ES are Production Systems with frills

- production system:

  DB (STM) examined by rules in LTM:

  recognize: pattern matching, conflict resolution

  act: change DB

  DB: C1, C2, C3. **LTM**: C1 --> A1, C1 & C3 --> A2, C4 & C2 --> A3, C1 & C2 & C3 --> A4, ….

  matching leads to **conflict set**:

  C1 --> A1, C1 & C3 --> A2, C1 & C2 & C3 --> A4

  conflict resolution: choose C1 --> A1, execute A1 ...

production systems

- DB - a set of symbols - is the <u>only</u> store for all state vbls
  - no program counters
  - no separate control storage: DB accessible to every rule
- interpreter: recognize - act cycle
  - alternation of selection and execution
    - every rule chosen on basis of total DB contents
    - any rule can fire at any time
    - complete reevaluation of control state at each cycle
    - sensitivity, at each cycle, to <u>any</u> change in environment
- programming by pattern-directed invocation

conflict resolution strategies

- fire first matching rule
- rule fires if it matches high priority data elements
- fire most specific rule
- fire most general rule
- fire most recently used rule
- fire rule containing most recently matched elements
- execute all matching rules

- conflict resolution algorithm is expressed by meta-rules in KB

example production system

- production systems proposed by Newell & Simon as control structures for cognitive processing (eip's)

Human Information Processing

LTM:	pd1: (A & B --> (old*))	
	pd2: (C & B --> (say hi))	
	pd3: (D & (E) --> B)	
	pd4: (A --> C D)	
STM:	Q (E F) R S T	now input A

STM0:	(A Q (E F) R S)	T forgotten
STM1:[4]	(D C A Q (E F))	
STM2:[3]	(B D (E F) C A)	rehearsal of matched el, in order of cond
STM3:[1]	((old A)B D (E F) C)	flag 1.el of rehearsed STM
STM4:[2]	(C B (old A) D (E F))	say hi (forever)

to say hi once: pd2': (C & B --> (say hi) (old*))

STM4: ((old C) B (old A) D (E F)) etc.

example production system, cont.

p1: (ready) --> attend; deposit (count 0)

p2: (count x_1)(m $\neq x_1$)(n x_2) --> deposit (count (succ x_1)); deposit (n (succ x_2))

p3: (count x_1)(m x_1)(n x_2) --> (say x_2)

p4: (n x_1) --> deposit (n_0 x_1)

p5: (count x_1)(n x_2)(n_0 x_3) --> x_4 := gensym; put(x_4, ((m x_1)(n x_3)), (say x_2), p1.x_4)

[where put(a,b,c,d) = a: b --> c; d in control language CL]

input: (m 4)(n 2) output = ???

CL: p1p2*p3:

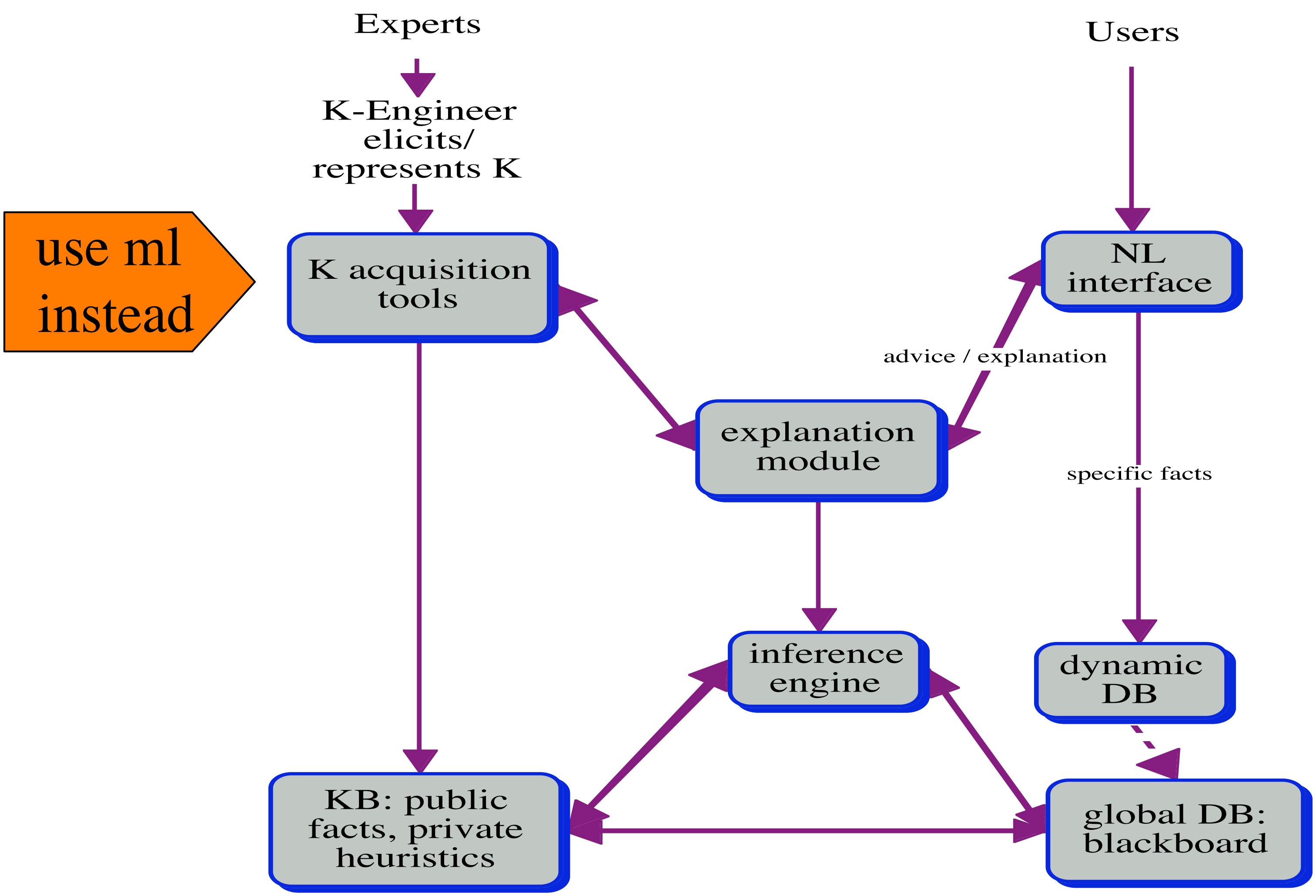
c0 m4 n2; c1 m4 n2; c2 m4 n4; c3 m4 n5; c4 m4 n6 ==> say 6 i.e. **addition**

