Have(x)



[Note: this abstracts away many important details of buying!]

Restricted language $\Rightarrow$ efficient algorithm

Precondition: conjunction of positive literals

Effect: conjunction of literals

A complete set of STRIPS operators can be translated into a set of successor-state axioms

Example: Air cargo transport

A classical transportation problem: Loading and unloading cargo and flying between different airports.

Actions: Load(cargo, plane, airport), Unload(cargo, plane, airport),
Fly(plane, airport, airport)

Predicates: In(cargo, plane), At(cargo$\vee$plane, airport)

Example solution:
Load(C1, P1, SFO), Fly(P1, SFO, JFK), Unload(C1, P1, JFK),
Load(C2, P2, JFK), Fly(P2, JFK, SFO), Unload(C2, P2, SFO).

Example: The blocks world

Cube-shape blocks sitting on a table or stacked on top of each other.

Actions: PutOn(block, block), PutOnTable(block)

Predicates: On(block, block$\vee$table), Clear(block$\vee$table)

How difficult is planning?

Does there exist a plan that achieves the goal? PlanSat

Does there exist a solution of length at most k? Bounded PlanSat

PlanSat and Bounded PlanSat are PSPACE-complete.
– i.e., difficult!

PlanSat without negative preconditions and without negative effects is in P.
– i.e., solveable!

State-space search

◇ Forward (progression):
state-space search considers actions that are **applicable**

◇ Backward (regression):
state-space search considers actions that are **relevant**

Neither of them is efficient without good heuristics!

Heuristics for forward state-space search

For forward state-space search there are a number of domain-independent heuristics:

◇ Relaxing actions:
- Ignore-preconditions heuristic
- Ignore-delete-lists heuristic

◇ State abstractions:
- Reduce the state space

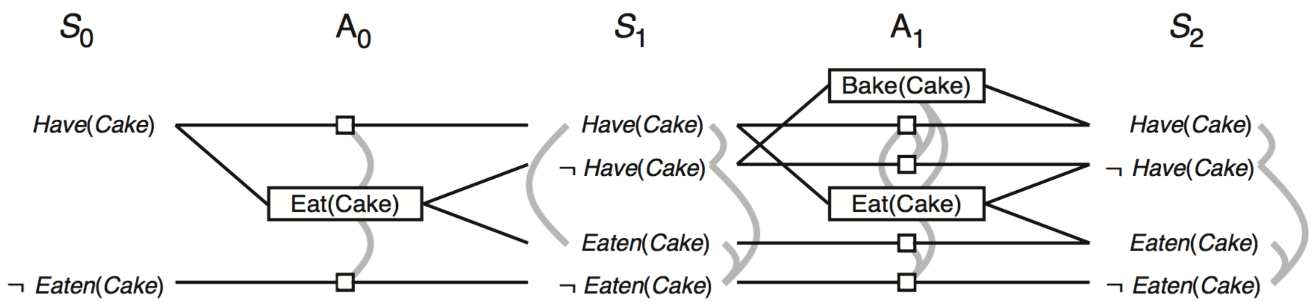
Programs that has won the bi-annual Planning competition has often used
- FF (fast forward) search with heuristics, or
- planning graphs, or
- SAT.

Planning graphs

The main disadvantage of state-space search is the size of the search tree (exponential). Also, the heuristics are not admissible in general.

The planning graph is a polynomial size approximation of the complete tree. Search on this graph is an admissible heuristic.

The planning graph is organized in alternating levels of possible states S_i and applicable actions A_i. Links between levels represent preconditions and effects whereas links within the levels express conflicts (mutex-links).



Planning graphs

A planning problem with l literals and a actions has a polynomial size planning graph:

– Levels S_i contain at most l nodes and l^2 mutex links
– Levels A_i contain at most $a + l$ nodes and $(a + l)^2$ mutex links
– At most $2(al + l)$ links between levels for preconditions and effects
– Therefore, a graph with n levels has size $O(n(a + l)^2)$

The GraphPlan algorithm

The GraphPlan algorithm expands the graph with new levels S_i and A_i until there are no mutex links between the goals. To extract the actual plan, the algorithm searches backwards in the graph.

The plan extraction is the difficult part and is usually done with greedy-like heuristics.

SatPlan and CSP

Translate the PDDL description into a SAT problem or a CSP (constraint satisfaction problem).

The goal state as well as all actions have to be propositionalized. Action schemas have to be replaced by a set of ground actions, variables have to be replaced by constants, fluents need to be introduced for each time step, etc.
$\Rightarrow$ combinatorial explosion

In other words, we remove a part of the benefits of the expressiveness of PDDL to gain access to efficient solution methods for SAT and CSP solvers.

