

Search Control (for Efficiency)

- Even within Strategy, still decisions:
  When to use which literals/Clauses?

- For SINGLE query: depends on which variables bound / how Structural information:
  “No (extra) answers in this path”

  Conjunct (Rule) Order, How to Backtrack
  Procedural Attachment, Equality

- Consider DISTRIBUTION \( \mathcal{D}_Q \) of queries asked of (fixed) \( KB \)
  + Best FIXED ordering of rules/conjuncts
  + Best FIXED heuristics (“control rules”)
  \( \Rightarrow \) Save part of derivation, for re-use
    Caching
    Explanation-based Learning (macros)
    Direction of Rules
1a. Conjunct Ordering

Eg: “What is income of president's spouse?”
   \(\text{Income}(s,i) \land \text{Married}(s,p) \land \text{Job}(p, \text{President})\)

Prolog: Enumerate all person/income \(\langle s, i \rangle\) pairs
   For each \(s_j\) in \(\langle s_j, i_j \rangle\),
      Find spouse(s) \(p\)
      For each such \(p\), check Job(p, President)

Silly!
   \(\text{Job}(p, \text{President}) \land \text{Married}(s,p) \land \text{Income}(s,i)\)
   much more efficient
   (Only 1 \(p\), then only 1 \(s\), then only one \(i\))

\(\Rightarrow\) MetaReasoning:
   Determine \# of solutions / literal
   seek SMALLEST

“most constraining conjunct first”
   NP-hard, but \(\exists\) good heuristics
   (fewest free variables)
1b. How to Backtrack?

Eg: “Who lives in same town as president?”
   Live(p, town) ∧ Live(x, town) ∧ Job(p, Pres)

Prolog: Enumerate all person/town \( \langle p, \text{town} \rangle \) pairs
   For each \( \text{town}_j \) in \( \langle p_j, \text{town}_j \rangle \),
     For each x S.t. Live(x, town\(_j\))
       Check Job(p\(_j\), Pres)

If fail, take next \( x_2 \) in \( \text{town}_j \), ...

SILLY:
   If \( \neg \text{Job}(p_j, \text{Pres}) \), should take NEXT town!

ie, backtrack to 1st literal, not 2nd

- Problem: Chronological backtracking

Better: Backjumping
   Which variable led to problem?
   Goto literal which sets that variable.

- “Dependency Directed Backtracking”
  Store combination of variables that led to dead-end
2a. Re-Using Information Over Distribution of Queries

- Cache (then re-use) Results
  Eg: cache (Aunt J E)

- Cache (then re-use) Rule-chains
  – Used \( \langle R8, R2, R6, R5 \rangle \) to solve (Aunt J E).
  – “Chain together” these rule into:
    \[
    R_n : (Mo \ k \ m) \land (Mo \ m \ gm) \land (Mo \ a \ gm) \\
    \land (Fl \ a) \land (Unk \ (= \ m \ a)) \\
    \Rightarrow (Aunt \ k \ a)
    \]

- “Derivation Path Heuristic”
  Both \( R2 \) and \( R3 \) reduce (Pa \( \gamma_k \ p \)) goal.
  Spse \( R3 \) succeeded prior 200 times, and \( R2 \) failed?
    Suggests only Fathers; so . . .
    Try \( R3 \) first, next time!
2b. When to do what?

• Reasoning Agent
  + is Telled info
  + is Asked questions
    ... based on info.

• Here (like Prolog):
  + Tell trivial: simple storage
  + Ask does ALL the work.

  “Backward Chaining”

• But... (Production System):
  + Ask is trivial: check current $KB$
  + Tell does all the work

  “Forward Chaining”

• Which is better?
  ... Branching factors

  + Mixed strategies
Forward Chaining

• Compute AUTOMATICALLY
  Rather than wait for a question.

• Useful when
  (1) Same question will be posed many times.
  (2) single query is expensive:
      large (disjunctive) BACKWARD branching.

Recall Search Space can be
  – Exponential Backwards
  – Linear Forwards
Forward vs Backward Chaining

Given: zebra(zeke)
Query: animal(zeke) ?

$KB_1$

\[
\begin{align*}
\text{ant}(X) & \Rightarrow \text{insect}(X) \\
\text{bee}(X) & \Rightarrow \text{insect}(X) \\
\text{spider}(X) & \Rightarrow \text{insect}(X) \\
\text{insect}(X) & \Rightarrow \text{animal}(X) \\
\text{lion}(X) & \Rightarrow \text{mammal}(X) \\
\text{tiger}(X) & \Rightarrow \text{mammal}(X) \\
\text{zebra}(X) & \Rightarrow \text{mammal}(X) \\
\text{mammal}(X) & \Rightarrow \text{animal}(X)
\end{align*}
\]

$KB_2$

\[
\begin{align*}
\text{zebra}(X) & \Rightarrow \text{medium}(X) \\
\text{zebra}(X) & \Rightarrow \text{striped}(X) \\
\text{zebra}(X) & \Rightarrow \text{mammal}(X) \\
\text{medium}(X) & \Rightarrow \text{nonSmall}(X) \\
\text{medium}(X) & \Rightarrow \text{nonLarge}(X) \\
\text{striped}(X) & \Rightarrow \text{nonSolid}(X) \\
\text{striped}(X) & \Rightarrow \text{nonSpot}(X) \\
\text{mammal}(X) & \Rightarrow \text{animal}(X) \\
\text{mammal}(X) & \Rightarrow \text{warm}(X)
\end{align*}
\]
**Forward vs Backward Chaining**

**FC:** $KB \rightarrow KB'$
- Finds other truths
  - No specific query/objective/goal
  - Might conclude irrelevant statements

**BC:** $KB \times \sigma \rightarrow \{T, F\}$ (w/binding lists)
- Determines if query is true
  - Might follow false leads
    - Prolog
    - Q/Aing, Diagnosis Tasks

≠ Order of Search
- I.e., which rules to use, ...
Mixed Forward & Backward Chaining

- Label each rule as FC or BC or both
  (Eg: \( R_1, R_2, R_3 \) all FC; others are BC.)

- If all rules matching (if \( \text{if } ? \text{' } ,p \) are FC, as are rules for antecedents, then use Fetch for \( p \):
  (MetaTell '(ToAnswer ,p Fetch))
  [Otherwise: full BCs]

- Eg: All (Pa \( \cdots \)) facts ALREADY present
  (MetaTell '(ToAnswer (Pa &k &p) Fetch))

- In general, specify:
  - Rule: FC and/or BC?
  - Ground clause: whether to FC when Telled? \( \cdots \) or just store?
  - Query: whether to BC on asked query? \( \cdots \) or just Fetch?