CL: p1p4p2*p3p5:

c0 m4 n2; $n_0$2 c0 m4 n2; c1 n3 m4 $n_0$2; c2 n4 m4 $n_0$2; c3 n5 m4 $n_0$2; c4 n6 m4 $n_0$2; say 6; p.x_4: (m 4)(N 2) --> (say 6) i.e. remember result!

CL: p1p4(p2p5)*p3p5 remember everything!

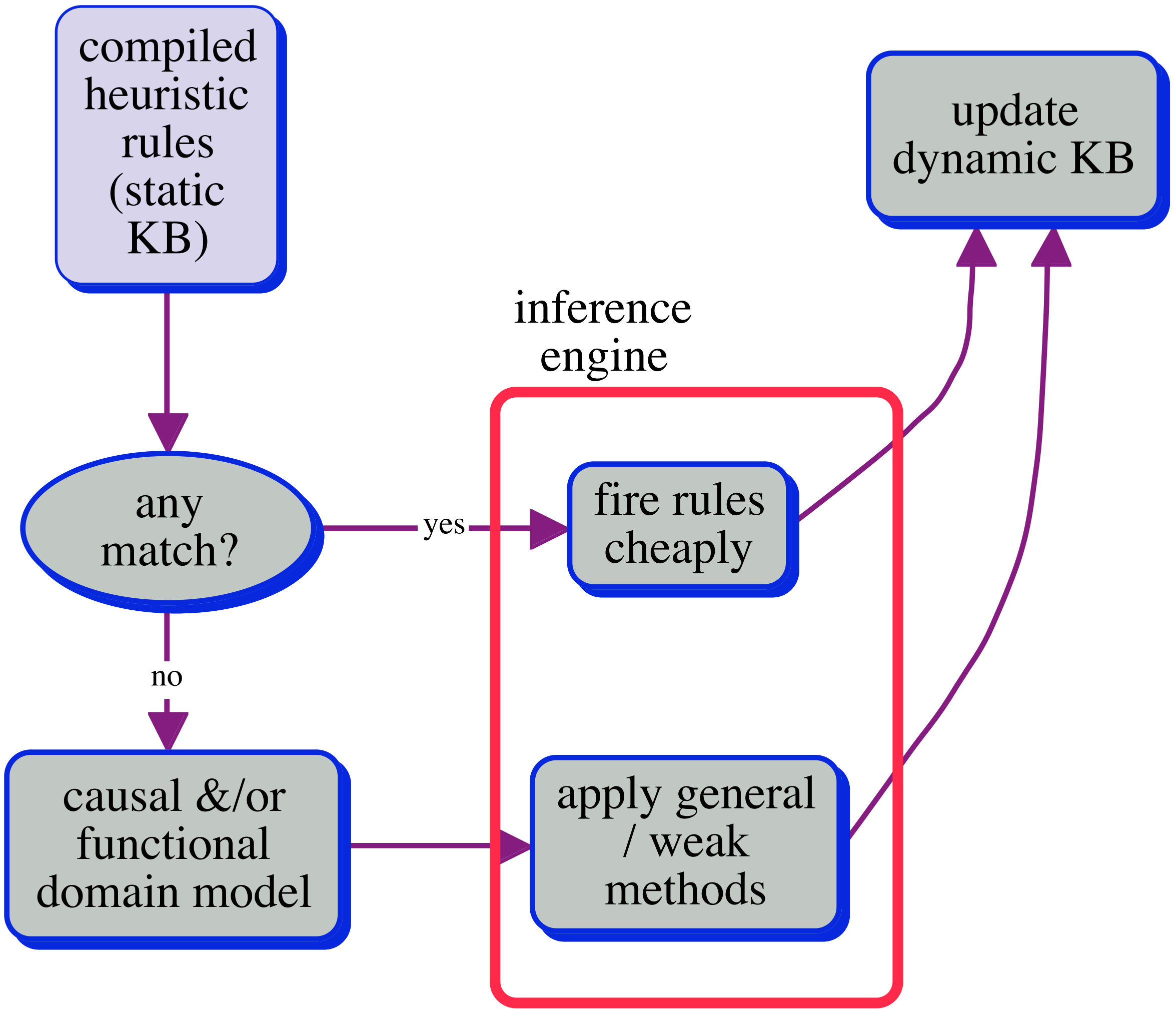
ES: trend: drop KEng; drop Expert!