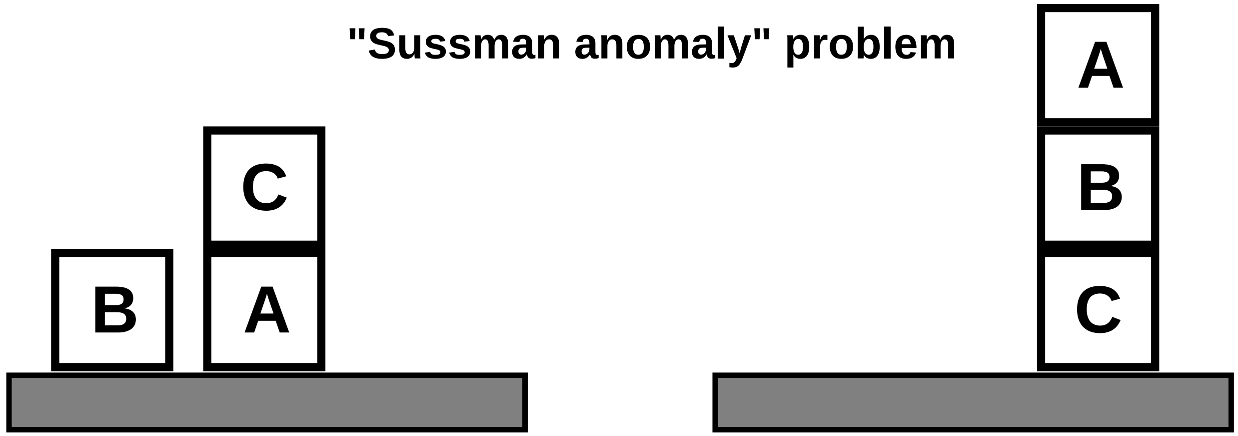
Historical remark: Linear planning

Planners in the early 1970s considered totally ordered action sequences
– problems were decomposed in subgoals
– the resulting subplans were stringed together in some order
– this is called linear planning

But, linear planning is **incomplete**!
– there are some very simple problems it cannot handle
– e.g., the Sussman anomaly
– a complete planner must be able to interleave subplans

Enter partial-order planning, state-of-the-art during the 1980s and 90s
– today mostly used for specific tasks, such as operations scheduling
– also used when it is important for humans to understand the plans
– e.g., operational plans for spacecraft and Mars rovers are checked by human operators before uploaded to the vehicles

Example: The Sussman anomaly



Start State   Goal State

Clear(x) On(x,z) Clear(y)

PutOn(x,y)

~On(x,z) ~Clear(y)
Clear(z) On(x,y)

Clear(x) On(x,z)

PutOnTable(x)

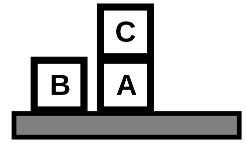
~On(x,z) Clear(z) On(x,Table)

+ several inequality constraints

Example contd.

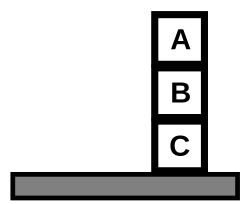
START

On(C,A) On(A,Table) Cl(B) On(B,Table) Cl(C)

