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 \( \alpha \) 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. Matching rules with known facts:
   - To apply \( 	ext{Missile}(x) \Rightarrow 	ext{Weapon}(x) \), find all facts which unify with \( 	ext{Missile}(x) \).
   - Can be done in constant time using indices.
   - To apply \( 	ext{Missile}(x) \land 	ext{Owns}(\text{Nono},x) \Rightarrow 	ext{Sells}(\text{West},x,\text{Nono}) \), we can either
     - first find all objects \( \text{Nono} \) owns and then test if the objects are missiles or
     - first find all missiles and then test if they are owned by \( \text{Nono} \)
   - 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
  - as a single definite clause:
    \[
    \text{Diff}(wa,nt) \land \text{Diff}(wa,sa) \land \text{Diff}(nt,q) \\
    \land \text{Diff}(nt,sa) \land \text{Diff}(q,nsw) \land \text{Diff}(q,sa) \\
    \land \text{Diff}(nsw,v) \land \text{Diff}(nsw,sa) \land \text{Diff}(v,sa) \\
    \Rightarrow \text{Colorable()}
    \]
  - with the according facts:
    - Diff(\text{Red,Blue}) \quad \text{Diff (Red,Green)}
    - Diff(\text{Green,Red}) \quad \text{Diff(Blue,Green)}
    - Diff(\text{Blue,Red}) \quad \text{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:
       \[
       \text{Diff}(wa,nt) \land \text{Diff}(nt,q) \land \text{Diff}(q,nsw) \land \text{Diff}(nsw,v) \Rightarrow \text{Colorable()}
       \]
  c) Avoid redundant rule matches (see following slides)
Efficiency of forward chaining

2. Incremental Forward Chaining

- Simple FOL-FC-Ask would repetitively and redundantly match rules, e.g.
  $\text{Missile}(x) \Rightarrow \text{Weapon}(x)$
  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 if its premises contain a fact $p_i$ which unifies with a fact $p_i'$ 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($p_i$, $p_i'$) \(\Rightarrow\) match($r$)

- Database indexing allows O(1) retrieval of known facts
  - e.g., query $\text{Missile}(x)$ retrieves $\text{Missile}(M_1)$

- Redundancy can be avoided if partial derivations are buffered:
  - Rete-Algorithum: 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. Irrelevant facts

- 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:
       $\text{Magic}(x) \land \text{American}(x) \land \text{Weapon}(y) \land \text{Sells}(x, y, z) \land \text{Hostile}(z) \Rightarrow \text{Criminal}(x)$
  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(θ₁, θ₂), p) = Subst(θ₂, Subst(θ₁, p))

Backward chaining example
Backward chaining example

Backward chaining example

![Diagram showing 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 :- literal₁, ..., literalₙ.
  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=[1,2] 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:

```prolog
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 A₁ to J₄:

```
  A -- B -- C
  \  |  /  \
   \|/   \
    / \  \
```

(a)  

```
  A₁ -- J₄
  \  |  /  \
   \|/   \
    / \  \
```

(b)  

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

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

• Of

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

• and the facts

```prolog
link(a, b).
link(b, c).
```

with the query

```prolog
path(a, c).
```

• This results in two different search paths:

(1)

```
  path(a, c)
  \-- link(a, c)
    \   \  \  \\
     \  \|/ \\
      \|/  \\
       /  \\
```

(2)

```
  path(a, c)
  \-- link(a, c)
    \   \  \  \\
     \  \|/ \\
      \|/  \\n       /  \\
```

• 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:

\[
\text{triangle}(X, Y, Z) :- X \geq 0, Y \geq 0, Z \geq 0, X+Y \geq Z, Y+Z \geq X, X+Z \geq Y.
\]

\[
\text{triangle}(3,4,5). \quad \% \text{ works????}
\]

\[
\text{triangle}(3,4,Z). \quad \% \text{ works????}
\]

- 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:

\[
\frac{\bigvee l_1 \lor \cdots \lor l_k, m_1 \lor \cdots \lor m_n}{\bigvee l_1 \lor \cdots \lor l_i-1 \lor l_{i+1} \lor \cdots \lor l_k \lor m_1 \lor \cdots \lor m_{j-1} \lor m_{j+1} \lor \cdots \lor m_n} \theta
\]

where \( \text{Unify}(l_i, \neg m_j) = \theta \).

- The two clauses are assumed to be standardized apart so that they share no variables.
- For example,

\[
\frac{\neg \text{Rich}(x) \lor \text{Unhappy}(x)}{\text{Rich}(\text{Ken})} \frac{\text{Rich}(\text{Ken})}{\text{Unhappy}(\text{Ken})}
\]

with \( \theta = \{x/\text{Ken}\} \)

- Apply resolution steps to CNF(KB \( \land \neg \alpha \)); complete for FOL
Conversion to CNF

- Everyone who loves all animals is loved by someone:
  \[ \forall x \left[ \forall y \, \text{Animal}(y) \Rightarrow \text{Loves}(x,y) \right] \Rightarrow \exists y \, \text{Loves}(y,x) \]

- 1. Eliminate biconditionals and implications
  \[ \forall x \left[ \neg \forall y \, \neg \text{Animal}(y) \lor \text{Loves}(x,y) \right] \lor \exists y \, \text{Loves}(y,x) \]

- 2. Move \( \neg \) inwards:
  \[ \forall x \left[ \exists y \, \neg \left( \neg \text{Animal}(y) \lor \text{Loves}(x,y) \right) \right] \lor \exists y \, \text{Loves}(y,x) \]

Conversion to CNF contd.

- after given after 1. and 2. :
  \[ \forall x \left[ \exists y \, \text{Animal}(y) \land \neg \text{Loves}(x,y) \right] \lor \exists y \, \text{Loves}(y,x) \]

- 3. Standardize variables: each quantifier should use a different one
  \[ \forall x \left[ \exists y \, \text{Animal}(y) \land \neg \text{Loves}(x,y) \right] \lor \exists z \, \text{Loves}(z,x) \]

- 4. Skolemize: a more general form of existential instantiation. Each existential variable is replaced by a Skolem function of the enclosing universally quantified variables:
  \[ \forall x \left[ \text{Animal}(F(x)) \land \neg \text{Loves}(x,F(x)) \right] \lor \text{Loves}(G(x),x) \]

- 5. Drop universal quantifiers:
  \[ \left[ \text{Animal}(F(x)) \land \neg \text{Loves}(x,F(x)) \right] \lor \text{Loves}(G(x),x) \]

- 6. Distribute \( \lor \) over \( \land \):
  \[ \left[ \text{Animal}(F(x)) \lor \text{Loves}(G(x),x) \right] \land \left[ \neg \text{Loves}(x,F(x)) \lor \text{Loves}(G(x),x) \right] \]
Resolution proof: definite clauses

Any set of sentences $S$ is representable in clausal form

Assume $S$ is unsatisfiable, and in clausal form

Some set $S'$ of ground instances is unsatisfiable

Resolution can find contradiction in $S'$

There is a resolution proof for the contradiction in $S'$

Completeness proof for FOL resolution