knowledge-based systems

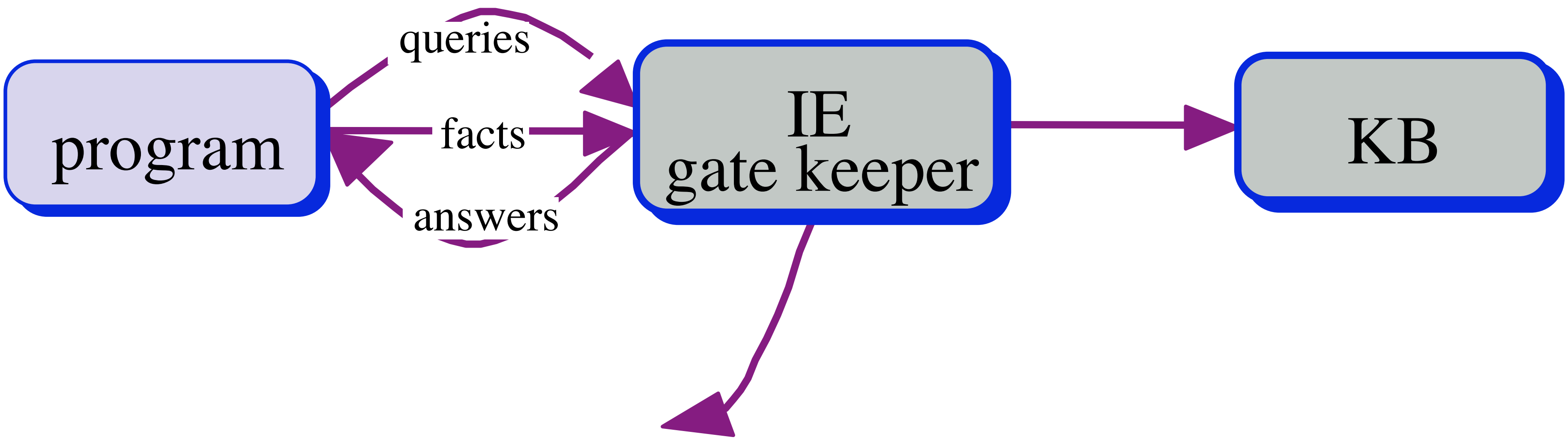
- separate task k and data from control k
  - ---> flexibility, modularity
- invoke rules/facts associatively
  - event-driven; call by content; Pattern Matching
  - heterarchical control structure
- contain lots of domain-specific, **judgemental** rules
  - invoked by Pattern Matching via task features in global DB
- work even if some rules are missing
- work even if rules are rearranged
- can easily explain their behavior
  - I used $rule_i$ in order to prove G; I wanted to prove G in order to;;;

knowledge-based systems, cont.



hard problems:
uncertain rules;
partial matching;
handling inconsistency;
user-tailored explanations;
reason maintenance;
learning rules;
handling unexpected situations;
etc...

simple rule interpreter / inference engine



- MP + PM "pattern-directed inference system"

- limited resolution
- full FOL
- certainty factors ...
- non-monotonicity
- abduction etc.

simple rule interpreter / inference engine

gate keeper: assert retract query

inference at assertion time: forward chaining

inference at query time: backward chaining

forward: when backward is inefficient;
Dog joe triggers Mammal joe immediately;
when inferred fmlas are likely to be queried;
when chaining terminates quickly:
taxonomies, definitions.

rule interpreter, cont.

