First-Order Logic

CSCE 420 – Spring 2023

read: Ch. 8,9

First-Order Logic as Knowledge Repr. for AI

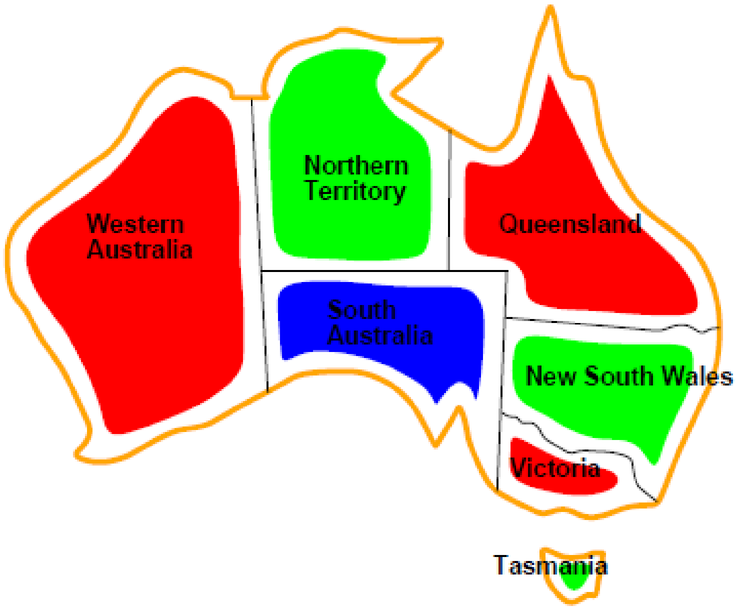
- while Prop Log and Boolean satisfiability has many applications, it has limited expressiveness
  - think of how many rules or clauses were required for the Wumpus world, or tic-tac-toe, or map-coloring
- First-Order Logic (FOL) is more expressive
  - FOL is considered the *lingua franca* for AI, or the standard concept representation language for underlying most knowledge bases
  - flexible enough to express almost any concept
  - many KR systems have been proposed over the years, but the AI community has found FOL to be the common, most useful, general language
- two influential books/papers (among many) showing the generality of FOL for KR:
  - Patrick Hayes – Naive Physics Manifesto (1978)
  - Ernest Davis – Representations of Commonsense Knowledge (1990)

Overview of FOL

- the main extensions to the language are:
  - we now have predicates, not just propositions, making it relational
    - father(Bart,Homer) instead of FatherOfBartIsHomer
  - we now have variables and quantifiers
    - $\forall$c car(c)→hasEngine(c)

Example of FOL Expressiveness

- Map-coloring
  - PropLog
    - WAR v WAG v WAB, NTR v NTG v NTB...
    - WAR→ ¬WAB^¬WAG, WAG → ¬WAB^ ¬WAR ...
    - WAG → ¬NTG, WAG → ¬SAG...
    - (about 50 sentences)
  - FOL
    - neigh(WA,NT),neigh(WA,SA),neigh(NT,SA),neigh(NT,Q)...
    - color(R),color(G),color(B)
    - state(WA),state(NT)...,state(V),state(T)
    - ∀s state(s) →∃c color(c)^hasColor(s,c)
    - ∀s,c,d state(s)^hasColor(s,c)^hasColor(s,d)→c=d
    - ∀s,t,c state(s)^state(t)^neigh(s,t)^hasColor(s,c) →¬hasColor(t,c)
    - (more concise than Prop Log - only 3 rules!)



food for thought: how would you write a KB in FOL for the wumpus world?
or for choosing optimal moves in tic-tac-toe given a current board state?

Syntax of FOL

- BNF
  - <sentence> ::= <atomic> | <complex>
  - <atomic> ::= <predicate> | <equality>
  - <predicate> ::= <predicatename>(<term>*)
    - predicate names are symbols, like propositions
    - they represent *properties* or *categories* (for unary case, 1 arg), or *relationships* (for n-ary case, $n \geq 2$)
    - examples: cat(garfield), hungry(garfield), owner(garfield,jon),feeds(jon,garfield,lasagna)
  - <term> ::= <const> | <var> | <function>
    - consts and vars both look like symbols, but the difference is usually clear from context
    - some languages mark vars, e.g '?x',
  - <function> ::= <functionname>(<arg>*)
    - functions look like predicates, but they are always *embedded inside predicates* as args
    - loves(bill,<u>motherOf</u>(bill)), in(<u>keys</u>(<u>carOf</u>(jon)),<u>pocketOf</u>(<u>pantsOf</u>(jon)))

Syntax of FOL

- BNF cont'd
  - <complex> ::= (<sent>) | <sent> <binop> <sent> | ¬<sent> |<quantified>
  - <binop> ::= ^ | v | → | ↔ | ⊕
  - <quantified> ::= <quantifier><var><sentence>
  - <quantifier> ::= ∀ | ∃
    - note: all variables in sentence should be quantified (else they are called 'free')
    - we can combine several variables for concision: ∀x∀y P(x,y) ≡ ∀x,y P(x,y)
    - scoping and order of quantifiers matters!
    - ∀x ∃y loves(x,y) // everybody loves somebody
    - ∃y ∀x loves(x,y) // there is somebody loved by everybody

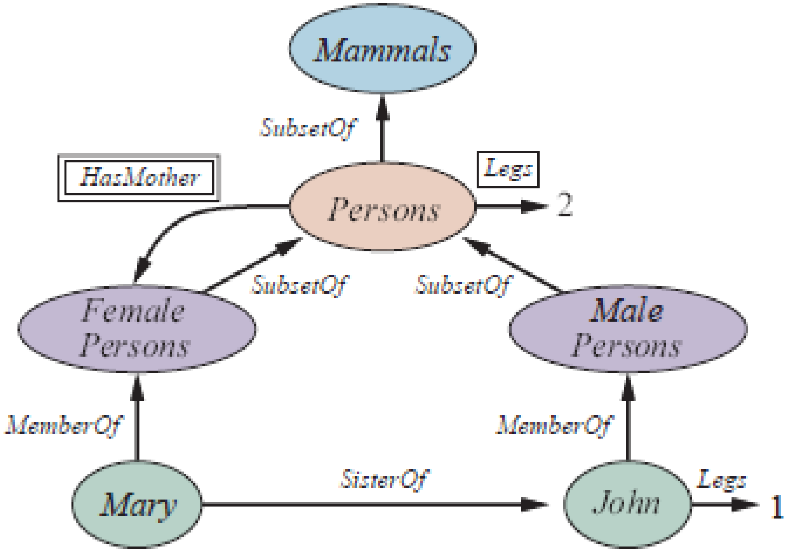
Syntax of FOL

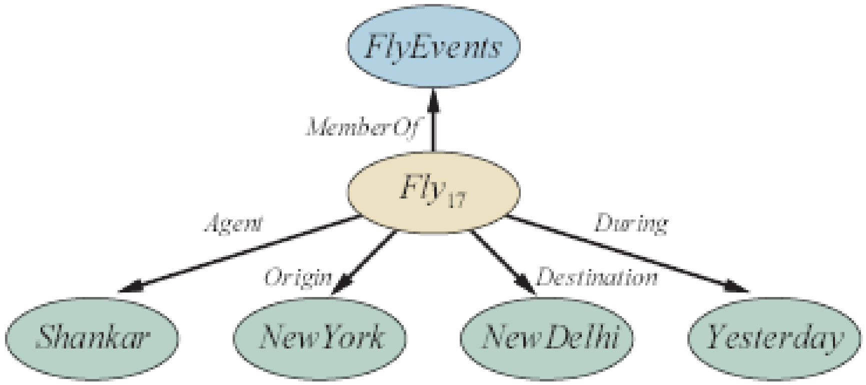
- Equality
  - <equality> ::= <term>=<term>
    - includes <var>=<const>, <var>=<var>, <const>=<const>, <const>=<funct>...
    - examples: ?c=red, ?x=?y, alice=motherOf(bill)
    - technically, '=' is just a binary predicate! like this: Eq(alice,motherOf(bill))
    - can negate these too: $\forall$s,t,c,d hasColor(s,c)^hasColor(t,d)^neigh(s,t)→$\neg$c=d ($\equiv$ c$\neq$d)
- Numbers
  - constants with conventional meanings, like 0, 1, -2, 4.501, (and π, e,...)
  - $\forall$x biped(x)→numLegs(x)=2 // Eq(numLegs(x),2), note: numLegs() is a function
  - or... $\forall$x biped(x)→$\exists$y,z leg(y)^leg(z)^partOf(y,x)^partOf(z,x)^y$\neq$z ^ ...
    ($\forall$w leg(w)^partOf(w,x)→(w=y v w=z))
  - actually, although this definition is more verbose, it is preferred because you can do more reasoning with it, because it identifies specific objects as legs; leg() and partOf() are useful as general predicates for making other inferences

Guidelines for Translating Knowledge into FOL

- divide the world into:
  - objects
    - I mean this in the abstract, conceptual way - anything we can 'talk about' or 'refer to'
    - garfield, sam's birthday, queen of England, the signing of the magna carte, ...
  - types/categories/properties of things
    - cats, game pieces, colors, states, people, apples, legs...
    - events, situations
    - model these with unary predicates, e.g. cat(garfield), F150($truck_7$), birthday(b_{152})
    - happy(x), salty(x), broken(x), hasPower(x)...
  - relations
    - prerequisite(csce411,csce420), instructor(csce221,DrWelch), birthdayPerson(b_{152},sam), owner(cheers,sam), girlfriend(sam,diane)

Using FOL





∀x person(x)→mammal(x)
∀x person(x)→∃y hasMother(x,y)^femalePerson(y)
∀x femalePerson(x)→person(x)
femalePerson(mary)
malePerson(john)
sisterOf(john,mary)

this illustrates rules encoding taxonomic info

FlyEvent(Fly17)
agent(Fly17,Shankar)
origin(Fly17,NewYork)
destination(Fly17,NewDelhi)
during(Fly17,yesterday)

this illustrates 'reification' – treating an abstract thing such as an event like an 'object' which has properties and relates to other objects

Using FOL

- writing concept definitions as rules
  - ∀x batchelor(x) ↔ person(x) ^ adult(x) ^ male(x) ^ ¬married(x)
  - ∀x,y grandmother(x,y) ↔ ∃z parent(x,z) ^ parent(z,y) ^ female(x)
  - how would you define: hard-drive? chair? ambush? bargain?
- properties are like subsets
  - ∀x plant(x)→green(x) // plants are a subset of things that are green
- describing compositions of objects: partOf predicate
  - ∀c car(c) → ∃t tire(t) ^ partOf(x,t) // don't forget to relate the 2 objects
  - ∀x biped(x) → ∃y,z leg(y) ^ leg(z) ^ partOf(y,x) ^ partOf(z,x) ^ y≠z ^ (∀w leg(w) ^ partOf(w,x) → (w=y v w=z))
  - partOf(toe,foot), partOf(foot,leg), partOf(leg,humanBody)
- location and spatial relationships:
  - loc(house(joe),BCS) // i.e. geographic location; BCS is a 'place'
  - ∀d,h frontDoor(d,h)↔door(d)^house(h)^in(d,frontSideOf(h)) // note the function
  - ∀x,y,z in(x,y)^in(y,z)→in(x,z) // transitivity, e.g. milk in fridge, in kitchen
  - ∀a,b,L in(a,L)^partOf(b,a) →in(b,L) // if in(patient59,room1002), so are his toes...

