RN, Chapter 10

Implemented Systems

Logical Agents

- Reasoning [Ch 6]
- Propositional Logic [Ch 7]
- Predicate Calculus
  - Representation [Ch 8]
  - Inference [Ch 9]
- **Implemented Systems [Ch 10]**
  - DataBase Systems
  - Prolog + Extensions (MRS)
  - General Theorem Provers
  - Frame Systems
  - Description Languages
  - Truth Maintenance – Retractions
- Planning [Ch 11]

Properties of Derivation Process

- $\vdash_\alpha$ is <u>SOUND</u>

  $$\Sigma \vdash_\alpha \Psi \quad \Rightarrow \quad \Sigma \models \Psi$$

  Produces only "true" results

- $\vdash_\alpha$ is <u>COMPLETE</u>

  $$\Sigma \vdash_\alpha \Psi \quad \Leftarrow \quad \Sigma \models \Psi$$

  Produces all "true" results

- $\vdash_\alpha$ is <u>DECIDABLE</u>

  $\Sigma \overset{?}{\vdash_\alpha} \Psi$ returns Y or N in finite time

Degenerate $\vdash_\alpha$

- For any $\Sigma, \Psi \subset$ WFFs:

  $\Sigma \not\vdash_N \Psi$

  $\Sigma \vdash_P \Psi$

- Notice:

Degenerate $\vdash_\alpha$

- For any $\Sigma, \Psi \subset$ WFFs:

  $\Sigma \nvdash_N \Psi$

  $\Sigma \vdash_P \Psi$

- Notice:
  - $\vdash_N$ is SOUND
    (everything it returns is logically entailed)
  - $\vdash_P$ is COMPLETE
    (it returns everything logically entailed)
  - $\vdash_N$, $\vdash_P$ are DECIDABLE
    (answer every question)

Fundamental Limitation

- For any sufficiently complicated domain (as complex as arithmetic)
- NO $\vdash_\alpha$ can be **SOUND, COMPLETE, DECIDABLE**!!
- . . . related to “Halting Problem”
- Not Predicate Calculus' fault: Reasoning is inherently undecidable, no manner what formalism used.

Responses

- Deals only with WORST-Case!
  "Typical" case can be better

TradeOffs (to increase efficiency):

- ? Sacrifice **SOUNDness**?
  ? Very severe ??
- ? Sacrifice **COMPLETEness**?
  Reasonable… Specific proposals:
  - Use only (incomplete set of) Inference Rules
  - Use complete set of Inference Rules,
    but limit depth (stop applying rules…)
- ? Sacrifice **EXPRESSIVEness**?
  [EXPRESSIVEness ≈ what can be distinguished]
  Common approach!
  (After all, Logic's distinctions caused problems!)
  - Disallow "$\vee$" "$\neg$" "$\exists$" …

Implemented Systems

- **DataBase Systems**
  ≈Sound, Complete, Limited Expressiveness
- **Prolog**
  ≈Sound, Complete, Limited Expressiveness
  + Extensions: Constraints, MetaLevel aspects:
    Control, Procedural Attach, Equality, Caching, Direction
- **General Theorem Provers**
  Sound, Complete, Complete Expressiveness
- **Production System** (Emycin, OPS)
  ≈Sound, ≈Complete, Limited Expressiveness
- **Frame Systems**
  ?Sound?, ?Complete?, Limited Expressive
- **Description Languages**
  Sound, Complete, Limited Expressiveness
- **Truth Maintenance – Retractions**

DataBase Systems

Comp.	Product	Cost/Unit
McD	BigMac	$1.35
McD	FrenchFry	$0.80
McD	Ketchup	$0.01
BurKing	FrenchFry	$0.80
Lego	Blocks	$15.00
Army	Tanks	$50,000.00
Army	Bullets	$1.00
Navy	Destroyers	$100,000,000.00
Navy	Torpedos	$1000.00

$\approx$

```
Makes( mcD, bigMac)          ∧  CostUnit( bigMac, $1.35)
Makes( mcD, frenchFry)       ∧  CostUnit( frenchFry, $0.80)
Makes( mcD, ketchup)         ∧  CostUnit( ketchup, $0.01)
Makes( lego, blocks)         ∧  CostUnit( blocks, $15)
Makes( army, tanks)          ∧  CostUnit( tanks, $50K)
Makes( army, bullets)        ∧  CostUnit( bullets, $1)
Makes( navy, destroyers)     ∧  CostUnit( destroyers, $100M)
Makes( navy, torpedos)       ∧  CostUnit( torpedos, $1000)
```

Comments on DataBase Systems

- Basically set (&) of Positive Ground Atomic Literals

- Hard to Express Partial Knowledge
  - "FooBarInc makes either bicycles or rockets"
  - "FooBarInc does not make torpedos"
  - "FooBarInc does make something"
  - "Some company makes LegBands"
  - "Bicycles cost between $100 and $1000"

- No (explicit) General Knowledge
  - "Every large company has a president."
  - "Every large company has a president with salary over $500,000"

- Efficient Reasoning
  . . . as Reasoning Fetch + And + Or
  [Cost metric is # of swaps, not retrievals. . .]

Standard DB Assumptions

- **Closed World Assumption**
  - **Q**: "*Does McDonalds makes Tanks?"*
  - **A**: No.

  . . . is really "Unknown". But CWA:
    - *If σ is a positive literal that could be in DB, but σ is not in DB, then conclude $\neg\sigma$*

  EG: As makes(mcD tanks) $\notin$ DB, conclude $\neg$makes(mcD tanks)

  So, unknown(σ) means $\neg\sigma$!
- **Unique Names Assumption**
  - **Q**: "How many companies make FrenchFry?"
  - **A**: 2.