- modus ponens

  p

  (if p q)

  ∴ q

  (inst chair-7 chair)

  (forall (x)(if (inst x chair)(inst x furniture))) **∴ ?**

  to get: (inst chair-7 furniture) we need a UI rule:

  (forall (- vars -) p) ∴ p'

  • drop universal quantifiers (and syntactically indicate variables)

  (if (inst ?x chair)(inst ?x furniture))

  • replace existentially quantified variables with new symbols (Skolemization)

Modus Ponens'

p'
(if p q)
∴ q'

where p' unifies with p and $q\sigma = q'$

e.g. (inst ?x chair)
unifies with
(inst chair-7 chair);
$\sigma = \{ ?x \leftarrow \text{chair-7}\}$; now we get (inst chair-7 furniture);

backward chaining

- goal driven, 'wishful thinking'
  - theorems are **found** by backward chaining but **presented** via forward chaining

prove **h**, given **b, c**. rules:

1) b & d & e --> f; 2) d & g --> a; 3) c & f --> a; 4) c --> d; 5) d --> e; 6) a --> h

6) prove a; 2) prove d, g; 4) proves d; g impossible, so backup!

3) prove c, f; c given; 1) prove b, d, e; b given; 4) proves d; 5) proves e;

now we have f; 3) proves a, 6) proves h.

c --> d

d --> e

b & d & e --> f

c & f --> a

a --> h

backward chaining, cont.

- inference at query time, generates sub … sub-queries & substitutions

  e. g. (show (inst chair-7 furniture)) unifies with
  (if (inst ?x chair)(inst ?x furniture)) & generates subgoal:
  (show (inst chair-7 chair))

(show: q')

(if p q)   in KB

q' and q have MGU σ

(show: pσ)   if this yields answer σ', then answer = $\sigma \cup \sigma'$

backward chaining, cont.

- backward chaining always returns substitutions
- MP' = combine MP and UI via Unification (log incomplete)
- reversed skolemization
  - "?"-var is existentially interpreted
  - (show (inst ?x chair)) means show that there exists a chair
  - (show (forall(y)(inst ?y chair))) means (show (inst sk_9 chair)), where sk_9 is a Skolem constant: if an arbitrary thing has predicate F, then everything has...
- conjunctive goals (show r') & (if (and p q) r) generate (show (and p' q'))
  - find answers, i.e. vbl bindings for one goal & discard those that don't work on the other goal; use heuristics…
  - selective backtracking: which vbl bindings caused goal failure?

resolution

- just one inference rule!

e.g. $\neg A$

$A \leftarrow B$

resolvent: $\neg B$

e.g. $\neg(A_1, \ldots, A_n)$

$A_k \leftarrow B_1, \ldots, B_m \quad (1 \le k \le n)$

resolvent: $\neg(A_1, \ldots, A_{k-1}, B_1, \ldots, B_m, A_{k+1}, \ldots, A_n)$

¬(dark & winter & cold)
winter if january
resolvent: ¬(dark & january & cold)

e.g. 1) receives(you, power) ← gives(logic, power, you)

2) gives(logic, power, you)

? receives(you, power)

3) **query: ¬ receives(you, power)**

1), 3) imply 4) ¬ gives(logic, power, you)

2), 4) imply □; answer: yes

resolution, cont.

convert fmlas to **clausal form (CNF)**

$\alpha = \alpha_1 \wedge \alpha_2 \ldots \wedge \alpha_n$, where $\alpha_i = \beta_{i,1} \vee \ldots \vee \beta_{i,k}$ and each $\beta_{i,j}$ is a **literal.**

Conversion (in propositional logic) uses

$(\alpha \rightarrow \beta) \leftrightarrow (\neg\alpha \vee \beta)$

$(\alpha \leftrightarrow \beta) \leftrightarrow (\alpha \wedge \beta) \vee (\neg\alpha \wedge \neg\beta)$

$\neg(\alpha \vee \beta) \leftrightarrow (\neg\alpha \wedge \neg\beta)$

$\neg(\alpha \wedge \beta) \leftrightarrow (\neg\alpha \vee \neg\beta)$

$\neg\neg\alpha \leftrightarrow \alpha$

$((\alpha \wedge \beta) \vee \gamma) \leftrightarrow ((\alpha \vee \gamma) \wedge (\beta \vee \gamma))$

e.g. convert $((p \rightarrow q) \rightarrow (p \wedge q))$:

$\neg(p \rightarrow q) \vee (p \wedge q)$, $\neg(\neg p \vee q) \vee (p \wedge q)$,

$(p \wedge \neg q) \vee (p \wedge q)$, $(p \vee p) \wedge (p \vee q) \wedge (\neg q \vee p) \wedge (\neg q \vee q)$,

$(p \vee q) \wedge (\neg q \vee p)$, i.e. $\{p,q\}, \{\neg q, p\}$.

resolution, cont.