Guidelines for Translating Knowledge into FOL

- important: divide long constants and predicate names into simple concepts (and define them)
  - instead of *below30psi(leftFrontTireOfJohnsKia)*, say:
  - ∃t,c tire(t) ^ car(c) ^ partOf(t,c) ^ owner(c,john) ^ make(c,kia) ^ on(t,LeftSide(c))
    ^ on(t,frontSide(c)) ^ pressure(t)<psi(30)
  - this is a common trick - using existentially quantified variables to refer to objects, and then using lots of basic predicates to describe the properties of and relations among the objects
  - remember our example of replacing 'numlegs(x)'...
- usually, implications go with universal quantifiers
  - correct: ∀x plant(x)→green(x)
  - incorrect: ∃x plant(x)→green(x)
- usually, conjunctions go with existential quantifiers
  - (see tire example above)

Axiomatizing Numbers

- Natural numbers (0,1,2...)
- Peano axioms
  - natNum(0) // there exists a natural number, denoted by '0'
  - $\forall$ n natNum(n)→natNum(S(n)) // *successor function*
  - $\forall$ *m* and *n*, *m* = *n* ↔*S*(*m*) = *S*(*n*).
  - $\forall$ *n* (*S*(*n*) $\neq$ 0) // there is no natural number whose successor is 0.
  - $\forall$ *n* plus(n,0)=n // n+0=n
  - $\forall$ *n,m* plus(n,S(m))=S(plus(n,m)) // n+(m+1)=(n+m)+1
  - ...there are a few more
- the point is that natural numbers <u>exist</u> and we can use basic arithmetic (as functions) in FOL sentences
  - $\forall$x,n,y biped(x) ^ Eq(numLegs(x),n) ^ tripod(y) → Eq(numLegs(y),Plus(x,1)) // "x+1"
- note that functions in the arithmetic sense are represented by functions in the logical sense

Axiomatizing Numbers

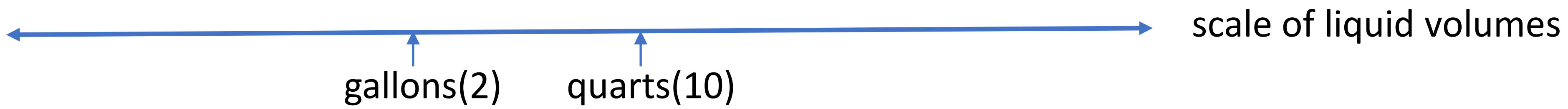
- rational numbers – easy:
  - $\forall$q rational(q)↔$\exists$a,b natNum(a)^natNum(b)^b≠0^q=frac(a,b)
- real numbers: Continuum hypothesis
  - it's trickier to axiomatize these, but we can go ahead and assume real numbers exist! so we can use them is our FOL sentences
  - furthermore, we can assume functions, like Plus(a,b), Times(x,y) exist, so we can say things like:
  - $\forall$c,t,d,m car(c)^trip(t)^
    distanceTraveled(t,d)^gasMileage(c,m)→fuelUsed(c,t)=Times(d,m) *or "=d*m"*
- axioms for transcendental numbers; transfinite numbers...(axioms for the math cognoscenti)

Sets

- remember: order doesn't matter (or repeats)
- $\forall$s set(s)$\leftrightarrow$s=$\emptyset$ v [$\exists$x,a set(x) ^ s=Add(a,x)]
- ¬$\exists$s,a Add(a,s)=$\emptyset$
- $\forall$s,a Member(a,s) $\leftrightarrow$ $\exists$t Add(a,t)=s // a$\in$s is shorthand for Member(x,s)
- $\forall$r,s Subset(r,s)$\leftrightarrow$[$\forall$x Member(x,r)$\rightarrow$Member(x,s)]
- $\forall$r,s set(r)^set(s)^r=s$\leftrightarrow$[Subset(r,s) ^ Subset(s,r)]
- $\forall$r,s,x Member(x,Union(r,s))$\leftrightarrow$[Member(x,r) v Member(x,s)] // r$\cup$s
- $\forall$r,s,x Member(x,Intersection(r,s))$\leftrightarrow$[Member(x,r) ^ Member(x,s)]

Quantities

- it is useful to be able to specify quantities, e.g. Bill bought 2 gallons of gas and 10 quarts of milk (which was more?)
- use *functions* to indicate units of quantities
  - ∃g bought(Bill,g)^gas(g)^volume(g)=gallons(2)
  - ∃m bought(Bill,m)^milk(m)^volume(m)=quarts(10)
- the functions map numbers to 'volumes' as objects on an abstract scale, where quarts(10) is more than gallons(2)
  - we want to be able to infer that volume(m)>volume(g)
- we can connect them and reason about quantities with axioms like
  - ∀x,y volume(x)=gallons(y) → volume(x)=quarts(4*y)



Semantics of FOL: Model Theory

- in Prop Log, models were truth-assignments over propositions <P=T, Q=F...>
- in FOL, a model consists of 3 things: <**U,D,R**>
  - **U** is a set of abstract objects in the universe (also called 'domain'); not necessary finite!
  - **D** are *denotations*, mappings from constants and functions to objects, *d:const|→U*
    - for functions, there can be only one denotation for each argument
    - example: loves(bill,motherOf(bill)) works because there is only 1
    - loves(sue,pet(sue)) would not work, because she could have more than 1 pet
    - in 1-to-many situations, use a predicate: ▯x pet(sue,x)→loves(sue,x)

  - **R** is a set of relations (tuples over U^n) defining each predicate
    - for a unary predicate (n=1), it is just the subset of objects U that satisfies it
    - note: we can't just say R_{dog}={snoopy,marmaduke,...} because these are constant terms
    - they need to be the objects in U that are denoted by theose terms, e.g. $R_{dog}=\{u_1,u_2...\}$ if d('snoopy')=u_1, d(marmaduke')=u_2, for $u_1,u_2 \in U$
    - for n-ary predicates, it is the set of n-tuples$\subset$UxU..xU that satisfies it
    - note: the equality binary predicate, =, is always implicitly defined in any model as $R_{Eq}=\{\langle o_1,o_1\rangle,\langle o_2,o_2\rangle...\}$ for all $o_i \in U$

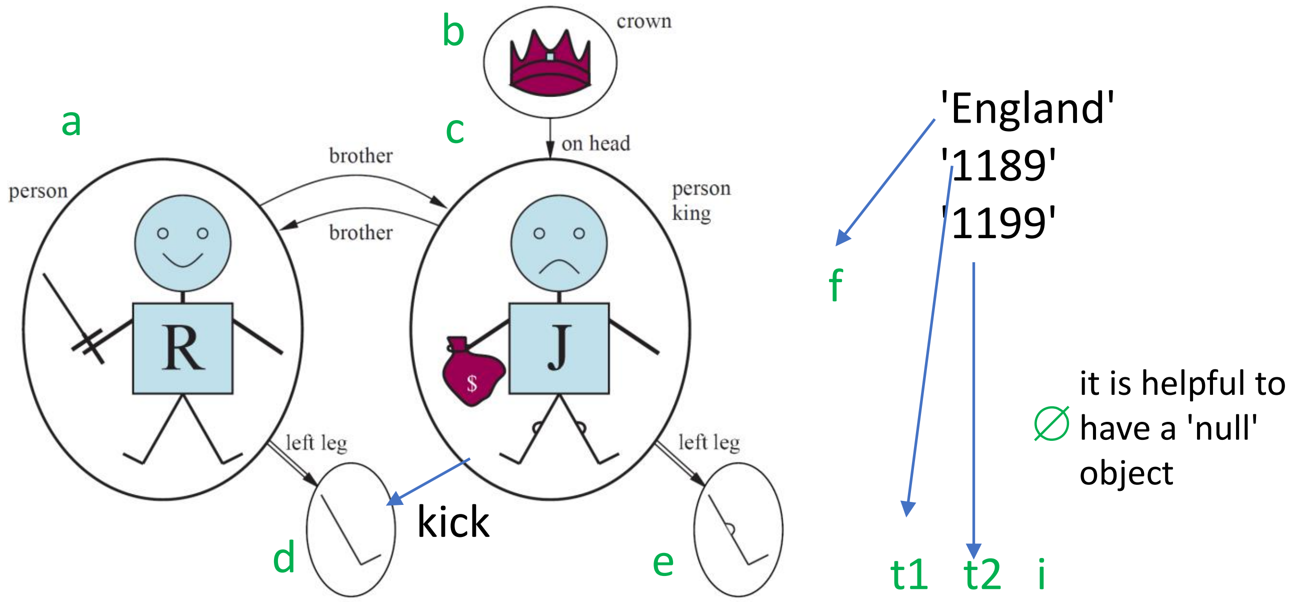
Semantics of FOL

- note: there are usually many, many models that could represent the KB

- although this sounds abstract, think of a model as an "**envisionment**" of what the KB describes (also known as an "interpretation")

Example of a Model

- KB={king(john),evil(john),ruler(john,England, interval(1189,1199)),
- brother(john,Richard),kick(john,leftLegOf(Richard), person(john),person(richard),
- ∀x,y brother(x,y)→brother(y,x),
- ∀x king(x)→∃y crown(y)^onHead(x,y) }



- model=<U,D,R>
- U=<a,b,c,d,e,f,t1,t2,i,∅> anonymous designators
- D: denotations={
  - constants: {'john'→c,'richard'→a,'England'→f,1189→t1, 1199→t2}
  - functions:
  - leftLegOf(.): {a→d, c→ e; b,d,e,f,t1,t2,i→∅}
  - interval(.,.): {<t1,t2>→i; all others <u,v>→∅}
- R: relations for each predicate:
  - $R_{brother}$={<a,c>,<c,a>}
  - R_{evil}={<c>} ; R_{crown} = {<b>}
  - R_{ruler}={<c,f,i>}
  - R_{person}={<c>,<a>}