  . . . could be 1, if McD=BurgKing !
  - But UNA:

    different names refer to different things.
- So... "How many products does McDonalds make?"
  - < 2 unless UNA
  - > 3 unless CWA


MOM SAID I CAN'T GO OUTSIDE UNTIL I FINISH MY HOMEWORK. IF YOU'LL HELP ME, I'LL BE DONE FASTER. WHAT'S FIVE PLUS SEVEN?
I DON'T KNOW.
I DON'T EITHER.
THEN WRITE, "I DON'T KNOW."
HEY, THAT'S A TRUE ANSWER, ISN'T IT! I CAN WRITE THAT FOR ALL OF THESE! WE'RE DONE!
WE'D BETTER HAVE A LOOK AT OUR PRODIGY'S HOMEWORK

Prolog

- Refutation Resolution

  so Sound, Complete ... but ...
- Only deals with Horn clauses

  (≤1 positive literal per clause ⇒ no REASONING by Cases)
- Fixed Search Strategy:
  - Depth first: Prefer resolvant from prior step
  - Left to Right on conjunctions (antecedents)
  - Then chronological, by input (FIFO)
- Efficient in general, but ∞ loops, . . .
- Only Backward Chaining

  (No Forward Chaining)

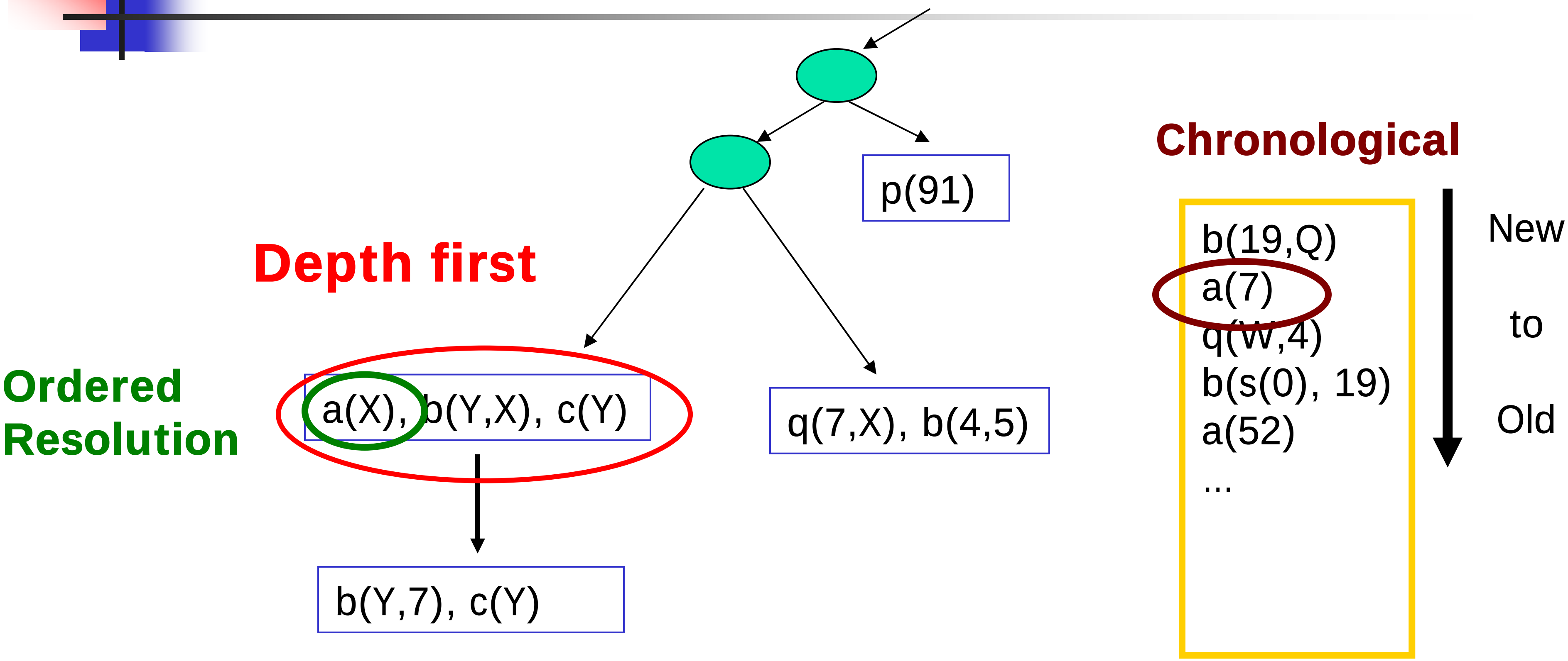
  No Meta-Level control
- Difficult to cache, re-use justification, . . .

Prolog's Decisions

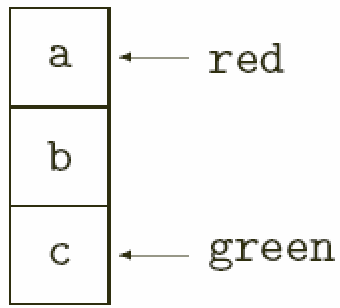
Resolution:
Find two clauses w/"matching literals" and smash them together.