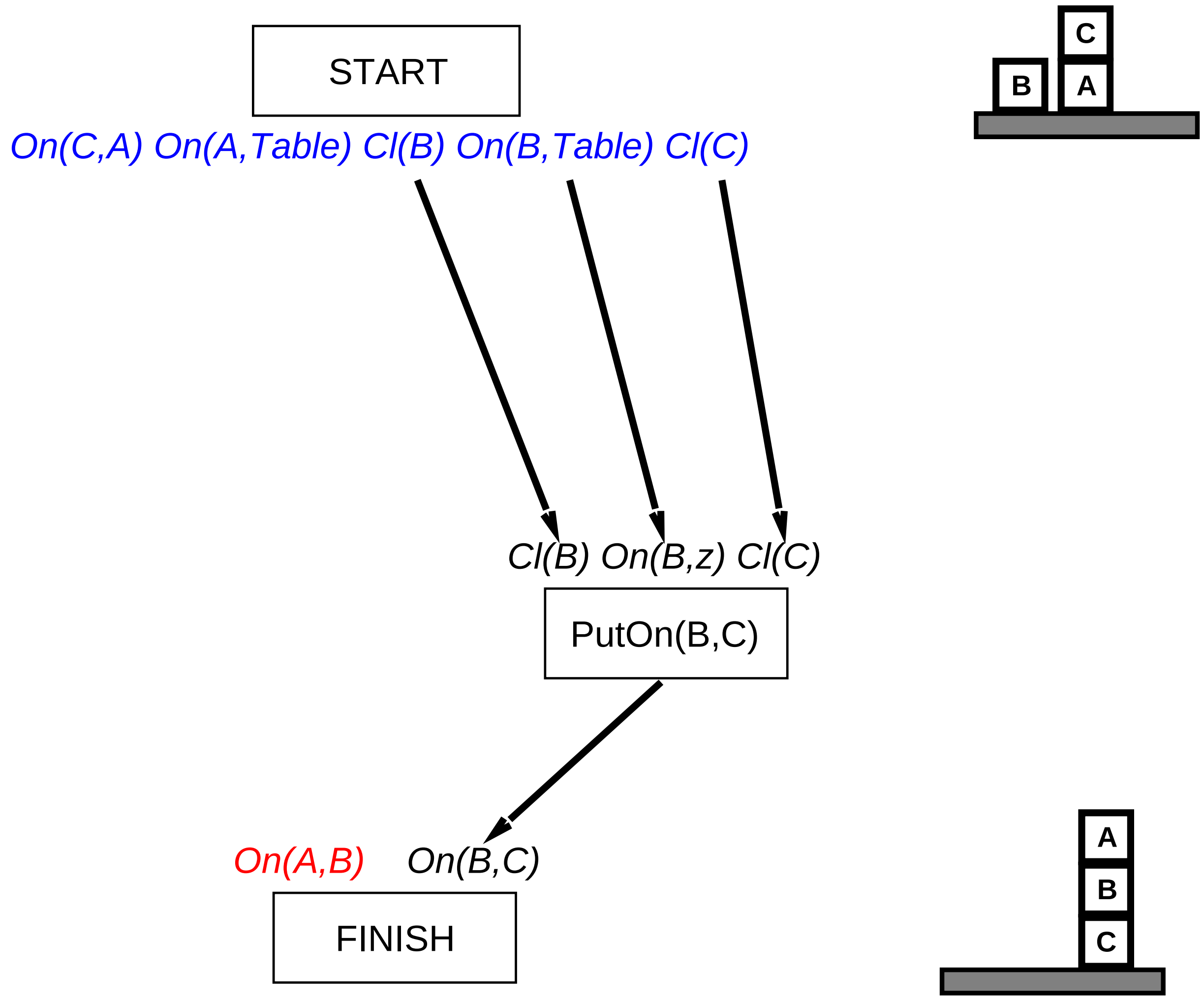


On(A,B) *On(B,C)*

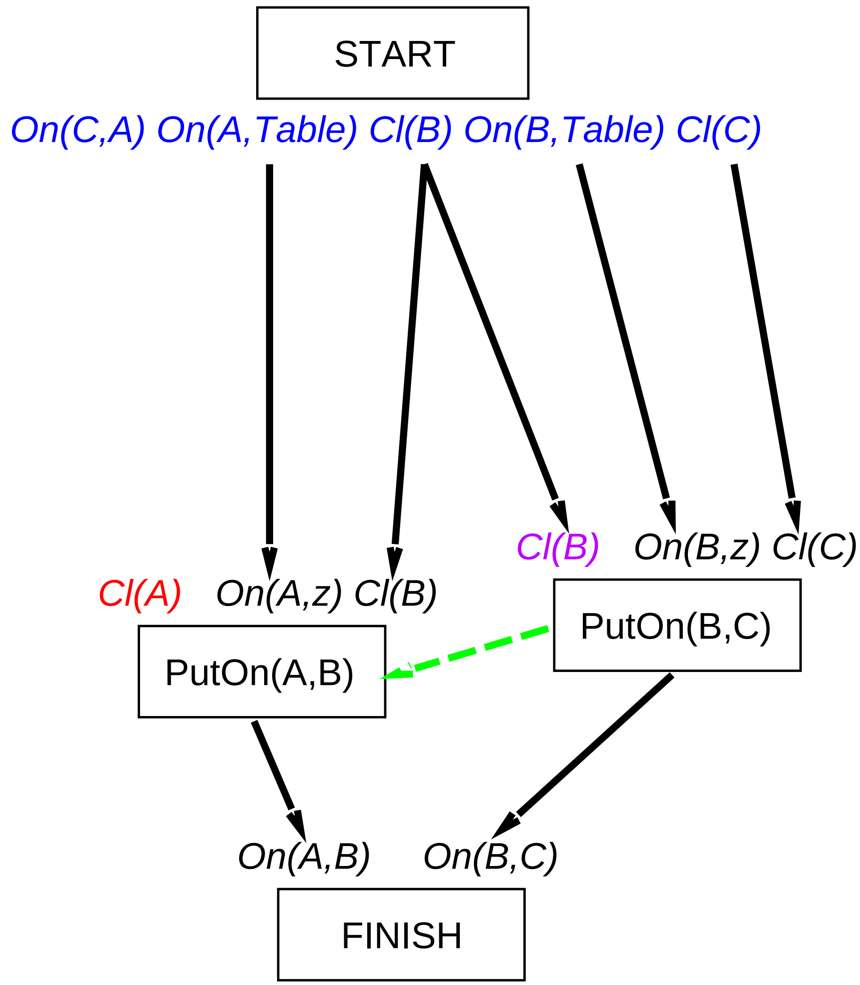
FINISH

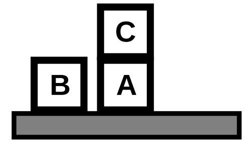


Example contd.

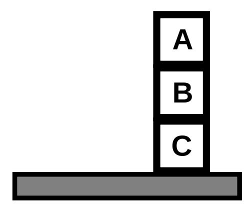


Example contd.

