Properties of forward chaining

- Sound and complete for first-order definite clauses
- **Datalog** = first-order definite clauses + no functions
  - FC terminates for Datalog in finite number of iterations
- May not terminate in general if α is not entailed
- This is unavoidable:
  Entailment with definite clauses is semidecidable!

Efficiency of forward chaining

Source of complexity for FOL-FC-Ask:

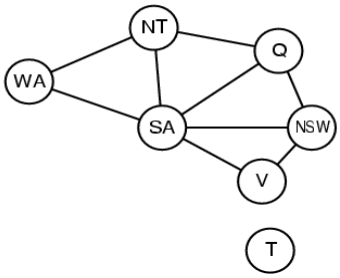
1. Inner loop searches all unifiers ⇒ **pattern matching** is expensive.
2. Every rule is tested again in every iteration.
3. Algorithm may produce many facts not relevant for the goal.

1. <u>Matching rules with known facts:</u>

- To apply *Missile(x)* ⇒ *Weapon(x)*, find all facts which unify with *Missile(x).*
  - Can be done in constant time using indices.
- To apply *Missile(x)* ∧ *Owns(Nono,x)* ⇒ *Sells(West,x,Nono),* we can either
  - first find all objects Nono owns and then test if the objects are missiles or
  - first find all missiles and then test if they are owned by Nono
  - This is the problem of **conjugate order**
    - Choose and order which minimizes overall costs (depends on KB).
    - Use heuristics, e.g., "most constraint variable" aka "minimum remaining value (MRV)"
    - ➢ Pattern matching has **strong relation to CSPs**!
    - ➢ Every conjunct is a constraint for the contained variables.

Hard matching example

- Every CSP with finite domain…



…can be expressed

- <u>as a singe definite clause:</u>

  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()*

- <u>with the according facts:</u>

  Diff(Red,Blue) *Diff (Red,Green)*
  Diff(Green,Red) *Diff(Green,Blue)*
  Diff(Blue,Red) *Diff(Blue,Green)*

➢ *Colorable*() is inferred iff the CSP has a solution,

➢ CSPs include 3SAT as a special case, hence **matching is NP-hard!**

Hard matching

- NP completeness of FC in its inner loop put into perspective:

  a) Most rules in real world knowledge bases are small and simple in contrast to CSP formulation.
     - In DB world:
       - often there are limits for rule sizes and predicate arity
       - Inference complexity is just dependent of number of facts

  b) Generate subclasses of rules which are efficient.
     - Each Datalog clause can be seen as CSP
     - Solve the CSP, if it is a tree it is solvable in linear time.
     - Same can be done for the rules, e.g., delete SA in prior example:

     Diff(wa,nt) $\wedge$ *Diff(nt,q)* $\wedge$ *Diff(q,nsw)* $\wedge$ *Diff(nsw,v)* $\Rightarrow$ *Colorable()*

  c) Avoid redundant rule matches (see following slides)

Efficiency of forward chaining

2. <u>Incremental Forward Chaining</u>

- Simple FOL-FC-Ask would repetitively and redundantly match rules, e.g.
  Missile(x) ⇒ *Weapon(x)* during second iteration
  - Observation: No need to match rule on iteration *k* if a premise wasn't added on iteration *k-1!*
  - ⇒ match each rule whose premise contains a newly added positive literal
  - ⇒ match rule it its premises contain a fact pi which unifies with a fact pi' derived during k-1!
  - ⇒

```
for every iteration k
        for every rule r
                for each p_i in premises(r)
                        for each p_i' derived during k-1
                                if unify(p_i, p_i') ⇒ match(r)
```

- **Database indexing** allows O(1) retrieval of known facts
  - e.g., query *Missile(x)* retrieves *Missile(M_1)*

- Redundancy can be avoided if partial derivations are buffered:
  - Rete-Algorithm: Uses a data-propagation network which propagates variable bindings.
  - Every node is literal from premises.