- Q1: *Which two clauses?*
  A1: Set-of-Support (Backward)
  One clause is query, or descendant;
  Other is from KB
- Q2: At any time, "Frontier" of Subgoals. Which one to work on?
  A2: DEPTH-FIRST.
- Q3: *Within subgoal* $G = \{ f_1, \ldots, f_k \}$*, which literals?*
  A3: Ordered resolution ... just f_1
- Q4: *Which rule/fact?*
  A4: Chronological!

Derivation Process...



Limitation of Horn Clauses



on(a, b)
on(b, c)
red(a)
green(c)
$\forall X.\,\mathtt{red}(X) \vee \mathtt{green}(X)$

- Question: ∃ X, Y: on(X, Y) & red(X) & green(Y) ?
- Yes:
  - b is either red or green
  - If b is red, then { X/b, Y/c }
  - If b is green, then { X/a, Y/b }
- . . . not in Prolog..

  Cannot express ∀X: red(X) v green(X)

Efficiency Tricks

- No *OccursCheck* ... so p(X) unfies w/ p(f(X))
  Why?
  - Unification w/o OccursCheck is *O(n)* (n = size of clause)
  - Unification w/ OccursCheck is $O(n^2)$
- ⇒ Too many clauses match
  ⇒ Too many conclusions reached
  ⇒ may conclude [] incorrectly ⇒ Not sound!
- Compilation... avoid explicit run-time “Fetch”
- Parallelism:
  - OR-parallelism: different rules
  - AND-parallelism: different literals w/in rule
- Direct binding (single value/variable, on path)

⇒ > 1M LIPS (Logical Inference per Second)

Prolog's "Impurities"

- **Negation as Failure**

```
%% Knowledge Base:
bach(X) :- male(X), not( married(X) ).
male(fred).
%% Query:
?- bach(fred).
Yes
```

- Prolog tries to prove "married(fred)" and fails.

  So concludes "not(married(fred)) "
- Control Information – "Cut" !
  - Tells Prolog NOT to backtrack
  - Complicated to explain… see Cmput325

Extensions to Prolog #I: Constraints

- Constraint Logic Programs

  triAng(X,Y,Z) :- (X>0), (Y>0), (Z>0),
  (X+Y>Z), (Y+Z>X), (X+Z>Y)

  Use#1: confirm triAng(3,4,5).

  Use#2: Constrain triAng(X,4,5)?

  Prolog: fail

  But. . . { X > 1 & X < 9}

- Later use, to constrain other predicates

  triAng(X,4,5) & prime(X) & …

  triAng(X,4,5) & X>100 & …

Extensions to Prolog #II: Search Control (for Efficiency)

- Even within Resolution 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 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

- "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 ⟨ S_j, I_j ⟩ ,

  Find spouse(s) P

  For each such P, check job(P, president)

10^6 pairs?

- Silly!

  job(P, president) & married(S,P) & income(S,I)

  is 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?

- "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 ⟨ P_j, $Town_j$ ⟩ ,

  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 that sets that variable

- "Dependency Directed Backtracking"
  Store combination of variables that led to dead-end

1c. How to Compute (> 174 50)?

- Challenge: Determine truth of (> 174 50)
- Option 1: Explicitly store

    (> 51 50) (> 52 50) (> 53 50) (> 54 50)
    (> 55 50) (> 56 50) (> 57 50) (> 58 50)

    (> 173 50) (> 174 50) (> 175 50) (> 176 50)

    (> 2021 50) (> 2022 50) (> 2023 50) (> 2024 50)

  and negative facts:
  ¬(> 41 50) ¬(> 42 50) ¬(> 43 50) ¬(> 44 50)
  …
  as well as
  (> 1 0) (> 2 0) (> 3 0) (> 4 0) …
  (> 2 1) (> 3 1) (> 4 1) …
  (> 3 2) (> 4 2) …
  (> 4 3) …
  …
- Requires ∞ storage!
- Is there a better way?

Option 2: Procedural Attachment

- To compute (> x y) ,

  Use procedure *FetchGT*

  where *FetchGT* returns Yes or No
- FetchGT(σ: proposition)

  if second[σ] > third[σ] then "yes"
  else "no"

Eg: -> (FetchGT '(> 174 50))

yes

-> (FetchGT '(> 23 41))

no

Procedural Attachment: +

- Find w s.t. (+ 10 65 w)
- Explicit storage: ∞ space!
- Procedure:

  To compute (+ 10 65 w) , use procedure *FetchPlus* where FetchPlus returns appropriate binding list:

  FetchPlus(σ : proposition)
  ```
     (Match (cadddr σ) (+ (cadr σ) (caddr σ)) )
    ;;;          w          10       65
  ```
  (FetchPlus (+ 10 65 w)) → YES ... w/75
  (FetchPlus (+ 10 65 75)) → yes
  (FetchPlus (+ 10 65 921)) → no

- MRS Solution:
  - MetaTell (ToFetch (> &x &y) FetchGT)
    MetaTell (ToFetch (+ &x &y &z) FetchPlus)
  - MetaTell (relnproc > >)
    MetaTell (funproc + +)

Procedural Attachment

- **Why?** (Space) inefficient to store explicitly.
- **What?** Use procedure to solve query.
- **Constraints**: Sound procedure

    ?Only some bound-sets (directions)?
- **Eg**: <, +, Sort, . . .
- **Gen'l**: MRS allows user to define how to answer arbitrary Asked proposition

1d. Dealing with Equality

- Given axioms

  russ = profG

  happy(russ)

  poor(profG)

  confused(X) :- happy(X), poor(X).

- Expect to conclude

  confused(russ)

- Prolog would not:
  - Reduce confused(russ) to poor(russ) ,
  - but not match poor(russ) w/ poor(profG) .
- ? Could add rule:

  poor(Y) :- poor(X), Y=X.