- If clause C_1 contains p and C_2 contains ¬p, then the **resolvent of C_1 and C_2 on p** is a clause containing all <u>other</u> elements of C_1 and C_2.
- Principle of (propositional) resolution:

∴ $((p \vee \alpha) \wedge (\neg p \vee \beta)) \rightarrow (\alpha \vee \beta)$

$\neg p \rightarrow \alpha$, $p \rightarrow \beta$, and since $p \vee \neg p$:
$\alpha \vee \beta$, i.e $\neg \alpha \rightarrow \beta$.
…. transitivity of implication

- algorithm: to derive α from S:

convert S and $\neg\alpha$ to clausal form (and aim for a contradiction)
repeat
    <u>pick</u> p and C_1, C_2 that can be resolved on p
    simplify resolvent by eliminating duplicates
    remove resolvent if it has both a q and ¬q
    add resolvent to original set if not there
if the empty clause results, S implies α.

conversion to clausal form

- e.g. $\{c \rightarrow s, \neg g \rightarrow d, \neg g \vee c\}$ implies $\neg s \rightarrow d$???

$\neg c \vee s$, $g \vee d$, $\neg g \vee c$, neg of conclusion: $\neg s \wedge \neg d$

$C_1 = \{\neg c, s\}$, $C_2 = \{g,d\}$, $C_3 = \{\neg g, c\}$, $C_4 = \{\neg s\}$, $C_5 = \{\neg d\}$,

$(((C_2 * C_5) * C_3) * C_1) * C_4) = \Box$ = empty clause (successful derivation)

- e.g. $\{\neg p \rightarrow r, p \rightarrow s, r \rightarrow q, s \rightarrow \neg t, t\}$ implies q??

$C_1=\{p,r\}$, $C_2=\{\neg p,s\}$, $C_3=\{\neg r,q\}$, $C_4=\{\neg s,\neg t\}$, $C_5=\{t\}$, $C_6=\{\neg q)$.

… $(((C_3 * C_6) * C_1) * C_2) * C_4) * C_5 = \Box$

- e.g. prove $p \vee \neg p$: negate: $\neg p \wedge p$; $\{\neg p\}$, $\{p\}$ resolves to $\Box$.

Conversion to clausal form is a bit trickier in fol:

$$\neg \forall \upsilon\, \Phi \Leftrightarrow \exists \upsilon\, \neg \Phi, \quad \neg \exists \upsilon\, \Phi \Leftrightarrow \forall \upsilon\, \neg \Phi$$

standardize variables apart: $\forall x Fx \wedge \exists x Gx$ becomes $\forall x Fx \wedge \exists y Gy$

converting fol to clausal form

- $\neg\forall\upsilon\ \Phi \Leftrightarrow \exists\upsilon\ \neg\Phi$, $\neg\exists\upsilon\ \Phi \Leftrightarrow \forall\upsilon\ \neg\Phi$
- remove ∃:

  if ∃ is not in scope of universal quantifier, drop it and replace quantified variable by **new** constant,

  else … by a term formed from **new function symbol** applied to variables associated with enclosing univ. quantifiers (Skolem function)

  ∀x∀y∃z F(xyz) becomes ∀x∀y F(xy**Sk(xy)**)
- remove ∀
- put into CNF, drop operators, standardize apart again; add **unification:**

  1. {Px, Qxy}
  2. {¬Pa, Rbz}
  3. {Qay, Rbz} 1,2

a very modest example

- assume: Art is father of Joe, Bob is father of Kim, fathers are parents. **Is Art parent of Joe?**

1. {Faj} 2. {Fbk} 3. {¬Fxy, Pxy} 4. **{¬Paj}**		neg of conclusion
5. {Paj}	1,3	
6. {Pbk}	2,3	
7. {¬Faj}	3,4	
8. {}	1,7	
9. {}	4,5	

- **who is Joe's parent?** 1..3 as above; 4. **{¬Pzj, Az}**

5. {Paj}	1,3	
6. {Pbk}	2,3	
7. {¬Fwj,Aw}	3,4	
8. {Aa}	4,5	
9. {Aa}	1,7	A(i.e. answer)=Art

unification --- general pattern matching

- unification is needed to make pred. logical expressions **identical** for resolution
- a substitution σ is a set of assignments of terms to vbles, no vbl being assigned more than 1 term
  - E σ (subst. inst. of E): replace vbls in E by terms assigned by σ.
    vbls in E, not mentioned in σ: unchanged.
    assignments in σ to vbls not in E: ignored.
- $E_1 \sigma = E_2 \sigma$: common instance of E_1, E_2;
  σ is a unifier for E_1 and E_2
- $E_1, \ldots, E_n$ are unifiable if there is a σ making them identical:
  $E_1 \sigma = E_2 \sigma = \ldots.$

unification, cont.