Sematics of FOL

- there are other models...
  - with more (unmentioned objects)
  - where richard also has a crown
  - where richard also kicks john
  - where the crown has a brother...
- but
  - some models are not consistent with the KB
  - for example, if john was richard's brother, but richard was not john's brother, i.e. $\langle a,c\rangle \in R_{brother}$ but $\langle c,a\rangle \notin R_{brother}$
  - the reflexive axiom for *brother* constrains which models satisfy the KB
  - in fact, models with $R_{brother}=\{\langle a,c\rangle\langle c,a\rangle,\langle b,e\rangle,\langle e,b\rangle\}$ are OK too

FOL Truth Conditions

- remember in Prop Log, we used truth tables to evaluate the truth value of any sentence, given a model (composed ground-up from propositions)
- In FOL, if m=<U,D,R> (and there are no free vars in P,Q) then:
  - sat(m,pred($<t_1,\ldots,t_n>$)) iff $<d(t_1),\ldots,d(t_n)> \in R_{pred}$
  - sat(m,$\neg$s) iff sat(m,s) is false
  - sat(m,P^Q) iff sat(m,P) and sat(m,Q)
  - sat(m,PvQ) iff sat(m,P) or sat(m,Q)
  - sat(m,P$\rightarrow$Q) iff sat(m,$\neg$P) or sat(m,Q)
  - sat(m,$\forall$x P(x)) iff for every o$\in$U, sat(m,P(x/o)) where x is substituted by o
  - sat(m,$\exists$x P(x)) iff for some o$\in$U, sat(m,P(x/o)) where x is substituted by o
  - for any sentence P(...x...) containing x

Semantics of FOL

- using the truth conditions, you should be able to prove that:
  - $\neg\forall x\ P(x) \equiv \exists x\ \neg P(x)$ (semantically equivalent)
  - $\neg\exists x\ P(x) \equiv \forall x\ \neg P(x)$
  - you have to show this holds *for all models*

Entailment

- this is the key idea underlying inference
  - entailment = “logical consequence” of a KB
- $\alpha \models \beta$ *iff all models of* α *also satisfy* β (same as in Prop Log)
- the problem is that there are many more models in FOL (possibly infinite, possibly uncountable) (not just 2^n)

- (a bit of related theory that you don't need to know...)
- Lowenheim-Skolem Theorem (paraphrased): For any finite, consistent set of first-order sentences, there always exists models of infinite size

Inference in FOL

- unlike Prop Log, we can't do model-checking (because the number of models is not finite)
- thus we NEED to use sound rules of inference to show that a sentence is entailed *purely by syntactic manipulation*
- most of the ROI from Prop Log carry over to FOL
- there are some new rules (e.g. related to quantifiers)
- the main new concept is unification, for dealing with variable when doing pattern matching (e.g. of sub-sentences)

Inference in FOL

ROI	from this...	derive this...
AndElimination (AE)	A^B	A
AndIntroduction (AI)	A, B	A^B
OrIntroduction	A, B	AvB
Commutativity	A^B	B^A
Distributivity	Av(B^C) A^(BvC)	(AvB)^(AvC) (A^B)v(A^C)
DoubleNegationElim (DN)	¬¬A	A
DeMorgan's Laws (DM)	¬(AvB) ¬(A^B)	¬A^¬B ¬Av¬B
ImplicationElimination (IE)	A→B	¬AvB
contraposition	A→B	¬B→¬A
Modus Ponens (MP)	A, A→B	B
Modus Tolens	A→B, ¬B	¬A
Resolution	AvB, ¬AvC	BvC
Universal Instantiation	∀x P(x)	P(c) for any const c
Existential Instantiation	∃x P(x)	P(c) for NEW const c

these work the same in FOL as in Prop Log

these need to be adapted to handle variables

new rules

Inference in FOL

- 2 new ROI
  - these can be used to make '<u>ground sentences</u>', or versions of quantified sentences with variable replaced by specific constants
- Universal Instantiation (UI)

  <u>∀x P(x)</u>   any sentence P containing x

  P(c)   where variable x is replaced with any constant c

  - example:

  <u>{∀x parent(x)→∃y child(y,x)}</u>

  parent(homer) →∃y child(y,homer)

  parent(fido) →∃y child(y,fido)

  parent(ReliantStadium) →∃y child(y,ReliantStadium) // nonsense, but still true

Inference in FOL

- Existential Instantiation (EI)

  $\exists$x P(x) any sentence P containing x

  P(c) where variable x is replaced with any *new* constant c *that does not appear anywhere else in the KB*

  - c is called a 'skolem constant'; it is like introducing an anonymous name for the object
- example:
  - {$\exists$x car(x)^owns(john,x)} ⊢ {car(car_{57})^owns(john,car_{57})}
  - where car_{57} is a made-up new symbol denoting the thing that exists
  - if you use any existing symbol, it doesn't work: owns(john,the_alamo)
  - in LISP, there is a 'gensym' function to create new symbols: owns(john,_X454912)

Unification

- MP and Reso involve pattern matching
- need to extend them to handle variables
- example:
  - KB = { $\forall$*x dog(x)→mammal(x), dog(fido)* }
  - we want to conclude KB ╞ *mammal(fido)* by MP, but technically, '*dog(fido)*' does not match the antecedent '*dog(x)*'
  - however, they would match if 'x' were substituted by 'fido'

Unification

- a *variable-substitution list* is a mapping of variables to terms,
  Var Ⱶ> Term
  - example: u={X/fido}
  - vars can map to constants, other vars, or functions
  - u={X/fido , Y/snoopy , U/V , Z/sqrt(2) , R/f(P,Q) , M/mother(bill)}
- a *unifier* of 2 expressions P and Q is a substitution-list that makes P and Q syntactically identical
  - P=dog(X), Q=dog(fido),
  - u={X/fido},
  - P'=subst(u,P)=dog(fido),
  - Q'=subst(u,Q)=dog(fido),
  - hence P'=Q'

Unification

- another example:
  - P=eats(X,dogfood); Q=eats(fido,Y)
  - unify(P,Q)=u where u={X/fido,Y/dogfood}
  - subst(u,P)=subst(u,Q)=eats(fido,dogfood)
- another example:
  - P = gives(bill,mother(bill),B,T,V) Q = gives(P,Q,present,R,V)
  - 3 alternative unifiers:
  - u1={P/bill, Q/mother(bill), B/present, T/R} // don't need to bind V
  - u2={P/bill, Q/mother(bill), B/present, T/R,V/3} // also works, but not necessary
  - u3={P/bill, Q/mother(bill), B/present, R/S, T/S} // also works, variable renaming

Unification

- negative examples that do not unify:
  - no substitution will make these identical; i.e. unify(P,Q)=fail
  - P=loves(bill,mother(bill)), Q=loves(X,X)
  - P=move(blockA,stack1,X), Q=move(Y,X,stack2)
  - P=lessThan(6,7), Q=lessThan(X,succ(X))
  - P=match(X,X), Q=match(Y,f(Y))
    - after binding X to Y, then X cannot be bound to f(X) which contains it
- most-general unifier (MGU) of P and Q
  - the unifier that makes the least commitments (no unnec. variable bindings)
  - the MGU always exists and is unique (modulo variable renaming)

Unification Algorithm

- given 2 expressions (FOL predicates or sentences), how to determine whether they are unifiable, and if so, what is the MGU?
- the gist of the algorithm:
  - imagine P and Q as parse trees
  - start with an empty substitution list and add variable bindings as you go
  - do a left traversal of the parse trees
  - whenever you see a variable in one tree
    - check to see if it is already bound
    - if not bind it to the corresponding subtree in the other expression

Unification Algorithm

- the algorithm treats each expression as a nested list, like "[loves fido [owner fido]]", which is a list of 3 terms, the last of which is a list of 2 terms
- (like S-expressions)
- the algorithm is recursive; if it can match element i in each list, it proceeds with trying to match elements i+1
- UnifyVar subroutine tries to add a binding of var to x in the current substitution list
- first, it checks of var or x already have substitutions
- it also checks that var does not occur inside of x, e.g. can't bind Z to f(Z)

function UNIFY(x, y, θ=*empty*) **returns** a substitution to make x and y identical, or *failure*
  if θ = *failure* **then return** *failure*
  else if $x = y$ **then return** θ
  else if VARIABLE?(x) **then return** UNIFY-VAR(x, y, θ)
  else if VARIABLE?(y) **then return** UNIFY-VAR(y, x, θ)
  else if COMPOUND?(x) **and** COMPOUND?(y) **then**
    return UNIFY(ARGS(x), ARGS(y), UNIFY(OP(x), OP(y), θ))
  else if LIST?(x) **and** LIST?(y) **then**
    return UNIFY(REST(x), REST(y), UNIFY(FIRST(x), FIRST(y), θ))
  else return *failure*

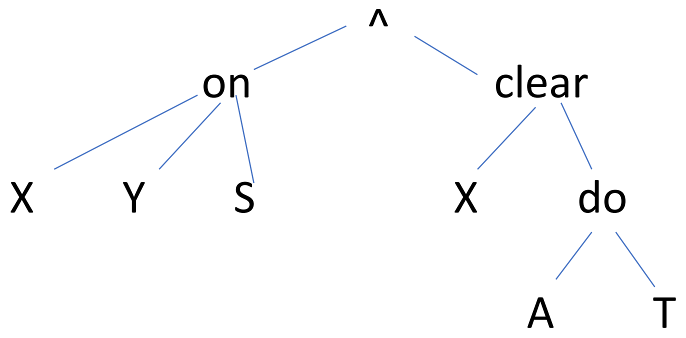
function UNIFY-VAR(var, x, θ) **returns** a substitution
  if $\{var/val\} \in \theta$ for some val **then return** UNIFY(val, x, θ)
  else if $\{x/val\} \in \theta$ for some val **then return** UNIFY(var, val, θ)
  else if OCCUR-CHECK?(var, x) **then return** *failure*
  else return add $\{var/x\}$ to θ

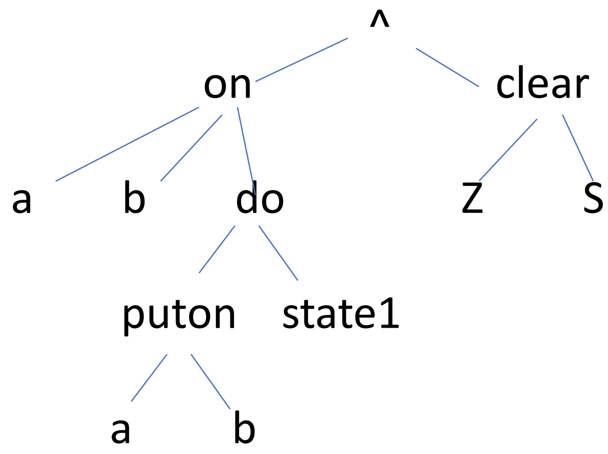
Unification Algorithm

in this example, capital letters are variables and lower-case are constants

this example describes
block a on block b in situation S,
which is the successor of doing
a puton action in a predecessor state T