Efficiency of forward chaining

- Rete and successors were basis for production systems like
  - XCON (DEC) hardware configuration and OPS-5, a general language

  or for cognitive architectures like
  - ACT (Anderson, 1983) or SOAR (Laird et al., 1987)

3. <u>Irrelevant facts</u>

- Deduction of facts not required for a given goal (similar to FC in PL)
- Solutions:
  1. Use of subset of rules (see PL)
  2. From **deductive database** research: Use a **magic set**
     - Only consider rules with a given variable binding:

     Magic(x) ∧ *American(x)* ∧ *Weapon(y)* ∧ *Sells(x,y,z)* ∧ *Hostile(z)* ⇒ *Criminal(x)*
  3. Use of Backward chaining (see next)

Backward chaining algorithm

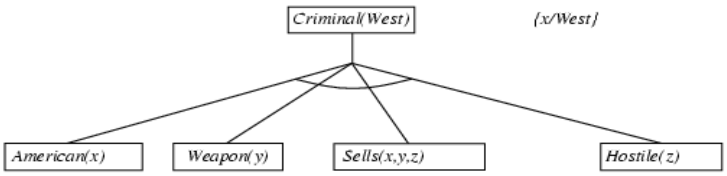
```
function FOL-BC-ASK(KB, goals, θ) returns a set of substitutions
   inputs: KB, a knowledge base
           goals, a list of conjuncts forming a query
           θ, the current substitution, initially the empty substitution { }
   local variables: ans, a set of substitutions, initially empty

   if goals is empty then return {θ}
   q' ← SUBST(θ, FIRST(goals))
   for each r in KB where STANDARDIZE-APART(r) = ( p1 ∧ ... ∧ pn ⇒ q)
           and θ' ← UNIFY(q, q') succeeds
     ans ← FOL-BC-ASK(KB, [p1, ..., pn|REST(goals)], COMPOSE(θ, θ')) ∪ ans
   return ans
```

where SUBST(COMPOSE(θ_1, θ_2), p) = SUBST(θ_2, SUBST(θ_1, p))

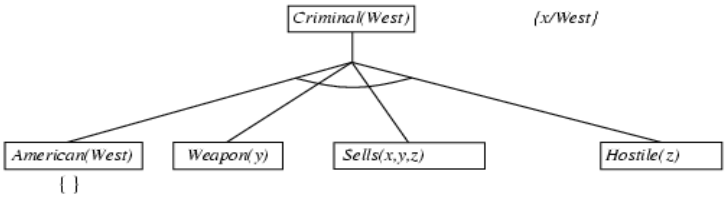
Backward chaining example

Criminal(West)