Comments on Equality

```
russ = profG.   happy(russ).   poor(profG).
confused(X) :- happy(X), poor(X).
```

- Need rule for each relation, function, . . .
- Rule

  `poor(Y) :- poor(X), X=Y.`

  would *NOT* work

  Reduce `confused(russ)` to `profG = russ`,
  NOT in knowledge base!

Fix:

```
X=Y :- Y=X.
X=X.
X=Z :- X=Y, Y=Z.
```

But. . .

```
poor(billGates)
  poor(russ), russ=billGates.
    russ=billGates
     billGates=russ
      russ=billGates
       billGates=russ
        ...
```

or worse:

```
russ=billGates
 russ=Y, Y=billGates
  profG=billGates
   profG=Y, Y=billGates
    Y=profG, Y=billGates
     russ=profG, russ=billGates
      russ=billGates
       ...
```

- Sol'n: Need lots of control rules!

Wrap-Up wrt Equality

Note: f(A) does NOT unify with f(B),
even if A = B

Eg: Father(Russ) = Leonard
MorningStar = Venus
2+2 = 4

Option#1: View "=" as std predicate

$$\forall x: \; x = x$$
$$\forall x, y: \; x = y \;\Rightarrow\; y = x$$
$$\forall x, y, z: \; x = y \;\wedge\; y = z \;\Rightarrow\; x = z$$

But also need...

$$\forall x, y: \; x = y \;\Leftrightarrow\; P_1(x) = P_1(y)$$
$$\forall x, y: \; x = y \;\Leftrightarrow\; P_2(x) = P_2(y)$$
$$\ldots$$
$$\forall x_A, x_B, y_A, y_B: \; x_A = x_B \wedge y_A = y_B \;\Leftrightarrow\; F_1(x_A, y_A) = F_1(x_B, y_B)$$
$$\ldots$$

for every predicate

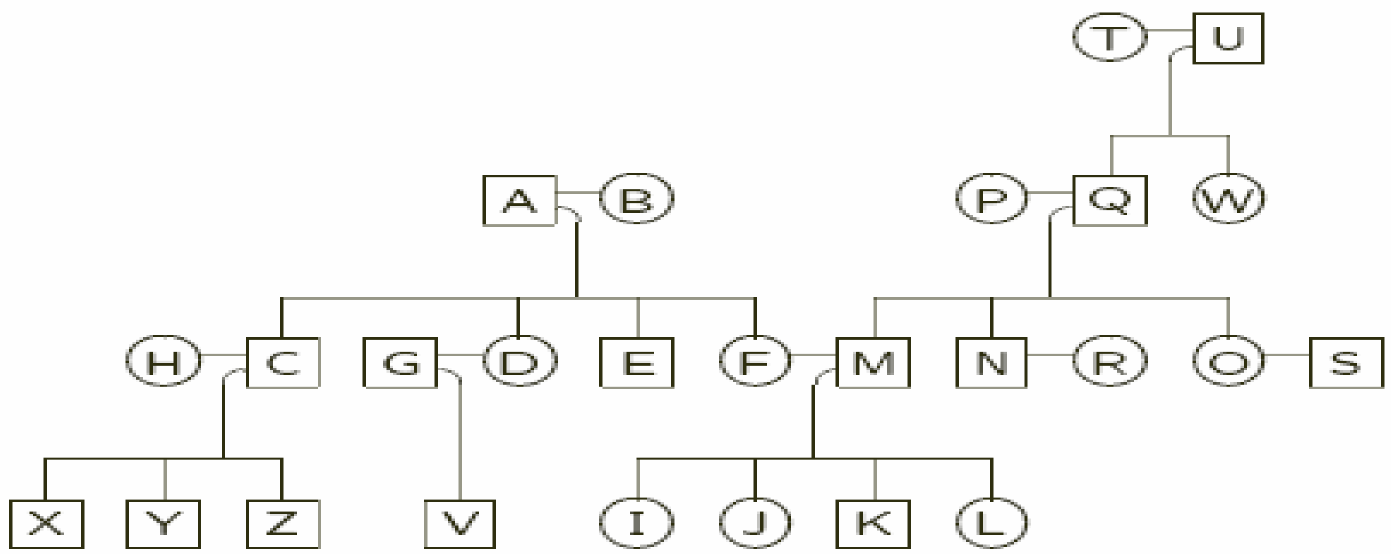
+ search control problems...

Demodulation: For any terms x, y, z where
Unify$(x, z) = \theta$:

$$\frac{x = y, \; (\ldots z \ldots)}{(\ldots \text{Subst}(\theta, y) \ldots)}$$

Paramodulation: ...do not know $x = y$,
but only "$x = y \;\vee\; P(x)$"

Family Primitive Relationships



female(F)	F is FEMALE.
male(M)	M is MALE.
mo(E, M)	M is the MOTHER of E.
husband(W, H)	H is the HUSBAND of W.

Definition of (Other) Relations

R1: pa(C, P) :- mo(C, P).
R2: pa(C, P) :- fa(C, P).
R3: sib(E, S) :- pa(E, M), pa(S, M), not(E = S).
R4: sis(E, S) :- sib(E, S), female(S).
R5: bro(E, B) :- sib(E, B), male(B).
R6: aunt(C, A) :- pa(C, P), sis(P, A).
R7: aunt(C, A) :- pa(C, P), bro(P, U), husband(A, U).

female(F)	F is Female.
male(M)	M is Male.
mo(E, M)	M is the MOTHER of E.
husband(W, H)	H is the HUSBAND of W.
fa(E, F)	F is the FATHER of E.
sis(E, S)	S is a SISTER of E.
bro(E, B)	B is a BROTHER of E.
sib(E, S)	S is a SIBLING of E.