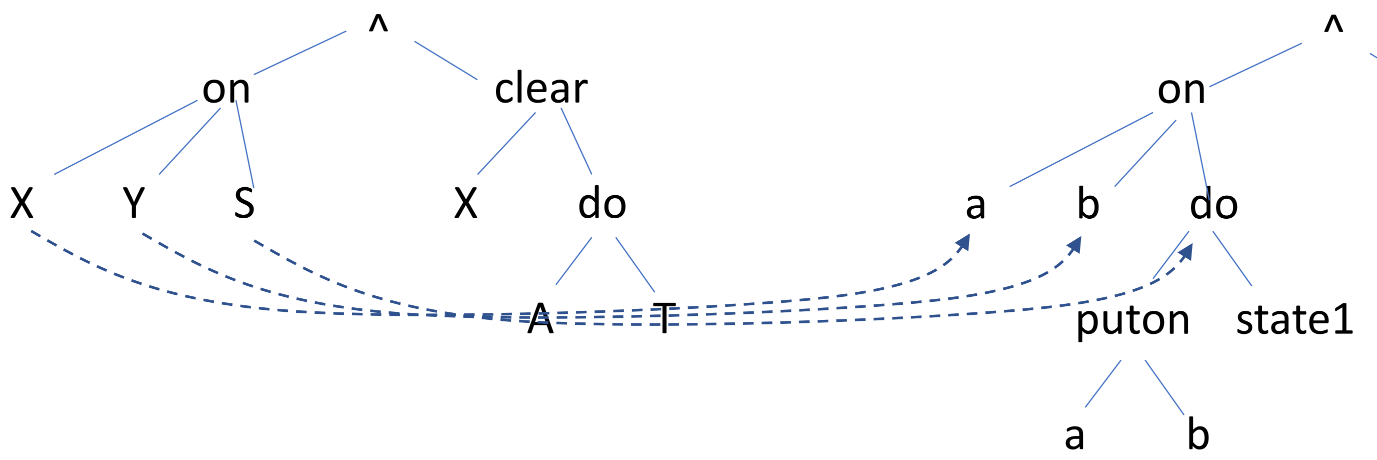
- P = on(X,Y,S)^clear(X,do(A,T))
- Q = on(a,b,do(puton(a,b),state1))^clear(Z,S)
- u={X/a , Y/b , Z/a , S/do(puton(a,b),state1) , A/puton(a,b) , T/state1 }
- subst(u,P) = on(a,b,do(puton(a,b),state1))^clear(a,do(puton(a,b),state1))





Unification Algorithm

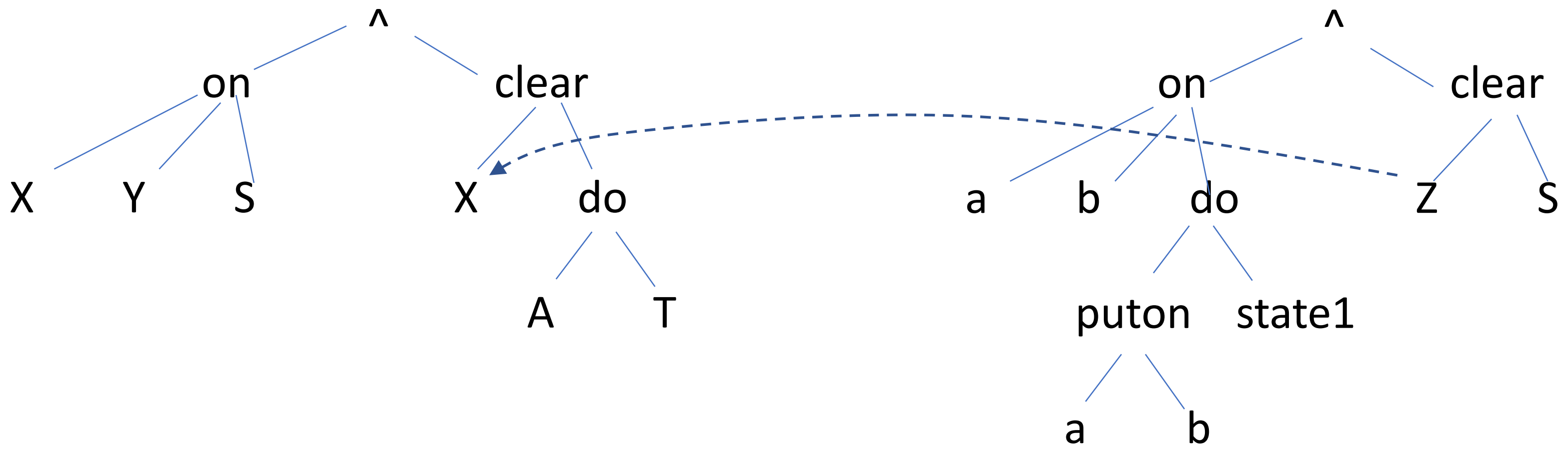
- P = on(X,Y,S)^clear(X,do(A,T))
- Q = on(a,b,do(puton(a,b),state1))^clear(Z,S)
- u={X/a , Y/b , Z/a , S/do(puton(a,b),state1) , A/puton(a,b) , T/state1 }
- subst(u,P) = on(a,b,do(puton(a,b),state1))^clear(a,do(puton(a,b),state1))



u={X/a}
u={X/a, Y/b}
u={X/a, Y/b,
S/do(puton(a,b),state1)}

Unification Algorithm

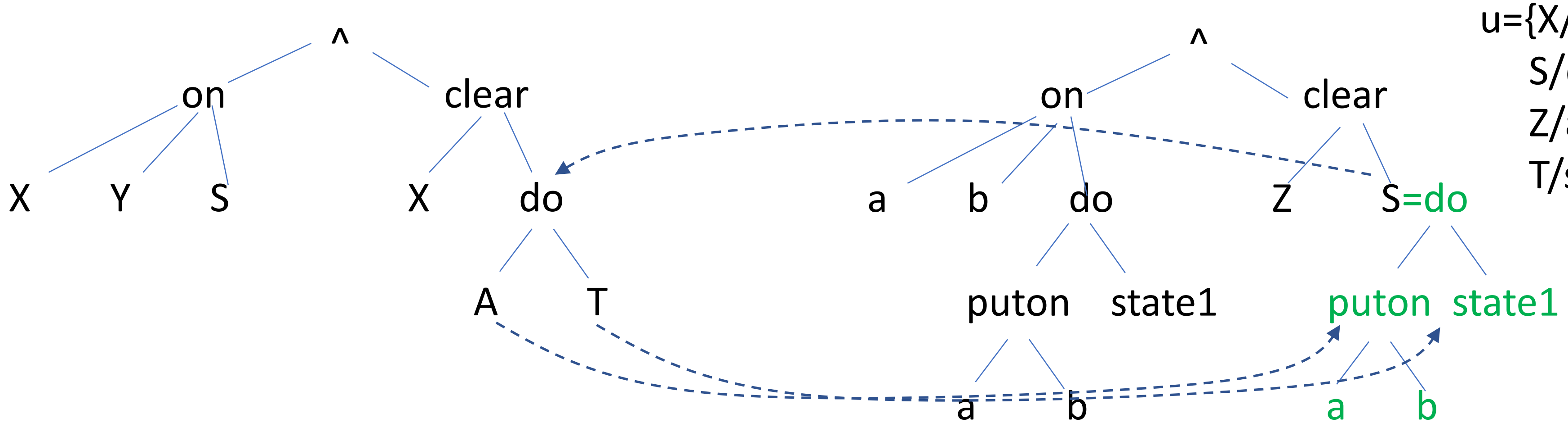
- P = on(X,Y,S)^clear(X,do(A,T))
- Q = on(a,b,do(puton(a,b),state1))^clear(Z,S)
- u={X/a , Y/b , Z/a , S/do(puton(a,b),state1) , A/puton(a,b) , T/state1 }
- subst(u,P) = on(a,b,do(puton(a,b),state1))^clear(a,do(puton(a,b),state1))



u={X/a, Y/b,
S/do(puton(a,b),state1),
Z/X}
"substitute through X to a"
u={X/a, Y/b,
S/do(puton(a,b),state1),
Z/a}

Unification Algorithm

- P = on(X,Y,S)^clear(X,do(A,T))
- Q = on(a,b,do(puton(a,b),state1))^clear(Z,S)
- u={X/a , Y/b , Z/a , S/do(puton(a,b),state1) , A/puton(a,b) , T/state1 }
- subst(u,P) = on(a,b,do(puton(a,b),state1))^clear(a,do(puton(a,b),state1))



u={X/a, Y/b,
S/do(puton(a,b),state1),
Z/a, A/puton(a,b),
T/state1}

Generalized Modus Ponens (GMP)

- from {P’, ∀...P→Q} derive Q’=subst(u,Q) where u=unify(P,P’)
- in other words...
  - if P’ unifies with the antecedents of the rule, where u is the unifier, then derive the consequent, but apply the unifier to it
- example 1:

  ∀X,Y cat(X)^mouse(Y)→chase(X,Y)

  cat(scratchy)^mouse(itchy)

  chase(scratchy,itchy)   using u={X/scratchy, Y/itchy}
- example 2:

  ∀M loves(M,M)→narcissist(M)

  loves(fonzie,fonzie)

  narcissist(fonzie)   using u={M/fonzie}

  note - this does not work for loves(joanie,chachi), does not unify with loves(M,M)

Natural Deduction Proofs in FOL

Example knowledge base

The law says that it is a crime for an American to sell weapons to hostile nations. The country Nono, an enemy of America, has some missiles, and all of its missiles were sold to it by Colonel West, who is American.