- for all E, $E(\sigma \circ \lambda) = (E\sigma) \circ \lambda$
- σ is a **most general unifier** of set X if every unifier λ of X satisfies $\lambda = \sigma \circ \lambda$
- if X is unifiable then there exists a MGU

p(x), p(5); $\sigma = \{x \leftarrow 5\}$; ci: p(x) σ = p(5) σ = p(5).

p(x,x), p(5,y); $\sigma = \{x \leftarrow 5, y \leftarrow 5\}$; ci: p(5,5).

p(1,3), p(x,g(5,y)); no match

p(x,g(joe,y)), p(h(3),g(z,mary))

σ = {x ← h(3), y ← mary, z ← joe }; ci: p(h(3),g(joe,mary)).

p(y,g(jack,y)), p(mary,g(w,z))

σ = {y ← mary, w ← jack, z ← mary }; ci: p(mary, g(jack,mary)).

unification, cont.

- each vbl is associated with at most one expression
- no vbl with an associated expression occurs in any of the assoc exprs
  - e.g. {x/g(y), y/f(x)} not a substitution
- composing σ_1 with σ_2: apply σ_1 to terms of σ_2 and add to σ_2 the bindings from σ_1
  - {w/g(x,y)} ° {x/A, y/B, z/C} = {w/**g(A,B)**, x/A, y/B, z/C}
  - {x/A, y/B, z/C} unifies p(A,y,z) and p(x,B,z) but - instead of z/Z - we could have z/D, z/f(w) or nothing ….

  MGU = {x/A, y/B}

unification algorithm

```
MGU (x, y) ←
    x = y → return {};
    Vbl(x) → return (MGUvar (x,y));  Vbl(y) → return (MGUvar (y,x));
    Constant(x) or Constant(y) → return false; ¬(length(x) = length(y)) → return false;
    i ← 0; g ← { };
    while i < length(x) do
        s ← MGU (Part(x, i), Part(y, i));  // toplevel fun or pred constant is Part 0
        s false → return false;
        g ← Compose(g, s); x ← Subst(x, g); y ← Subst(y, g); // returns expr σ
        i++;
    return g; // i.e. MGU
```

length=# of args; Subst(expr, substitution) returns resulting expr after applying substitution; Part(expr,i) is ith part of expr.

unification algorithm, cont.

MGUvar (x, y) ← // don't unify p(x) with p(f(x)) ….
Includes (x, y) → return false; // fail
return ({x/y}).

Includes (var, expr) checks whether var occurs in term with which it is being unified …. **occurs check**

occurs check prevents circularities:

given (not (sees ?z ?z)), (if (not(sees ?x (feet ?x))) (shouldDiet ?x)),

1. call of MGUvar(?z ?x) returns {?z/?x}

2.call: ?z already bound to ?x, thus

3. (recursive) call: MGUvar (?x (feet ?x));

without occurs check, this would return {?x/(feet ?x)}. **not** a unifier! We don't want: (shouldDiet (feet ?x))!!

unification algorithm, cont.

given:

(on ?x table)

(if (on something ?x)(collapses ?x))

does this imply:

(collapses table) ???

(on ?x table) and (on something ?x) do **not** unify because of **coincidental** occurrence of ?x !

* don't forget to rename vars so that no var occurs in more than one clause: **standardizing apart**

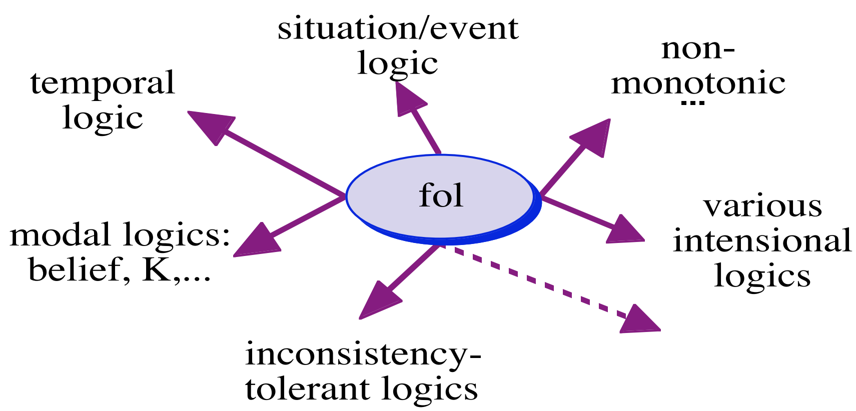
(on ?x table) and (on something ?y) unify: {?x/something, ?y/table}

logic in AI

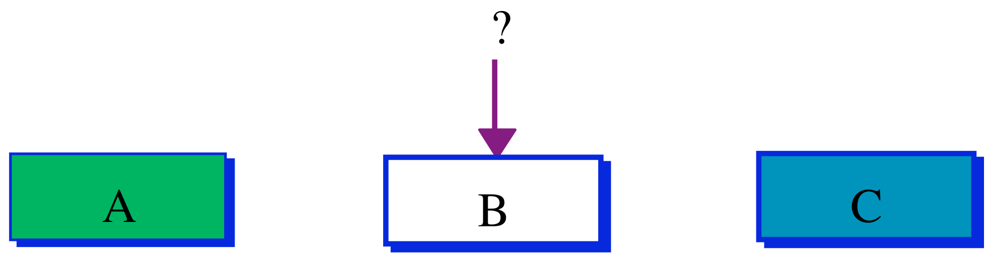
- 3 roles of logic
  - infer from KB, deductive problem solving
  - KR scheme