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 ⟨R6, R1, R4, R3, R1, R1⟩ to solve aunt(j, e)
  - "Chain together" these rule into:

    Rn: aunt(E, A) :- mo(E, M), mo(M, GM), mo(A, GM), female(A), not(M = A).
  - "Chunking", "Explanation-Based Learning", . . .
- "Derivation Path Heuristic"
  - Both R1 and R2 reduce (Pa k p) goal.
  - Spse R2 succeeded prior 200 times, and R1 failed?

    Suggests only Fathers; so. . . Try R2 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

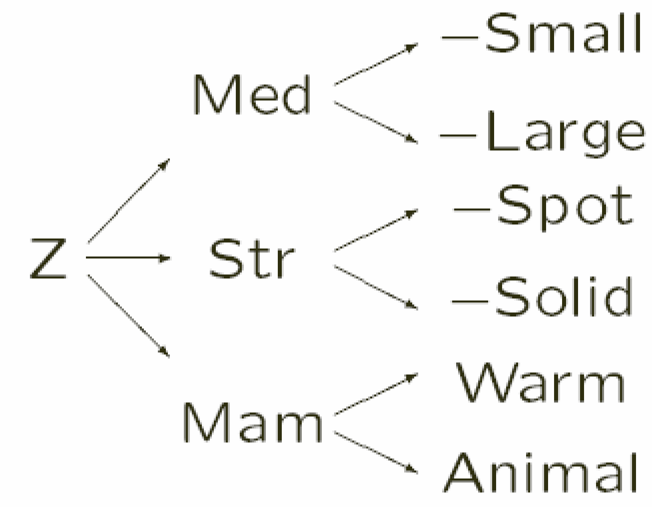
Forward vs Backward Chaining

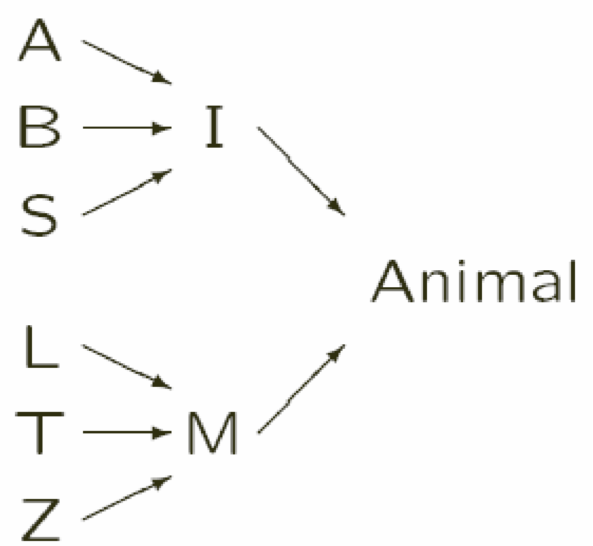
Query: Animal ?

	FC	BC
KB_1	9	**2**
KB_2	**2**	8

KB_1
- Zebra
- Zebra $\Rightarrow$ Medium
- Zebra $\Rightarrow$ Striped
- Zebra $\Rightarrow$ Mammal
- Medium $\Rightarrow$ NonSmall
- Medium $\Rightarrow$ NonLarge
- Striped $\Rightarrow$ NonSolid
- Striped $\Rightarrow$ NonSpot
- Mammal $\Rightarrow$ Animal
- Mammal $\Rightarrow$ Warm
- ...

KB_2
- Zebra
- Ant $\Rightarrow$ Insect
- Bee $\Rightarrow$ Insect
- Spider $\Rightarrow$ Insect
- Insect $\Rightarrow$ Animal
- Lion $\Rightarrow$ Mammal
- Tiger $\Rightarrow$ Mammal
- Zebra $\Rightarrow$ Mammal
- Mammal $\Rightarrow$ Animal...





Forward vs Backward Chaining

- **FC: KB ↦ KB'**
  - Finds other truths
  - No specific query/objective/goal
  - Might conclude irrelevant statements
  - OPS, Interpretation Tasks
- **BC: KB x σ ↦ { 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: *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 $\cdots$) facts ALREADY present
  `(MetaTell '(ToAnswer (Pa &k &p) Fetch) )`
- In general, specify:
  - Rule: *FC* and/or *BC*?
  - Ground clause: whether to *FC* when Telled?
    ...or just store?
  - Query: whether to *BC* on Asked query?
    ...or just *Fetch*?

Theorem Provers

Goal: Sound, Complete, Complete Expressiveness

Proving theorems from *fixed* data;
no Tells, ...

Inferencing: General, Difficult to Control

- Many like Prolog but...
  - Add "*OccursCheck*" (to be sound)
  - Include "rewrites": "x + 0 $\mapsto$ x"
  - Allow real negation $\neg P$
  - Iterative Deepening (not DFS)
    Linear Resolution + Locking
    ...