Prove that Col. West is a criminal

- It is a crime for an American to sell weapons to a hostile nation.
  1. $\forall X,Y,Z$ *american(X)*$\wedge$*weapon(Y)*$\wedge$*hostile(Z)*$\wedge$*sells(X,Y,Z)*$\rightarrow$*criminal(X)*
- Nono has some missiles.
  2. $\exists B$ *owns(nono,B)*$\wedge$*missile(B)*
- All of Nono's missiles were sold to it by Colonel West.
  3. $\forall C$ *owns(nono,C)*$\wedge$*missile(C)*$\rightarrow$*sells(west,C,nono)*
- Missiles are weapons.
  4. $\forall D$ *missile(D)*$\rightarrow$*weapon(D)*
- An enemy of America counts as "hostile".
  5. $\forall E$ *enemy(E,america)*$\rightarrow$*hostile(E)*
- The country Nono is an enemy of America.
  6. *enemy(nono,america)*
- Colonel West is an American.
  7. *american(west)*

Natural Deduction proof in FOL (with unifiers)

8. *hostile(nono)* [MP,5,6] θ={E/nono}

9. *owns(nono,m_1)$\wedge$missile(m_1)* [ExInst ,2] θ={B/m_1} skolem constant

10. *missile(m_1)* [AndElim,9]

11. *weapon(m_1)* [MP, 10,4] θ={D/m_1}

12. *sells(west,m_1,nono)* [MP, 3,9] θ={C/m_1}

13. american(west) ^ weapon(m1) ^ hostile(nono) ^ sells(west,m1,nono) [AndIntro, 7,8,11,12]

14. *criminal(west)* [MP,1,13] θ={X/west,Y/m_1,Z/nono}

(From previous page…)

1. *$\forall$X,Y,Z american(X)$\wedge$weapon(Y)$\wedge$hostile(Z)$\wedge$sells(X,Y,Z)$\rightarrow$criminal(X)*

Generalized Resolution

- from {PvQ, ¬P'vR} derive Q'vR'=subst(u,QvR) where u=unify(P,P')
- in other words...
  - if P and P' are two opposite literals that unify, and unifier is u, then combine the remaining literals and apply the substitution
- example:
  - clause 1: ¬dog(X) v mammal(X)
  - clause 2: ¬mammal(Y) v animal(Y)
  - resolvent(P,Q): ¬dog(Y) v animal(Y) after applying u={X/Y}

Resolution

- generalized resolution - with unifiers

Full first-order version:

$$\frac{\ell_1 \vee \cdots \vee \ell_k, \qquad m_1 \vee \cdots \vee m_n}{(\ell_1 \vee \cdots \vee \ell_{i-1} \vee \ell_{i+1} \vee \cdots \vee \ell_k \vee m_1 \vee \cdots \vee m_{j-1} \vee m_{j+1} \vee \cdots \vee m_n)\theta}$$

where $\text{UNIFY}(\ell_i, \neg m_j) = \theta$.

For example,

$$\frac{\begin{array}{l}\neg Rich(x) \vee Unhappy(x) \\ Rich(Ken)\end{array}}{Unhappy(Ken)}$$

with $\theta = \{x/Ken\}$

Apply resolution steps to $CNF(KB \wedge \neg\alpha)$; complete for FOL

Conversion to CNF ...in FOL

Everyone who loves all animals is loved by someone:

$\forall x ([\forall y \;\; Animal(y) \Rightarrow Loves(x,y)] \Rightarrow [\exists y \;\; Loves(y,x)])$

1. Eliminate biconditionals and implications

$\forall x \;\; [\neg \forall y \;\; \neg Animal(y) \vee Loves(x,y)] \vee [\exists y \;\; Loves(y,x)]$

2. Move $\neg$ inwards: $\neg \forall x, p \;\; \equiv \exists x \;\; \neg p$, $\neg \exists x, p \;\; \equiv \forall x \;\; \neg p$:

$\forall x \;\; [\exists y \;\; \neg(\neg Animal(y) \vee Loves(x,y))] \vee [\exists y \;\; Loves(y,x)]$

$\forall x \;\; [\exists y \;\; \neg\neg Animal(y) \wedge \neg Loves(x,y)] \vee [\exists y \;\; Loves(y,x)]$

$\forall x \;\; [\exists y \;\; Animal(y) \wedge \neg Loves(x,y)] \vee [\exists y \;\; Loves(y,x)]$

Conversion to CNF contd. ...in FOL

3. Standardize variables: each quantifier should use a different one

$$\forall x \;\; [\exists y \;\; Animal(y) \land \neg Loves(x,y)] \lor [\exists z \;\; Loves(z,x)]$$

4. Skolemize: a more general form of existential instantiation.

   Each existential variable is replaced by a Skolem function
   of the enclosing universally quantified variables:

$$\forall x \;\; [Animal(F(x)) \land \neg Loves(x,F(x))] \lor Loves(G(x),x)$$

5. Drop universal quantifiers:

$$[Animal(F(x)) \land \neg Loves(x,F(x))] \lor Loves(G(x),x)$$

6. Distribute $\land$ over $\lor$:

$$[Animal(F(x)) \lor Loves(G(x),x)] \land [\neg Loves(x,F(x)) \lor Loves(G(x),x)]$$

find a pair of clauses that are unifiable
e.g C_i=*¬pineTree(P) v plant(P)*
C_j=*pineTree(christmasTree29)*
they unify provided *P=christmasTree29*

```
function PL-RESOLUTION(KB, α) returns true or false
   inputs: KB, the knowledge base, a sentence in propositional logic
           α, the query, a sentence in propositional logic

   clauses ← the set of clauses in the CNF representation of KB ∧ ¬α
   new ← { }
   loop do
      for each pair of clauses Ci, Cj in clauses do
         resolvents ← PL-RESOLVE(Ci, Cj)
         if resolvents contains the empty clause then return true
         new ← new ∪ resolvents
      if new ⊆ clauses then return false
      clauses ← clauses ∪ new
```

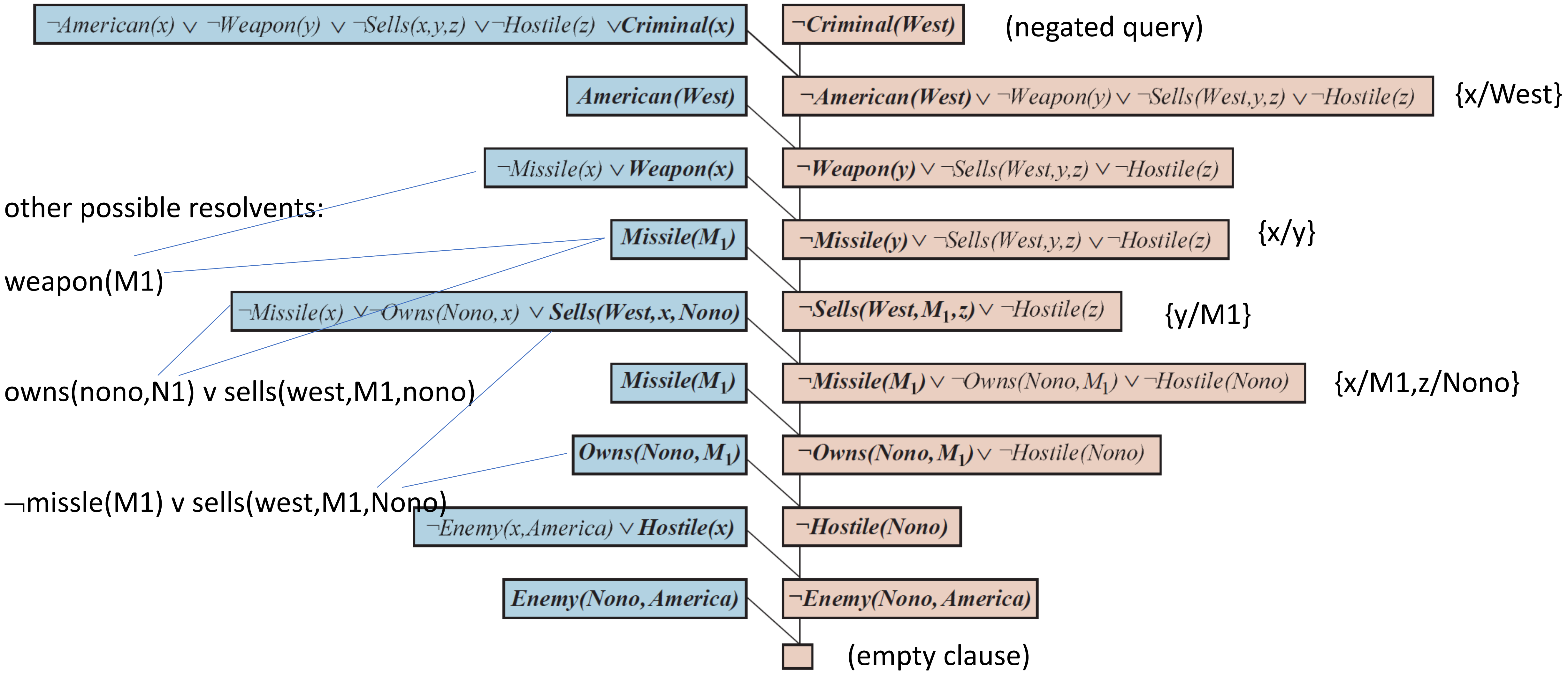
don't forget to negate the query

apply unifier θ to remaining literals to generate the “resolvent”:e.g. *plant(christmasTree29)*

termination: we are looking to generate the empty clause

KB converted to CNF

clauses derived by resolution



Resolution Strategies (Search Heuristics)

- Unit preference
  - choose pairs of clauses where one of them is a single literal
  - why? because will reduce length of other clause
- Set of Support
  - initially identify a subset of clauses likely to contain the inconsistency (e.g. the negated query)
  - with each iteration, choose one of the clauses from SOS, and add resolvent to SOS
  - example: in Wumpus World, focus only on clauses involving rooms (x,y) where x and y are restricted to 1-3
  - generates "goal-directed proofs", without deriving a lot of irrelevant conclusions from a large KB

Resolution Strategies

- Input resolution
  - always choose one of the clauses from the Input (KB or facts) - never resolve 2 derived clauses
  - restricted space of proof trees with a "spine" (see Col. West example)
  - efficient, but not complete (except for Horn clause KBs)
- Linear resolution
  - a variant of Input resolution
  - allow clauses to be resolved if one of them is in Input, or if one is an ancestor of the other
  - complete

Completeness of Resolution

- Recall that Reso in Prop Logic is complete - because of Ground Resolution Theorem:
  - If a set of Prop clauses S is unsatisfiable, the empty clause is in the Resolution Closure, so there exists a finite sequence of resolution steps that will generate the empty clause □
- To prove this for FOL, we need to take unification into account (for variables)
- Herbrand's Theorem:
  - If a set of FOL clauses S is unsatisfiable, then there exists a finite set of ground instances that is unsatisfiable
- combine this with the Ground Resolution Theorem and the Lifting Lemma to show that □ can be derive from the original clauses S (with variables)

for example: think of converting
$\exists x$ *missle(x)* and $\forall y$ *¬missile(y) v weapon(y)*
to: *missile(m_1)* and *¬missile(m_1) v weapon(m_1)*

Completeness of Resolution

- Herbrand Universe: set of all constants and functions of constants
  - a,b,c,f(a),f(b),f(f(a))…
- Herbrand base: set of all ground clauses made by using objects from Herbrand Universe as arguments
  - dog(a)→mammal(a)
  - dog(b)→mammal(b)
  - dog(f(a))→mammal(f(a))
  - …
- Lifting Lemma: once you have the structure of a proof of □ using ground sentences, you can put the variables back in to the same proof structure