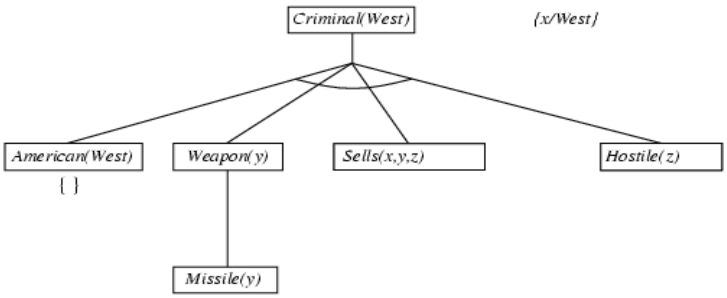
Backward chaining example



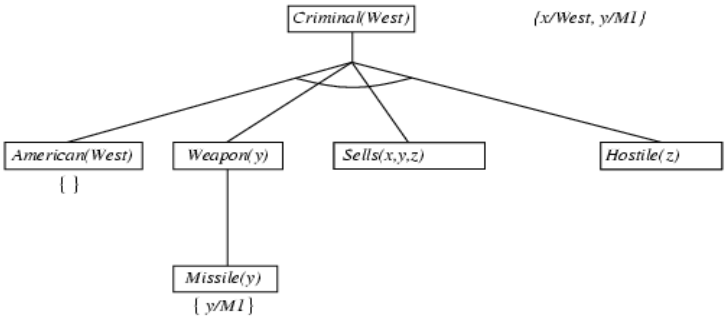
Backward chaining example



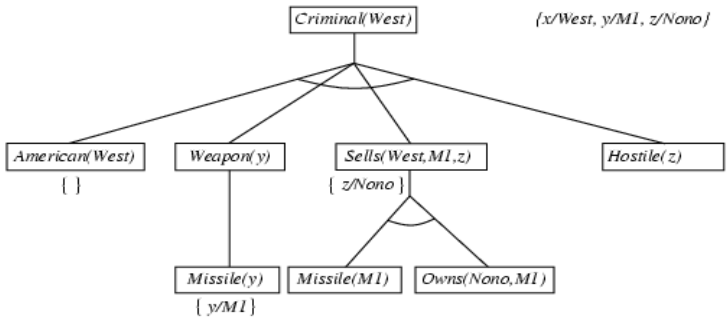
Backward chaining example



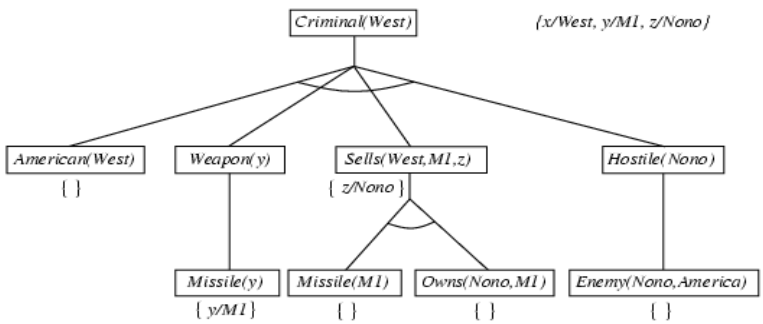
Backward chaining example



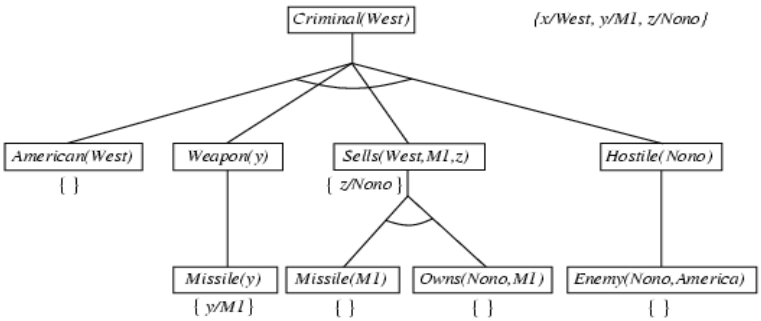
Backward chaining example



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
  - ⇒ fix by checking current goal against every goal on stack…
  - …or?
- Inefficient due to repeated subgoals (both success and failure)
  - ⇒ e.g., fix using caching of previous results (extra space)
- Widely used for **logic programming**

Logic programming: Prolog

- Algorithm = Logic + Control
- Basis: backward chaining with Horn clauses + bells & whistles
  Widely used in Europe, Japan (basis of 5th Generation project)
  Compilation techniques ⇒ 60 million LIPS
- Program = set of clauses = `head :- literal`$_1$`, … literal`$_n$.
  `criminal(X) :- american(X), weapon(Y), sells(X,Y,Z), hostile(Z).`
- Depth-first, left-to-right backward chaining
- Built-in predicates for arithmetic etc., e.g., `X is Y*Z+3`
- Built-in predicates that have side effects (e.g., input and output
- predicates, assert/retract predicates)
- Closed-world assumption ("negation as failure")
  - e.g., given `alive(X) :- not dead(X).`
  - `alive(joe)` succeeds if `dead(joe)` fails

