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")
  ⇒ 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?"
`Income(s,i)` ∧ `Married(s,p)` ∧ `Job(p, President)`

Prolog: Enumerate all person/income ⟨s, i⟩ 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!

`Job(p, President)` ∧ `Married(s,p)` ∧ `Income(s,i)`

much more efficient

(Only 1 `p`, then only 1 `s`, then only one `i`)

⇒ MetaReasoning:
Determine #of solutions / literal
seek SMALLEST

"most constraining conjunct first"
NP-hard, but ∃ 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 ⟨p, town⟩ pairs
For each $town_j$ in $\langle p_j, town_j \rangle$,
For each x s.t. `Live(x,` $town_j$`)`
Check `Job(`p_j`, Pres)`

If fail, take next x_2 in $town_j$, . . .

SILLY:

If ¬`Job(`p_j`, 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 ⟨*R8*,*R2*,*R6*,*R5*⟩ to solve `(Aunt J E)`.
  – "Chain together" these rule into:

  R_n : `(Mo k m)` ∧ `(Mo m gm)` ∧ `(Mo a gm)`
  ∧ `(Fl a)` ∧ `(Unk (= m a))`
  ⇒ `(Aunt k a)`

  – "Chunking", "Explanation-Based Learning", . . .

- "Derivation Path Heuristic"
  Both *R2* and *R3* reduce `(Pa` γ_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

```
ant(X)     ⇒ insect(X)
bee(X)     ⇒ insect(X)
spider(X)  ⇒ insect(X)
insect(X)  ⇒ animal(X)
lion(X)    ⇒ mammal(X)
tiger(X)   ⇒ mammal(X)
zebra(X)   ⇒ mammal(X)
mammal(X)  ⇒ animal(X)
    ...
```

KB_2

```
zebra(X)   ⇒ medium(X)
zebra(X)   ⇒ striped(X)
zebra(X)   ⇒ mammal(X)
medium(X)  ⇒ nonSmall(X)
medium(X)  ⇒ nonLarge(X)
striped(X) ⇒ nonSolid(X)
striped(X) ⇒ nonSpot(X)
mammal(X)  ⇒ animal(X)
mammal(X)  ⇒ warm(X)
    ...
```


Forward vs Backward Chaining

FC: $KB \mapsto KB'$

Finds other truths

+ No specific query/objective/goal

– Might conclude irrelevant statements

$\left\{ \begin{array}{l} \texttt{OPS} \\ \text{Interpretation Tasks} \end{array} \right.$

BC: $KB \times \sigma \mapsto \{\mathcal{T}, \mathcal{F}\}$ (w/binding lists)

Determines if query is true

– Might follow false leads

$\left\{ \begin{array}{l} \texttt{Prolog} \\ \text{Q/Aing, Diagnosis Tasks} \end{array} \right.$

$\neq$ Order of Search

I.e., which rules to use, ...

Mixed Forward & Backward Chaining

- Label each rule as *FC* or *BC* or both
  (Eg: *R1*, *R2*, *R3* all *FC*; others are *BC*.)

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

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

- In general, specify:
  - Rule: *FC* and/or *BC*?
  - Ground clause: whether to *FC* when `Telled`?
    ... or just store?
  - Query: whether to *BC* on `Asked` query?
    ... or just *Fetch*?