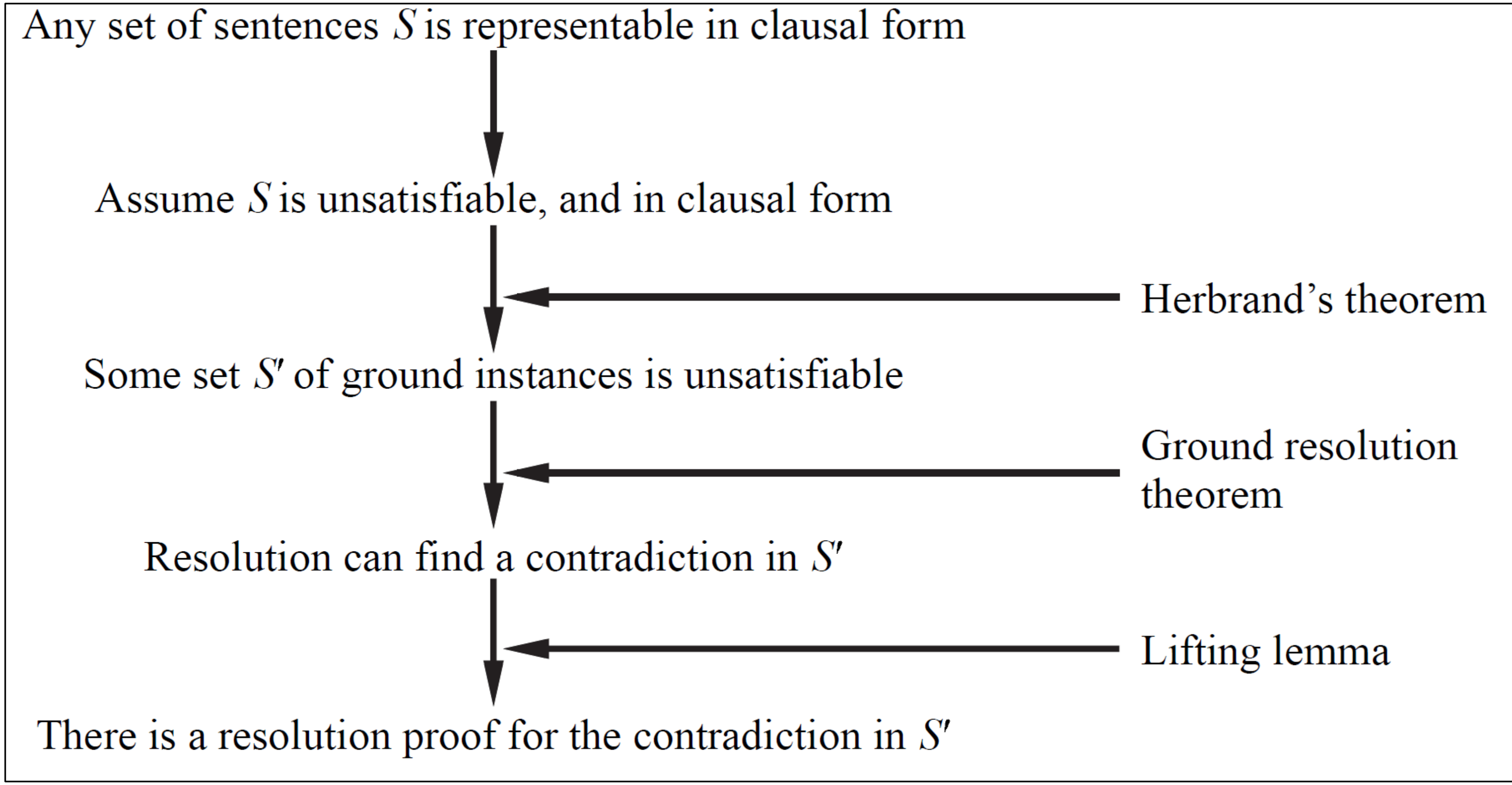


Illustration of Herbrand's Theorem

- Consider the FOL theory for Col West:

  1. $\forall X,Y,Z$ $american(X) \wedge weapon(Y) \wedge hostile(Z) \wedge sells(X,Y,Z) \rightarrow criminal(X)$
  2. $\exists B$ $owns(nono,B) \wedge missile(B)$
  3. $\forall C$ $owns(nono,C) \wedge missile(C) \rightarrow sells(west,C,nono)$
  4. $\forall D$ $missile(D) \rightarrow weapon(D)$
  5. $\forall E$ $enemy(E,america) \rightarrow hostile(E)$
  6. $enemy(nono,america)$
  7. $american(west)$

- We want to show KB|=criminal(west) by Resolution. Can we count on a derivation of □ by a finite number of steps?

- Herbrand says the FOL KB is equivalent to a collection of ground sentences where exist. vars are skolemized and univ. vars are replaced by <u>all possible constants</u>...

  4. // $\forall D$ $missile(D) \rightarrow weapon(D) \equiv$
  $missle(west) \rightarrow weapon(west)$
  $missle(nono) \rightarrow weapon(nono)$
  $missle(america) \rightarrow weapon(america)$
  $missle(m1) \rightarrow weapon(m1)$ *

  5. // $\forall E$ $enemy(E,america) \rightarrow hostile(E) \equiv$
  $enemy(west,America) \rightarrow hostile(west)$
  $enemy(nono,America) \rightarrow hostile(nono)$ *
  $enemy(america,America) \rightarrow hostile(america)$
  $enemy(m1,America) \rightarrow hostile(m1)$

  1.// *we would have all combinations of X,Y,Z...*
  $american(m1) \wedge weapon(m1) \wedge hostile(m1) \wedge sells(m1,m1,m1) \rightarrow criminal(m1)$
  ...
  $american(west) \wedge weapon(m1) \wedge hostile(nono) \wedge sells(west,m1,nono) \rightarrow criminal(west)$ *
  ...

- Most of these are irrelevant and silly, but they exist in principle.
- Our proof only relies on only selected ground instances (marked by asterisks)

Illustration of Herbrand's Theorem

- select just the right ground sentences (and add negated query):

  american(west)∧weapon(m1)∧hostile (nono)∧sells(west,m1,nono) →criminal(west)

  owns(nono,m1)∧missile(m1)

  owns(nono,m1)∧missile(m1) →sells(west,m1,nono)

  missile(m1)→weapon(m1)

  enemy(nono,america)→hostile(nono)

  enemy(nono,america)

  american(west)

  ¬criminal(west)

- propositionalize:

  american_west∧weapon_m1∧hostile _nono∧sells_west_m1_nono →criminal_west

  owns_nono_m1∧missile_m1

  owns_nono_m1∧missile_m1 →sells_west_m1_nono

  missile_m1→weapon_m1

  enemy_nono_america→hostile_nono

  enemy_nono_america

  american_west

  ¬criminal_west

- add negated query and convert to CNF:

  ¬american_west v ¬weapon_m1 v ¬hostile_nono v ¬sells_west_m1_nono v criminal_west

  owns_nono_m1

  missile_m1

  ¬owns_nono_m1 v ¬missile_m1 v sells_west_m1_nono

  ¬missile_m1 v weapon_m1

  ¬enemy_nono_America v hostile_nono

  enemy_nono_america

  american_west

  ¬criminal_west

Illustration of Herbrand's Theorem

- do resolution proof in propositional logic:
  1. ¬american_west v ¬weapon_m1 v ¬hostile_nono v ¬sells_west_m1_nono v criminal_west
  2. owns_nono_m1
  3. missile_m1
  4. ¬owns_nono_m1 v ¬missile_m1 v sells_west_m1_nono
  5. ¬missile_m1 v weapon_m1
  6. ¬enemy_nono_America v hostile_nono
  7. enemy_nono_america
  8. american_west
  9. ¬criminal_west
  10. ¬missile_m1 v sells_west_m1_nono [res, 2&4]
  11. sells_west_m1_nono [res, 3&10]
  12. hostile_nono [res, 6&7]
  13. weapon_m1 [res, 3&5]
  14. ¬weapon_m1 v ¬hostile_nono v ¬sells_west_m1_nono v criminal_west [res, 8&1]
  15. ¬hostile_nono v ¬sells_west_m1_nono v criminal_west [res, 14&13]
  16. ¬sells_west_m1_nono v criminal_west [res, 15&12]
  17. criminal_west [res, 16&11]
  18. □ [res, 17&9]

- Lifting the same proof structure back to FOL (in CNF) with unification:
  1. ¬american(X) v ¬weapon(Y) v ¬hostile(Z) v ¬sells(X,Y,Z) v criminal(X)
  2. owns(nono,m1)
  3. missile(m1)
  4. ¬owns(nono,C) v ¬missile(C) v sells(west,C,nono)
  5. ¬missile(D) v weapon(D)
  6. ¬enemy(E,america) v hostile(E)
  7. enemy(nono,america)
  8. american(west)
  9. ¬criminal(west)
  10. ¬missile(m1) v sells(west,m1,nono) [res, 2&4] {C/m1}
  11. sells(west,m1,nono) [res, 3&10]
  12. hostile(nono) [res, 6&7] {E/nono}
  13. weapon(m1) [res, 3&5] {D/m1}
  14. ¬weapon(m1) v ¬hostile(Y) v ¬sells(west,Y,Z) v criminal(west) [res, 8&1] {X/west}
  15. ¬hostile(Z) v ¬sells(west,m1,Z) v criminal(west) [res, 14&13] {Y/m1}
  16. ¬sells(west,m1,nono) v criminal(west) [res, 15&12], {Z/nono}
  17. criminal_west [res, 16&11]
  18. □ [res, 17&9]

Complexity of Resolution

- Recall that showing entailment by Resolution Refutation proofs in Propositional Logic is *NP-complete*
- FOL is only *semi-decidable*
  - if entailed ($\alpha \models \beta$), we could prove it (in theory, Herbrand’s Theorem)
  - if β is not entailed, cannot guarantee we can prove it (because of *Gödel's Incompleteness Theorem*)
- thus we say that Inference in FOL is "refutation-complete"
- computational complexity could be much worse than NP (depending on syntactic restictions on variables, functions, operators...)
  - e.g. satisfiability of quantified Boolean formulas (QBF) is PSPACE-complete

Forward-Chaining in FOL

- it works like it did in PropLog, but now we have to do unification when matching antecedents in rules, and keep track of variable bindings
- implementations
  - Rete algorithm: efficient way to store KB as a graph and determine which rules can fire, activating other nodes...
  - JESS – Java-based system in which you can build applications that use FC to make intelligent decisions