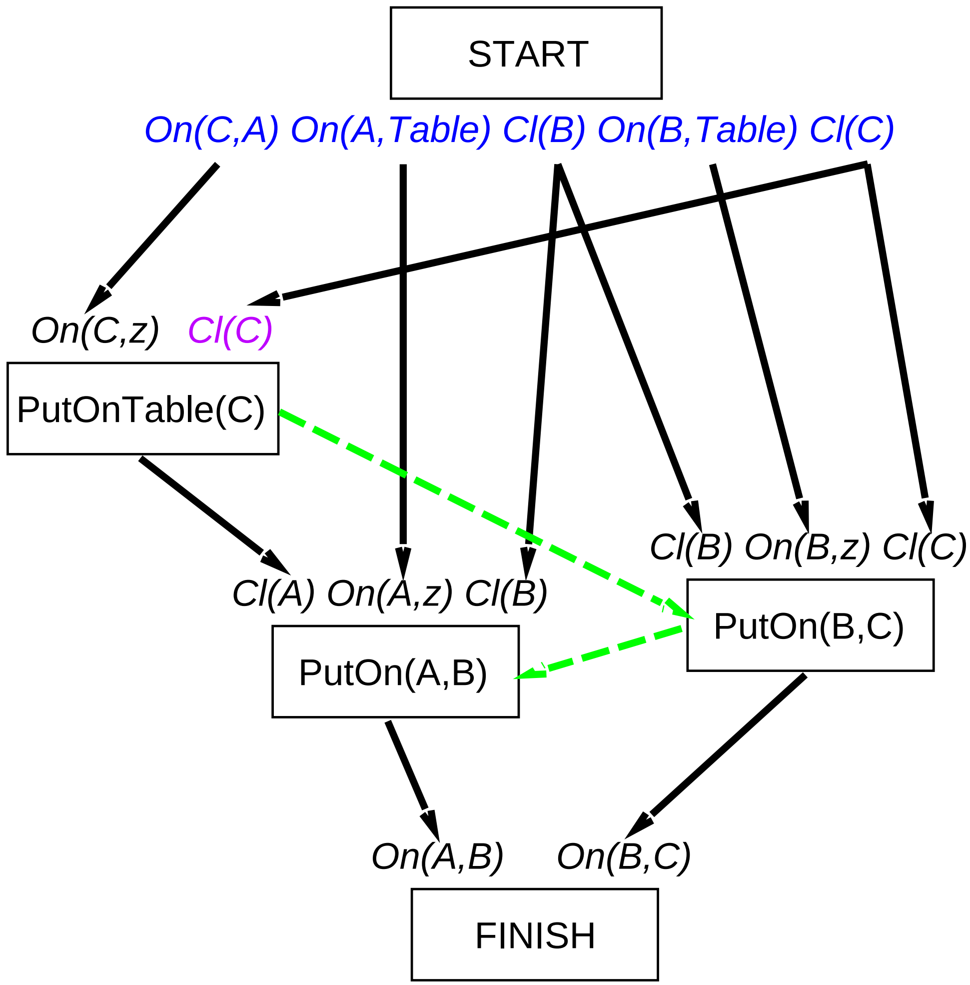


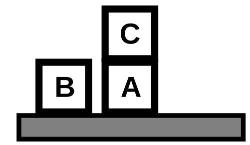


PutOn(A,B)
clobbers Cl(B)
=> order after
PutOn(B,C)



Example contd.





PutOn(A,B) clobbers Cl(B) => order after PutOn(B,C)

PutOn(B,C) clobbers Cl(C) => order after PutOnTable(C)