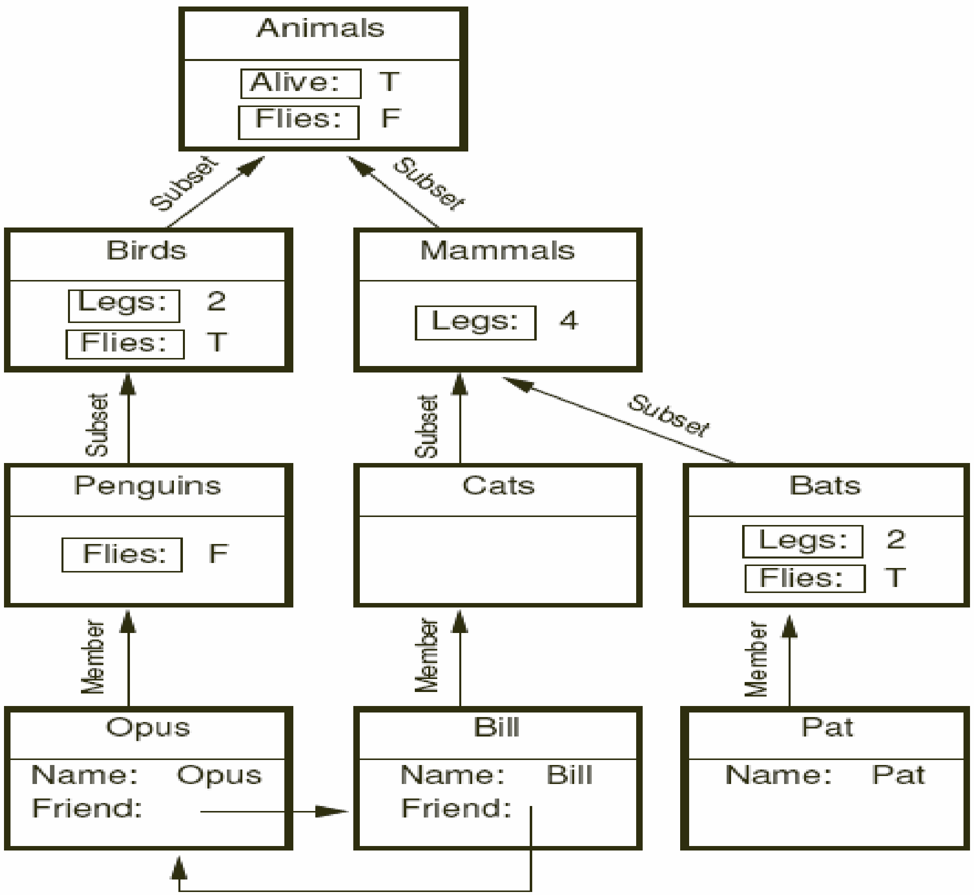
Why? Often used as *PROOF-Checkers*

For verifying ...
+ hardware circuits (16-bit adder, CPU [timing], ...)
+ software (RSA, B-M string matching, ...)

For synthesis
– automatic programming ...very hard!

(Domains with complete correct axiomitization)

Frame-based Systems



(a) A frame-based knowledge base

Rel(Alive,Animals,T)
Rel(Flies,Animals,F)

Birds $\subset$ Animals
Mammals $\subset$ Animals

Rel(Flies,Birds,T)
Rel(Legs,Birds,2)
Rel(Legs,Mammals,4)

Penguins $\subset$ Birds
Cats $\subset$ Mammals
Bats $\subset$ Mammals
Rel(Flies,Penguins,F)
Rel(Legs,Bats,2)
Rel(Flies,Bats,T)

Opus $\in$ Penguins
Bill $\in$ Cats
Pat $\in$ Bats
Name(Opus,"Opus")
Name(Bill,"Bill")
Friend(Opus,Bill)
Friend(Bill,Opus)
Name(Pat,"Pat")

(b) Translation into first-order logic

Notation for Frame Systems

- Each Cluster is “Unit”

  (aka “Frame”, “Script”, “Schema”, . . .)
- Each label is “Slot"

  (aka “Aspect”, “Attribute”, “Function”, . . .)
- Value of each unit's slot is “Value"

Eg: Birds is Unit

  Legs is Slot

  2 is Value of Unit:Slot, “Birds:Legs”

- Types of Units:
  - Penguins is “Class” type of Unit
  - Opus is “Instance” type of Unit

Use of Frame Systems

1. Begin with skeleton

   including "general frame knowledge"

2. Add information by { adding new unit / adding new slot / filling in value of slot }

   [+ sometimes by deleting a unit or a slot

   or by changing a value]

   System may "forward chain" to

   add other values. . .

   (Usually via "When-Added" procedure)

3. Pose questions by

   asking for value of slot of unit.

   Finding answer may involve

   simply RETRIEVING value, or

   COMPUTING by "backward chaining"

   "Inheritance" or

   COMPUTING via "If-Needed" procedure.

Use of Inheritance

- To answer questions like:
  What is name of Pat?
  Just look up value on Name slot of Pat.

Note: Opus unit does NOT have explicit slot
Flies, Legs

- To answer questions like:
  Does Opus fly?
  How many legs does Opus have?

  use INHERITANCE

Q: How many legs does Opus have?