Forward Chaining Systems

- also known as Production Systems or Expert Systems
  - e.g. diagnosis systems for medical, financial/corporate, or mechanical systems
  - also used for cognitive models of reasoning (e.g. ACT, SOAR)
    - model of long-term and short-term memory, with activation of concepts by association
- one advantage of ES is that they can generate *explanations* of their recommendations (i.e. a proof-tree showing the rules and facts that were used to support their conclusions)
- restriction: knowledge based must consist of facts and conjunctive rules (including universal quantifiers but not existential)

Conjunctive Rules in FOL

> *Note: standardize your variable apart between rules.*
> *If you use 'X' as a variable in multiple rules, replace each instance with a unique version (subscript).*
> *For example:*
> $\forall x\ dog(x) \rightarrow mammal(x)$
> $\forall x\ cat(x) \rightarrow mammal(x)$
> *becomes*
> $\forall x_1\ dog(x_1) \rightarrow mammal(x_1)$
> $\forall x_2\ cat(x_2) \rightarrow mammal(x_2)$
> *That way, there will be less confusion during unification.*

- many KBs have rules of this form
- $\forall x,y\ [\exists z\ P(..) \wedge Q(..) \wedge R(..)] \rightarrow S(..)$
  - LHS (antecedents) has to be a *conjunction* of *positive literals* (no negations)
  - Universally quantified variables (appear in both antecedents and consequent)
  - LHS can also have extra variables ($\exists z$), typically existentially quantified
- $\forall x\ [\exists z\ int(x) \wedge int(z) \wedge factor(z,x) \wedge 1<z<x] \rightarrow compositeNumber(x)$
- conjunctive rules are <u>equivalent to Definite Clauses</u>
  - convert conjunctive rule to CNF (note the scoping during Impl. Elim.!)

$\forall x,y\ [\exists z\ P(..) \wedge Q(..) \wedge R(..)] \rightarrow S(..)$
$\forall x,y\ \neg[\exists z\ P(..) \wedge Q(..) \wedge R(..)]\ v\ S(..)$
$\forall x,y\ [\forall z\ \neg(P(..) \wedge Q(..) \wedge R(..))]\ v\ S(..)$
$\forall x,y\ [\forall z\ \neg(P(..) \wedge Q(..) \wedge R(..))]\ v\ S(..)$
$\forall x,y\ [\forall z\ \neg P(..)\ v\ \neg Q(..)\ v\ \neg R(..)]\ v\ S(..)$
$\forall x,y,z\ \neg P(..)\ v\ \neg Q(..)\ v\ \neg R(..)\ v\ S(..)$ - definite clause, 1 pos. lit.

Forward chaining algorithm

"agenda"; initialize with known facts; add new facts as they are inferred

function FOL-FC-ASK(KB, α) **returns** a substitution or *false*

repeat until *new* is empty
  $new \leftarrow \{\}$
  for each sentence r **in** KB **do**
    $(p_1 \wedge \ldots \wedge p_n \Rightarrow q) \leftarrow$ STANDARDIZE-APART(r)
    for each θ such that $(p_1 \wedge \ldots \wedge p_n)\theta = (p'_1 \wedge \ldots \wedge p'_n)\theta$
        for some $p'_1, \ldots, p'_n$ in KB
      $q' \leftarrow$ SUBST(θ, q)
      if q' is not a renaming of a sentence already in KB or *new* **then do**
        add q' to *new*
        $\phi \leftarrow$ UNIFY(q', α)
        if ϕ is not *fail* **then return** ϕ
  add *new* to KB
return *false*

for each rule, like $\forall x$ dog(x)→mammal(x) replace with unique variable names, to avoid confusion with use of same variable in other rules, $\forall x_{101}$ dog(x_{101}) → mammal(x_{101})

Forward Chaining Example IN FOL

1. american(X)∧weapon(Y)∧hostile(Z)∧sells(X,Y,Z)→criminal(X)
2. owns(nono,C)∧missile(C)→sells(west,C,nono)
3. missile(D)→weapon(D)
4. <u>enemy(E,america)→hostile(E)</u>

agenda: initialized with facts

5. owns(nono,m1)
6. missile(m1)
7. enemy(nono,america)
8. american(west)
9. weapon(m1) // rule 3 fired, u={D/m1}
10. hostile(nono) // rule 4 fired, u={E/nono}
11. sells(west,m1,nono) // rule 2, u={C/m1}
12. criminal(west) // rule 1 fires

Because we had to convert KB to definite clauses,
∃B owns(nono,B)∧missile(B)
had to get made into a ground sentence by skolemization (EI):
owns(nono,m1)^missile(m1)
for a particular missle *m1*

Example: Kinship KB (Simpsons characters)

female(lisa)
female(marge)
male(bart)
male(homer)
male(tod)
male(rod)
male(flanders)

parent(bart,homer)
parent(bart,marge)
parent(lisa,homer)
parent(lisa,marge)
parent(rod,flanders)
parent(tod,flanders)

$\forall$x,y parent(x,y)^male(y)$\rightarrow$father(x,y)

$\forall$x,y parent(x,y)^female(x)$\rightarrow$daughter(y,x)

$\forall$x,y [$\exists$z parent(x,z)^parent(y,z)$\rightarrow$sibling(x,y)

interpret these as "father <u>of</u> x is y" etc.

food for thought: define rules for 'grandfather', 'cousin', 'aunt', 'related'...

Example: Kinship KB (Simpsons characters)

female(lisa)
female(marge)
male(bart)
male(homer)
male(tod)
male(rod)
male(flanders)

parent(bart,homer)
parent(bart,marge)
parent(lisa,homer)
parent(lisa,marge)
parent(rod,flanders)
parent(tod,flanders)

$\forall$x,y parent(x,y)^male(y)→father(x,y)
$\forall$x,y parent(x,y)^female(x)→daughter(y,x)
$\forall$x,y [$\exists$z parent(x,z)^parent(y,z)→sibling(x,y)

What new facts can we generate by **Forward Chaining**?

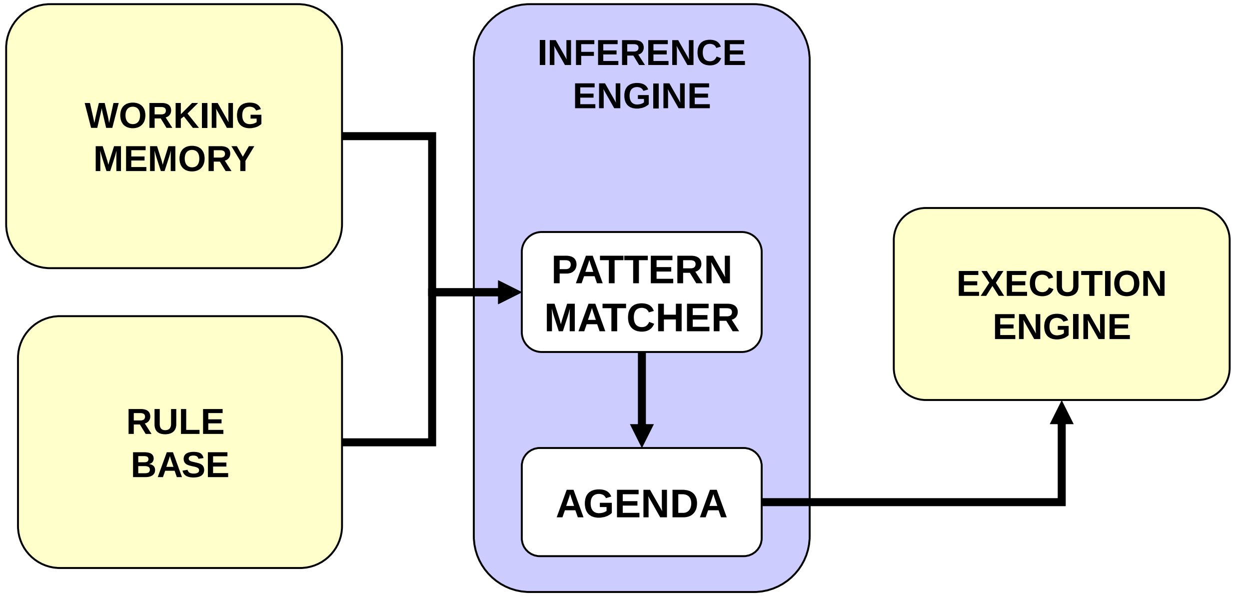
- find all combos of facts matching LHS of rules (try all var bindings)
- parent(bart,homer)^male(homer)→father(bart,homer)

father(bart,homer)
father(lisa,homer)
father(rod,flanders)
father(tod,flanders)
daughter(marge,lisa)
daughter(homer,lisa)

parent(bart,homer)^parent(lisa,homer)->sibling(bart,lisa)
sibling(bart,lisa)
sibling(lisa,bart)
sibling(rod,tod)
sibling(tod,rod)