Prolog

- 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=[]
```

Logic programming using Prolog

- Efficiency:
  1. Using or/and parallelism.
  2. Prolog programs can be interpreted or compiled.
  3. Interpreted:
     - In general uses FOL-BC-ASK
     - Prolog generates just one answer and a promise at a **choice point** to generate the rest
     - Prolog implements substitutions using logic variables which remember their binding
       - Current set of var/value bindings reflects substitution in current branch
       - var/value bindings are stored on a stack called **trail**
       - New bindings involves removal from trail and pushing the new binding

Logic programming using Prolog

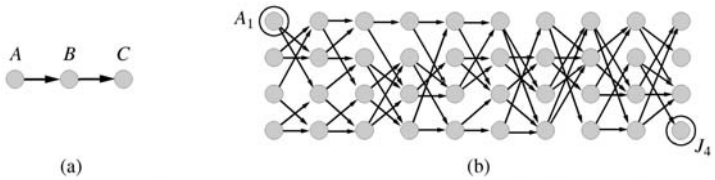
4. Compiled:
   - Instruction set of current CPUs is insufficient with Prolog semantics
   - Use of an intermediate language, e.g., WAM – Warren Abstract Machine
   - Predicates can be translated into subroutines:

```
procedure APPEND(ax, y, az, continuation)

   trail ← GLOBAL-TRAIL-POINTER()
   if ax = [] und UNIFY(y, az) then CALL(continuation)
   RESET-TRAIL(trail)
   a ← NEW-VARIABLE(); x ← NEW-VARIABLE(); z ← NEW-VARIABLE()

   if UNIFY(ax, [a — x]) und UNIFY(az, [a — z])
      then APPEND(x, y, z, continuation)
```

   - No need to search KB for append-clauses
   - storing of bindings on the trail
   - Continuation as choice points to *pack* procedure and parameter list.

Logic programming using Prolog

- Mismatch between depth-first search and search trees
  - Given two graphs (a) from A to C and (b) from A_1 to J_4:



  - Given (a), two Prolog encodings for the graph could be:

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

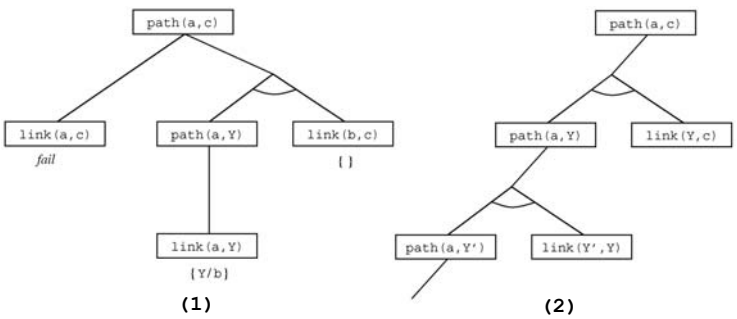
  - or

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

  - and the facts `link(a, b).` `link(b, c).` with the query `path(a, c).`

Logic programming using Prolog

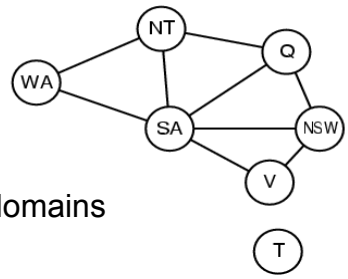
- This results in two different search paths:



- Hence **Prolog is incomplete**!
- Following (b) on prior slide it has problems with redundancy
  - (b) uses 877 inferences and repetitively visits states (see search)
  - FC would only need 62 inferences

Logic programming using Prolog

- Example from state coloring:
- Code the problem in Prolog!



- Backtracking only works for finite domains
- Imagine:

```
triangle(X, Y, Z) :- X>=0, Y>=0, Z>=0, X+Y>=Z, Y+Z>=X, X+Z>=Y.
triangle(3,4,5).  % works????
triangle(3,4,Z).  % works????
```

- Binding a variable to a term is a special form of a constraint.
- CLP (Constraint Logic Programming) allows variables to be constraint and not only to be bound!

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

- The two clauses are assumed to be standardized apart so that they share no variables.
- 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
  $\forall x\ [\neg\forall y\ \neg Animal(y) \vee Loves(x,y)] \vee [\exists y\ Loves(y,x)]$
- 2. Move $\neg$ inwards: $\neg\forall x\ p \equiv \exists x\ \neg p$, $\neg\ \exists x\ p \equiv \forall x\ \neg p$
  $\forall x\ [\exists y\ \neg(\neg Animal(y) \vee Loves(x,y))] \vee [\exists y\ Loves(y,x)]$
  $\forall x\ [\exists y\ \neg\neg Animal(y) \wedge \neg Loves(x,y)] \vee [\exists y\ Loves(y,x)]$
  $\forall x\ [\exists y\ Animal(y) \wedge \neg Loves(x,y)] \vee [\exists y\ Loves(y,x)]$

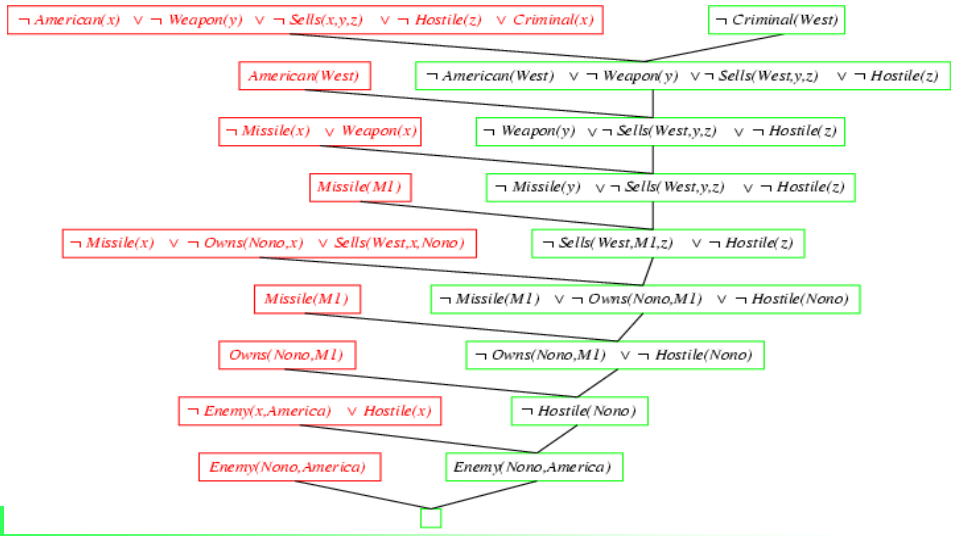
Conversion to CNF contd.

after given after 1. and 2. :

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

3. Standardize variables: each quantifier should use a different one
   $\forall x\ [\exists y\ Animal(y) \wedge \neg Loves(x,y)] \vee [\exists z\ Loves(z,x)]$
4. Skolemize: a more general form of existential instantiation.
   Each existential variable is replaced by a **Skolem function**
   of the enclosing universally quantified variables:
   $\forall x\ [Animal(F(x)) \wedge \neg Loves(x,F(x))] \vee Loves(G(x),x)$
5. Drop universal quantifiers:
   $[Animal(F(x)) \wedge \neg Loves(x,F(x))] \vee Loves(G(x),x)$
6. Distribute $\vee$ over $\wedge$:
   $[Animal(F(x)) \vee Loves(G(x),x)] \wedge [\neg Loves(x,F(x)) \vee Loves(G(x),x)]$

Resolution proof: definite clauses



Completeness proof for FOL resolution

Any set of sentences S is representable in clausal form

↓

Assume S is unsatisfiable, and in clausal form

↓ ← Herbrand's Theorem

Some set S' of ground instances is unsetisfiable

↓ ← Ground resolution theorem

Resolution can find contradiction in S'

↓ ← Lifting lemma

There is a resolution proof for the contradiction in S'