A: 2

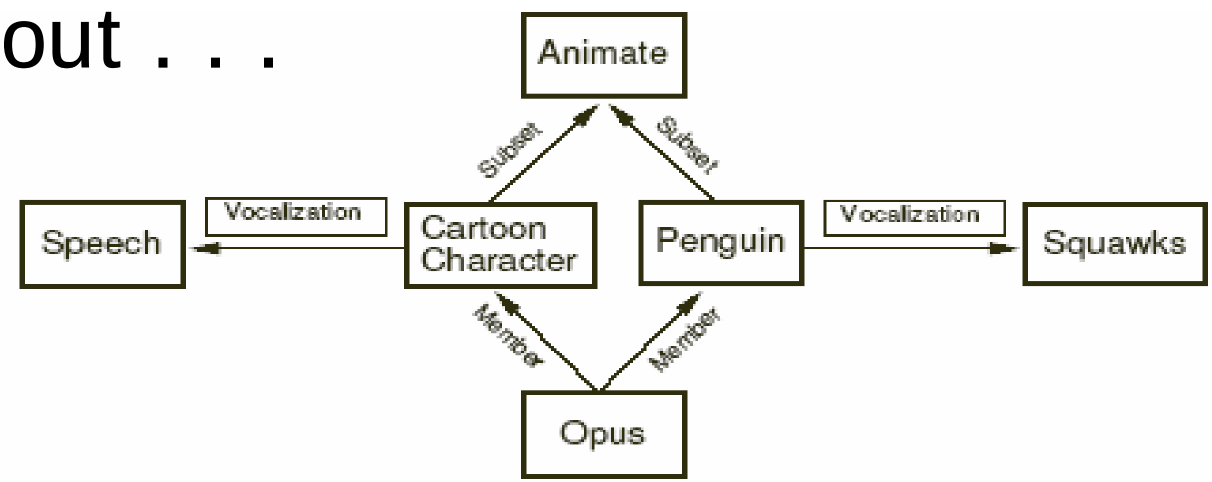
...as Opus is member of Penguins,
which are subset of Birds,
which (typically) have 2 legs.

Multiple Inheritance

- Issue: Does Opus fly?
  - F as Opus $\in$ Penguins
  - T as Opus $\in$ Penguins $\subset$ Birds
- Which to use?

  One found *first* (most specific): F

- What about . . .



Wording Frame System within Predicate Calculus

- (Most) Information in ClassUnits translates to rules.

(Eg, "Birds:Covering = Feathers" $\leadsto$

$\forall x.$ (Bird x) $\Rightarrow$ (Covering x Features))

- (Most) Information in InstanceUnits translates to ground atomic facts.

(Eg, "Tweety:Age = 3" $\leadsto$ (Age Tweety 3)
"Tweety:Flies = Yes" $\leadsto$ (Flies Tweety))

- Overall information is CONJUNCTION of these "facts".

Some exceptions:

- While Birds is ClassUnit wrt Tweety it is an InstanceUnit wrt Species.
- Meta-Information
  last access time? who created me?

Expressiveness of Frame Systems

- Easy to express: **Conjunctions of**
  - Rules of form

    $\forall x.\ (\underline{\texttt{Class}}\ x) \Rightarrow (\underline{\texttt{Slot}}\ x\ \underline{\texttt{Value}})$

  - Unary, Binary relations

    (Eg. (Flies Tweety)
    (Child Tweety B7)
    (Age Tweety 3))

- Difficult to express
  - Other atomic relations

    (Eg "(Between Rock Tweety HardPlace)")

  - Partial Knowledge

    $\exists$, $\neg$, $\vee$

Semantics of Frames

Link Type	Semantics	Example
$A \xrightarrow{Subset} B$	$A \subset B$	$Cats \subset Mammals$
$A \xrightarrow{Member} B$	$A \in B$	$Bill \in Cats$
$A \xrightarrow{R} B$	$R(A, B)$	$Bill \xrightarrow{Age} 12$
$A \xrightarrow{\boxed{R}} B$	$\forall x \;\; x \in A \;\Rightarrow\; R(x, B)$	$Birds \xrightarrow{\boxed{Legs}} 2$
$A \xrightarrow{\boxed{\boxed{R}}} B$	$\forall x \;\; \exists y \;\; x \in A \;\Rightarrow\; y \in B \wedge R(x, y)$	$Birds \xrightarrow{\boxed{\boxed{Parent}}} Birds$

Note: All Birds fly, but
Penguins (which are birds), do not.

- Semantics: $\boxed{R}$-link from A to B:
  Every member of A must have
  an R-relation to B
  unless $\exists$ intervening A' where $Rel(R, A', B')$