what about sibling(bart,bart)?
to prevent this, add x≠y to the rule

Forward-Chaining System Architecture

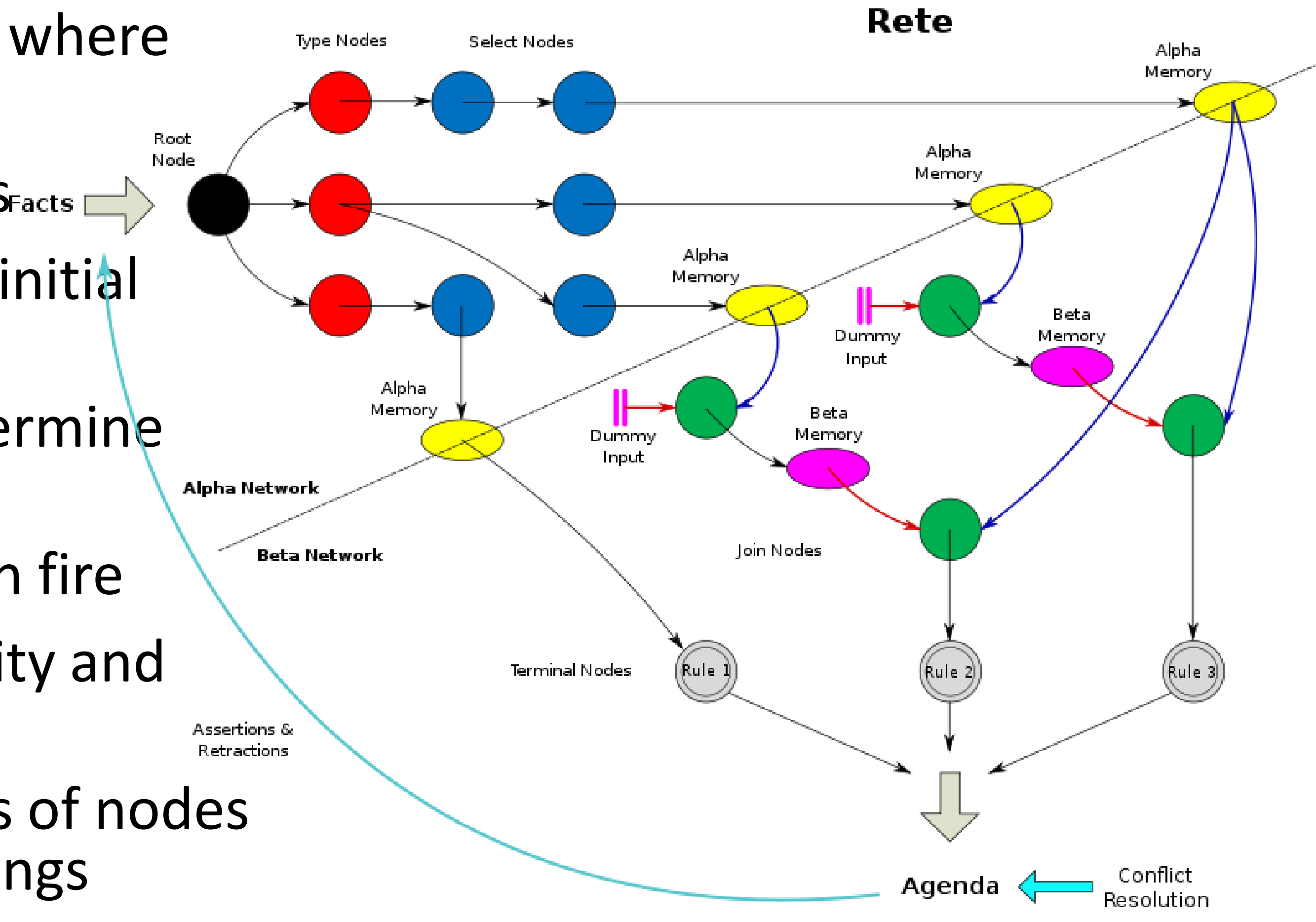


Rete Algorithm

alpha nodes essentially store lists of facts (tuples) matching the pattern of an antecedent in a rule

beta nodes perform "joins" of alpha nodes that will activate a rule, producing specifc new facts (tuples)

- representation of knowledge as a *network,* where nodes represent literals (predicates)
- rules link antecedent nodes to consequents
- start by *activating* nodes corresponding to initial facts
- uses efficient indexing of predicates to determine which rules can fire
- in each iteration, determine which rules can fire
- pick a rule (that can fire) with highest priority and modify the network
- rules with variables generate new instances of nodes for consequents with distinct variable bindings
- run until quiescence
- produces all the consequences of the facts



https://en.wikipedia.org/wiki/Rete_algorithm

Conflict Resolution

- a common issue in Forward Chaining that has to be dealt with
- What happens when two rules can fire that have opposite effects?
  - some rules can retract antecedents of other rules
  - e.g. one rule says *assert(P)* and the other says *retract(P)*
  - assign numeric priorities to rules – highest wins
- Subsumption Architecture (Rodney Brooks)
  - intelligent behavior in robots can be produced in a decentralized way by a lot of simple rules interacting
  - divide *behaviors* into lower-level basic survival behaviors that have higher priority, and higher-level goal-directed behaviors
  - example: 6-legged robot ants learning to walk

Ghenis:
https://en.wikipedia.org/wiki/Genghis_(robot)

CLIPS/JESS - implementations of FC using Rete

- C-Language Integrated Production System
  - developed at NASA
  - open source: http://clipsrules.sourceforge.net/
- JESS - Java Expert System Shell
  - developed by Ernest Friedman-Hill at Sandia (https://herzberg.ca.sandia.gov/)
  - Java implementation of Forward-Chaining and Rete algorithm
- can interface reasoning with GUI, controllers, etc.
- example of syntax for rules:

```
(defrule library-rule-1
  (book (name ?X)
        (status late)
        (borrower ?Y))
        (borrower (name ?Y)
        (address ?Z))
  =>
  (send-late-notice ?X ?Y ?Z))
```

CLIPS

- https://github.com/smarr/CLIPS
- Wine Expert - https://github.com/smarr/CLIPS/blob/master/examples/wine.clp
  - expert system for recommending wine pairings with food

```
(rule (if has-sauce is yes and sauce is spicy) (then best-body is full))

(rule (if tastiness is delicate) (then best-body is light))

(rule (if has-sauce is yes and sauce is cream)
    (then best-body is medium with certainty 40 and best-body is full with certainty 60))

(rule (if main-component is-not fish and has-sauce is yes and sauce is tomato)
    (then best-color is red))
```

Backward-Chaining in FOL

- it works like it did in PropLog, but now we have to do unification when matching goals on the goal stack, and keep track of variable bindings
- this is the basis of how Prolog works (BC in FOL)

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 already applied)
    θ, the current substitution, initially the empty substitution $\{\,\}$
  local variables: $answers$, a set of substitutions, initially empty

  if $goals$ is empty **then return** $\{\theta\}$
  $q' \leftarrow$ SUBST(θ, FIRST($goals$))
  for each sentence r **in** KB
    where STANDARDIZE-APART(r) $= (p_1 \wedge \ldots \wedge p_n \Rightarrow q)$
    and $\theta' \leftarrow$ UNIFY(q, q') succeeds
   $new\_goals \leftarrow [p_1, \ldots, p_n | \text{REST}(goals)]$
   $answers \leftarrow$ FOL-BC-ASK(KB, new_goals, COMPOSE(θ', θ)) $\cup$ $answers$
  return $answers$

goal q' matches consequent of rule q, or goal matches a fact (where fact is like a rule with no antecedents, i.e. n=0

1. ∀X,Y,Z american(X)∧weapon(Y)∧hostile(Z)∧sells(X,Y,Z)→criminal(X)

2a. owns(nono,m1) // skolemized to make it definite-clause KB

2b. missile(m1)

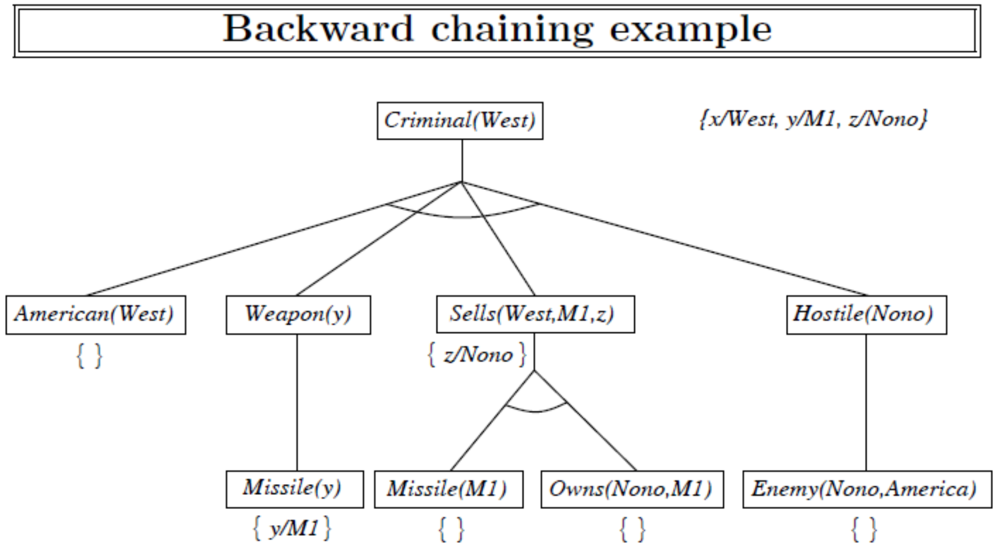
3. ∀C owns(nono,C)∧missile(C)→sells(west,C,nono)

4. ∀D missile(D)→weapon(D)

5. ∀E enemy(E,america)→hostile(E)

6. enemy(nono,america)

7. american(west)



(accumulated var bindings)

goal stack (left is top)	unifier	annotation
[criminal(west)]	θ={}	initialize with query
[american(west), weapon(Y), sells(west,Y,Z), hostile(Z)]	θ={X/west}	replace *criminal* with ants of rule 1.
[weapon(Y), sells(west,Y,Z), hostile(Z)]	θ={X/west}	pop *american* by fact 6
[missile(Y), sells(west,Y,Z), hostile(Z)]	θ={X/west, D/Y}	pop *weapon*, push *missile*, rule 4
[sells(west,m1,Z), hostile(Z)]	θ={X/west, D/Y, Y/m1}	pop *missile* by fact 2b
[owns(nono,m1),missle(m1),hostile(nono)]	θ={X/west, D/Y, Y/m1, C/m1, Z/nono}	match *sells* to conseq of rule 3
[missle(m1),hostile(nono)]	θ={X/west, D/Y, Y/m1, C/m1, Z/nono}	pop *owns* by fact 2a
[hostile(nono)]	θ={X/west, D/Y, Y/m1, C/m1, Z/nono}	pop *missile* by fact 2b
[enemy(nono,America)]	θ={X/west, D/Y, Y/m1, C/m1, Z/nono, E/nono}	match *hostile* to conseq of 5; replace with *enemy*
∅ (empty stack)	θ={X/west, D/Y, Y/m1, C/m1, Z/nono, E/nono}	pop enemy, since matches fact 6, leaving empty stack!

Example: Kinship KB (Simpsons characters)

female(lisa)
female(marge)
male(bart)
male(homer)
male(tod)
male(rod)
male(flanders)

parent(bart,homer)
parent(bart,marge)
parent(lisa,homer)
parent(lisa,marge)
parent(rod,flanders)
parent(tod,flanders)

$\forall$x,y parent(x,y)^male(y)→father(x,y)
$\forall$x,y parent(x,y)^female(x)→daughter(y,x)
$\forall$x,y [$\exists$z parent(x,z)^parent(y,z)→sibling(x,y)

What can we prove by **Backward Chaining**?

- remember to track variable bindings with unifiers!

query = father(lisa,homer)	query = sibling(rod,tod)
goal stack:	goal stack:
[father(lisa,homer)]	[sibling(rod,tod)]
// push antecedents	// push antecedents
[parent(lisa,homer),male(homer)] u={x/lisa,y/homer}	[parent(rod,z),parent(tod,z)] u={x/rod, y/tod}
// pop, since known fact	// pop, since unifies with parent(rod,flanders)
[male(homer)]	[parent(tod,flanders) u={x/rod, y/tod, z/flanders}
// pop, since known fact	// pop, since known fact
∅ empty stack	∅ empty stack

PROLOG

- PROLOG is an implementation of back-chaining in FOL.
- you can install PROLOG, and use it (by writing PROLOG programs) to build Expert Systems for all kinds of applications.