    "no notation without denotation", i.e. no KR scheme without proof theory and semantics
  - prove results about AI theories

    esp. in areas where theories are not (otherwise) mathematically analyzable



logic can handle many kinds of incomplete K!!



is there a green block next to a block that is not green?

- ability to say $\exists x Fx$ without knowing which object makes it T.
- ability to say $\forall x(Fx \rightarrow Gx)$ without enumerating all F-things.
- ability to say $p \vee q$ without knowing which is T
  (reasoning by cases)
- ability to handle negation

===> fol

KE with logic

KE	vs.	programming
choose a logic		choose progr. lang.
build a KB		write a program
implement proof theory		choose compiler
infer new facts		run a program

- Why KE with logic? it's easier…
  - KE just specifies what is true and inference engine figures out how to turn facts into solution
  - KB can be reused unchanged for different tasks
  - debugging a KB is easy because each sentence is T/F by itself, regardless of context
  - agent-based SE: make systems/resources interoperable via **declarative fol interface**

KE with logic, cont.

hard part of KE: write facts at 'proper' level of generality, choose 'proper' number of primitive predicates ….

BearOfVerySmallBrain(Pooh) does not imply that Pooh is a bear, has a small or very small brain or any brain at all, etc.

$\forall b$ (BearOfVerySmallBrain(b) $\rightarrow$ Silly(b)) even worse because it's much too specific...

Instead of BearOfVerySmallBrain(Pooh):

Bear(Pooh), $\forall b$ (Bear(b) $\rightarrow$ Animal(b)), $\forall b$ (Animal (b) $\rightarrow$ PhysicalThing(b))

RelSize(BrainOf(Pooh),BrainOf(TypicalBear))=Very(Small), where Very maps points on a scale towards extremes: Medium = 1, $\forall x$ (x>Medium $\rightarrow$ Very(x)>x),

$\forall x$ (x<Medium $\rightarrow$ Very(x)<x)

$\forall a$ (Animal (a) $\rightarrow$ Brain(BrainOf(a))), $\forall a$ PartOf (BrainOf(a),a)

$\forall x\ y$ (PartOf (x,y) $\wedge$ PhysicalThing(y) $\rightarrow$ PhysicalThing(x)) (!)

$\forall x$ (RelSize(BrainOf(x), BrainOf (TypicalMember (SpeciesOf(x)))) $\leq$ Small $\rightarrow$ Silly(x))

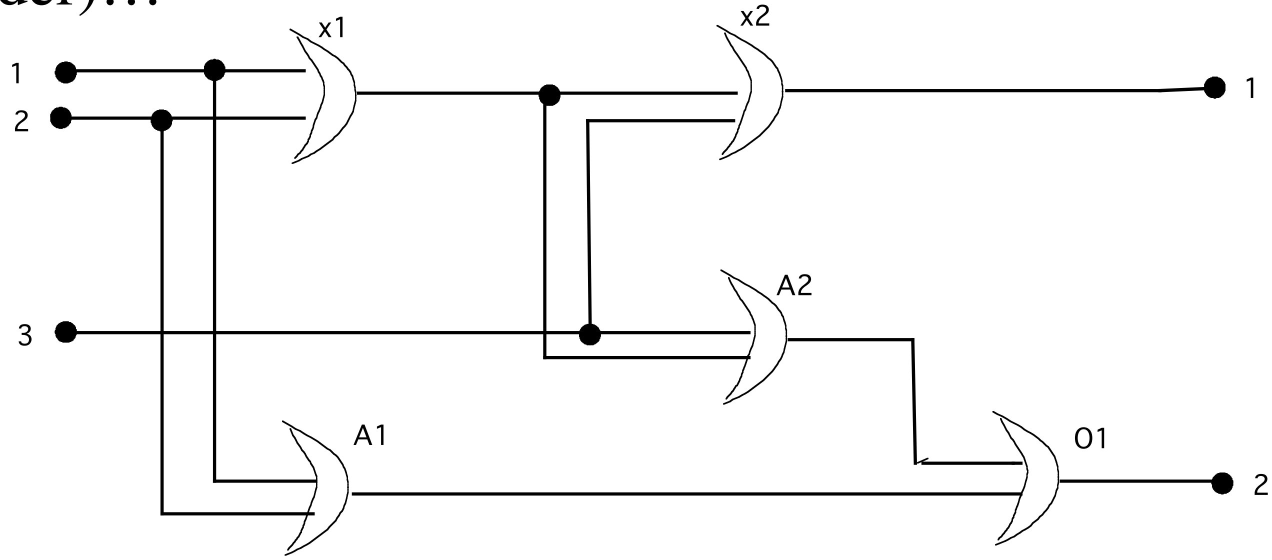
$\forall b$ (Bear(b) $\leftrightarrow$ SpeciesOf(b) = Ursidae), TypicalBear = TypicalMember(Ursidae)