  "default value"
  nonomonotic inference

- Issues:
  + Can "fake" within Predicate Calculus. . .
  – Multiple inheritance

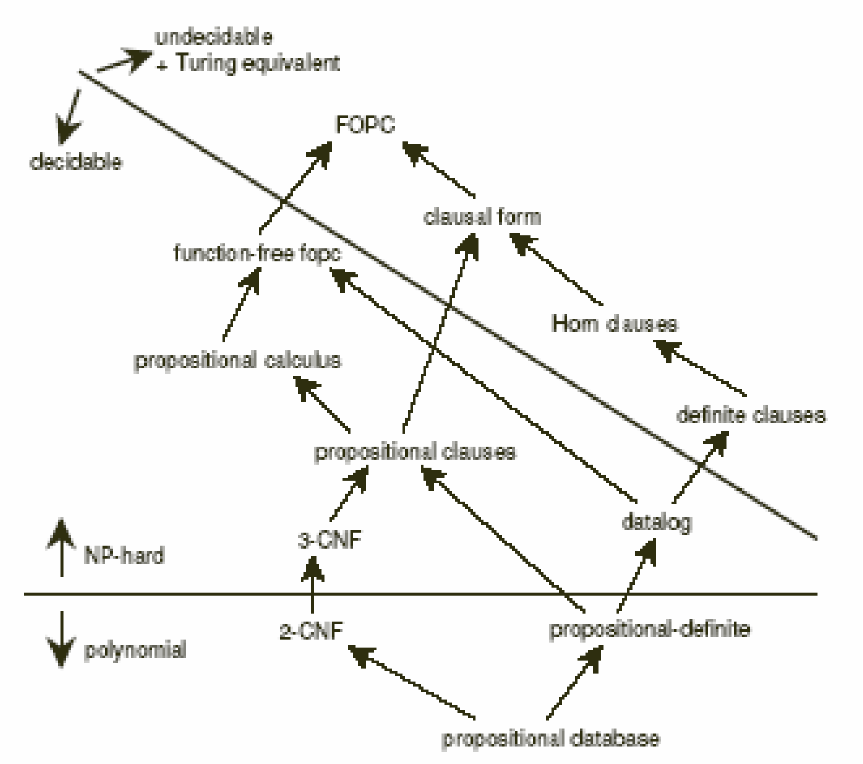
Objectives of Frame Systems

- Core ideas:
  - "Bundle" information together
    (Store everything about Birds in one place)
  - Exploit "hierarchy" to obtain "default" answers
- Cognitive Model
    (ie, people store information in similiar form)
  [. . . and so is appropriate language for
  communicating with people (designers/users). . .]
- Efficiency
  - Retrieval ["swap in" everything at once]
  - Inferencing [due to limited expressability]

≈ same as Semantic Nets...

“Complexity Cliffs”

- Complexity Cliffs:



- Be as expressive as possible within “tractable" side
- Frames/SemanticNets tractable, but not very expressive
- full Predicate Calculus is very expressive but not tractable

Description Logics

- Undecidable if $\forall$, $\exists$, $\neg, \vee, \wedge$
- Just use "tractable" subset

  eg, avoid $\neg$, only some types of $\vee$, . . .
- Define concepts

HappyMother $\equiv$
Woman whose children are all RICH, and all married to pediatricians.

SuccessfulParent $\equiv$ PERSON whose DAUGHTERS are married to doctors.

"Subsumption" questions like:
Is every HappyMother a SuccessfulParent?

What if. . .
at least 3 daughters?
either MDs or Profs?

- Uses: CLASSIC (AT&T)
  Financial Management, Database Interfaces, Software Information Systems

Explanation

- Necessary information:

  Clauses (Rules, Facts, Constraints) used in derivation
- They are needed for Derivation

  (Often required for conclusion)
- Store this info,

  associated with conclusion
- ?? Convincing story
  - Just High Points ... not all details
  - Why not another answer?

Retraction

- Tell *adds* new fact φ.

  What if φ turns out to be wrong?
  ... or if want to reclaim space?
  ... or if world has evolved, making φ irrelevant?

$\Rightarrow$ Need to Retract statement

- Retract(φ) $\not\equiv$ Tell($\neg\varphi$):

  After Tell(φ), Tell($\neg\varphi$)
  KB is INCONSISTENT

  After Retract(φ)
  KB is CONSISTENT, and Ask(φ) $=$??

Effects of Retraction

- After Tell(P) and Tell($P \Rightarrow Q$),
  agent may (forward-chain to) assert Q

  Glitter and Glitter $\Rightarrow$ Gold
  assert Gold

  If then Retract(P),
  should also Retract(Q).

- Maybe. . .

  Spse also
  Tell(R) and Tell($R \Rightarrow Q$)?

  Now INDEPENDENT reason to believe Q
  $\Rightarrow$ should NOT retract Q

$\Rightarrow$ Need to maintain REASONS for believing Q

Truth Maintenance

- Identify with each conclusion Q
  the "reason" for believing Q

  `Explain(`Q`,` $\{ P,\ P \Rightarrow Q \}$`)`

- In general, explanation is SET of SETS

  `Explain(`Q`,` $\left\{ \begin{array}{l} \{ P,\ P \Rightarrow Q \} \\ \{ R,\ S,\ R \wedge S \Rightarrow Q \} \end{array} \right\}$`)`

  Each element could be

  + Telled fact/clause/rule/...
  + assumption
  + fact reached by forward chaining

- Note after `Tell(`W`)`, have `Explain(`W`,` $\{ W \}$`)`

- Each set is "explanation"

  Retracting any member of an explanation
  removes that explanation

  If remove ALL of φ's explanation
  `Retract(`φ`)`

Comments on Explanation

- "Explain(φ, ...)" explains why agent believes φ
- In general, "explanation" can include
  - + facts in KB
  - + statements NOT in KB (defaults)

Eg: As x is bird, then x flies, unless abnormal$_{fly}$.
Given Tweety is bird, conclude Tweety flies.

$$\texttt{Explain(Fly(Tweety),} \left\{ \begin{array}{l} \texttt{Bird(Tweety)} \\ \texttt{Bird(x)} \wedge \neg \texttt{Ab}_{fly}\texttt{(x)} \Rightarrow \texttt{Fly(x)} \\ \text{“}\neg\texttt{Ab}_{fly}\texttt{(Tweety)”} \end{array} \right\} \texttt{)}$$

Then learn Tweety is penguin (Ab$_{fly}$)
$\Rightarrow$ retract Fly(Tweety)...

- Maintain SETS of consistent worlds

Then specify specific world using "in/out"

$\Rightarrow$ Assumption-based Truth Maintenance Systems

Summary

- Reasoning cannot be
  - Sound, Complete & Efficient
  - in complex domains
- Different tradeoffs:
  - Limited Expressibility:
    Database, Prolog, Description Logics
  - Incomplete, Unsound
- Which is best?
  - Depends on application, and goals
  - Good to EXPLICIT: why gave up what?