$\forall x$ (PhysicalThing (x) $\rightarrow \exists s$ Size(x) = s), Tiny < Small < Medium < Large < Huge

$\forall a\ b$ RelSize(a,b) = Size(a)/Size(b)

Circuits Domain

- wanted: a system to analyze circuits like this (maybe a one bit full adder)…



3 inputs, 2 outputs, 2 XOR gates, 2 AND gates, 1 OR gate.
Inputs are bits to be added, outputs are sum bit and carry bit.

Circuits Domain, cont.

- decide what to talk about: circuits, terminals, signals at terminals, gates, gate types.
  - omit time, propagation etc.
- decide on vocabulary
- encode **general** rules: axiomatize the domain.
  - rule generality interacts with vocabulary…

- if 2 terminals are connected, they have the same signal:

$$\forall\, t_1\ t_2\ (Connected(t_1, t_2) \rightarrow Signal(t_1) = Signal(t_2))$$

- the signal at every terminal is either on or off:

$$\forall\, t\ (Signal(t) = On \lor Signal(t) = Off),\ \ On \neq Off$$

- Connected is commutative:

$$\forall\, t_1\ t_2\ (Connected(t_1, t_2) \leftrightarrow Connected(t_2, t_1))$$

Circuits Domain, cont.

- an OR gate's output is On iff any of its inputs are On:

  $\forall$ g (Type(g) = OR $\rightarrow$ (Signal(Out(1,g)=On $\leftrightarrow$ $\exists$ n Signal(In(n,g)=On)))
- an AND gate's output is Off iff any of its inputs are Off:

  $\forall$ g (Type(g) = AND $\rightarrow$ (Signal(Out(1,g)=Off $\leftrightarrow$ $\exists$ n Signal(In(n,g)=Off)))
- an XOR gate's output is On iff its inputs are different:

  $\forall$ g (Type(g) = XOR $\rightarrow$ (Signal(Out(1,g)=On $\leftrightarrow$ Signal(In(1,g)$\neq$ Signal(In(2,g))))
- a NOT gate's output $\neq$ its input:

  $\forall$ g (Type(g) = NOT $\rightarrow$ (Signal(Out(1,g)$\neq$ Signal(In(1,g)))

- this KB can easily be extended….

representing a specific circuit

Type(X_1)=XOR, Type(X_2)=XOR, Type(A_1)=AND, Type(A_2)=AND, Type(O_1)=OR,
Connected(Out(1,X_1), In(1,X_2)), Connected(Out(1,X_1), In(2,A_2)),
Connected(Out(1,A_2), In(1,O_1)), Connected(Out(1,A_1), In(2,O_1)),
Connected(Out(1,X_2), Out(1,C_1)), Connected(Out(1,O_1), Out(2,C_1)),
Connected(In(1,C_1), In(1,X_1)), Connected(In(1,C_1), In(1,A_1)),
Connected(In(2,C_1), In(2,X_1)), Connected(In(2,C_1), In(2,A_1)),
Connected(In(3,C_1), In(2,X_2)), Connected(In(3,C_1), In(1,A_2)).

….that's easy….

- now we are ready to ask questions, i.e. use inference engine:
  - the answers come for free ...

questions about the circuit

- ?what inputs cause sum bit to be off and carry bit to be on?

$\exists\ i_1\ i_2\ i_3\ (Signal(In(1,C_1))=i_1 \wedge Signal(In(2,C_1))=i_2 \wedge$
$Signal(In(3,C_1))=i_3 \wedge Signal(Out(1,C_1))=Off \wedge$
$Signal(Out(2,C_1))=On)$

answer:

$(i_1 = On \wedge i_2 = On \wedge i_3 = Off) \vee (i_1 = On \wedge i_2 = Off \wedge i_3 = On) \vee$
$(i_1 = Off \wedge i_2 = On \wedge i_3 = On)$

- ?what are possible sets of values of all the terminals for the adder circuit?

$\exists\ i_1\ i_2\ i_3\ o_1\ o_2\ (Signal(In(1,C_1))=i_1 \wedge Signal(In(2,C_1))=i_2 \wedge$
$Signal(In(3,C_1))=i_3 \wedge Signal(Out(1,C_1))=o_1 \wedge Signal(Out(2,C_1))=o_2)$

answer: complete I/O table, useful for circuit verification.

etc...