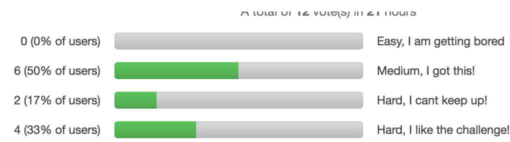
CS5100: Foundations of Artificial Intelligence

First Order Logic

Dr. Rutu Mulkar-Mehta
Lecture 4

Administrative

- EC1 (Constraint Satisfaction Problems) is graded and handed back to you
- Poll Results about difficulty of class (12 votes):



Administrative

- Project 1 – Grading in progress
  - Most of you have done well
  - Some have delayed submissions – 10% penalty per day
- Project 2 – Out today! Build your own Chatbot.
- Logic – some students expressed difficty in understanding the material. As a result, t oday's In Class Assignment – Postponed to Next week – Lets discuss more Logic today!

EC1 SOLUTIONS

1) 0
2) 3 * 3 * 2 * 1 * 1 * 1 * 1 = 18
3) 4 * 4 * 3 * 2* 2* 2* 2 = 768

Give the constraint on the rose and corpse flower explicitly.
r-c > 1 ^ (r-d>c-d or r > c)
C >= r + 2
List all pairs of variables which have a constraint other than A = B.
|r-d| > |c-d| ^ r < c, |r-c|>1, |v - r| > 1, |v-s| > 2, |v-t| > 1

What will the remaining domains be after arc consistency is enforced?

C = {3,4,5}
R={1,2,3}
S={1,2,4,5}
T={1,2,3,4,5}
V={1,4,5}

Which variable or variables would be assigned first according to MRV?

C,R,V have the least variables leftover;however, V reduces the most remaing

Assume that we assign T=3, enforce arc consistency.

C={4,5}
R={1,2}
S={1,2,5}
V={5}

List all solutions to this CSP when T=3

(R,S,T,C,V) (S,R,T,C,V)

BACK TO LECTURE 6

Last week

- Propositional Logic
- A statement that can have the value – True or False
  - I am happy.
  - Seattle is sunny.

Last week

- Propositions are connected using connectives

P	Q	¬P
True	True	False
True	False	False
False	True	True
False	False	True

Last Week

P	Q	P ∨ Q
True	True	True
True	False	True
False	True	True
False	False	False

P	Q	P ⇒ Q
True	True	True
True	False	False
False	True	True
False	False	True

P	Q	P ∧ Q
True	True	True
True	False	False
False	True	False
False	False	False

P	Q	P ⇔ Q
True	True	True
True	False	False
False	True	False
False	False	True

Exercise

- Consider the following statements:
  - p / sunny = It is sunny this afternoon
  - q / colder = it is colder than yesterday
  - r /swimming = We will go swimming
  - s / canoe = we will take a canoe trip
  - t / sunset = We will be home by sunset

It is not sunny this afternoon and it is colder than yesterday.

not sunny and colder

~p ^ q

Exercise

- Consider the following statements:
  - p / sunny = It is sunny this afternoon
  - q / colder = it is colder than yesterday
  - r /swimming = We will go swimming
  - s / canoe = we will take a canoe trip
  - t / sunset = We will be home by sunset
- **We will go swimming only if it is sunny.**

swimming → sunny

r → p

swimming	sunny	swim - sunny
True	True	True
True	False	False
False	True	True
False	False	True

Exercise

- Consider the following statements:
  - p / sunny = It is sunny this afternoon
  - q / colder = it is colder than yesterday
  - r /swimming = We will go swimming
  - s / canoe = we will take a canoe trip
  - t / sunset = We will be home by sunset
- **We always go swimming if it is sunny.**

sunny → swimming

p→ r

swimming	sunny	swim - sunny
True	True	True
True	False	True
False	True	False
False	False	True

Exercise

- Consider the following statements:
  - p / sunny = It is sunny this afternoon
  - q / colder = it is colder than yesterday
  - r /swimming = We will go swimming
  - s / canoe = we will take a canoe trip
  - t / sunset = We will be home by sunset

If we do not go swimming then we will take a canoe trip.

not swimming → canoe

Exercise

- Consider the following statements:
  - p / sunny = It is sunny this afternoon
  - q / colder = it is colder than yesterday
  - r /swimming = We will go swimming
  - s / canoe = we will take a canoe trip
  - t / sunset = We will be home by sunset

If we take a canoe trip, then we will be home by sunset.

canoe → sunset

Logical equivalence

- Two sentences are logically equivalent iff true in same models: α ≡ ß iff α ⊨ β and β ⊨ α

$$
\begin{array}{rcll}
(\alpha \wedge \beta) & \equiv & (\beta \wedge \alpha) & \text{commutativity of } \wedge \\
(\alpha \vee \beta) & \equiv & (\beta \vee \alpha) & \text{commutativity of } \vee \\
((\alpha \wedge \beta) \wedge \gamma) & \equiv & (\alpha \wedge (\beta \wedge \gamma)) & \text{associativity of } \wedge \\
((\alpha \vee \beta) \vee \gamma) & \equiv & (\alpha \vee (\beta \vee \gamma)) & \text{associativity of } \vee \\
\neg(\neg\alpha) & \equiv & \alpha & \text{double-negation elimination} \\
(\alpha \Rightarrow \beta) & \equiv & (\neg\beta \Rightarrow \neg\alpha) & \text{contraposition} \\
(\alpha \Rightarrow \beta) & \equiv & (\neg\alpha \vee \beta) & \text{implication elimination} \\
(\alpha \Leftrightarrow \beta) & \equiv & ((\alpha \Rightarrow \beta) \wedge (\beta \Rightarrow \alpha)) & \text{biconditional elimination} \\
\neg(\alpha \wedge \beta) & \equiv & (\neg\alpha \vee \neg\beta) & \text{de Morgan} \\
\neg(\alpha \vee \beta) & \equiv & (\neg\alpha \wedge \neg\beta) & \text{de Morgan} \\
(\alpha \wedge (\beta \vee \gamma)) & \equiv & ((\alpha \wedge \beta) \vee (\alpha \wedge \gamma)) & \text{distributivity of } \wedge \text{ over } \vee \\
(\alpha \vee (\beta \wedge \gamma)) & \equiv & ((\alpha \vee \beta) \wedge (\alpha \vee \gamma)) & \text{distributivity of } \vee \text{ over } \wedge
\end{array}
$$

Last week

- Entailment
- A |= B
- A entails B, B logically follows from A

Pros and cons of propositional logic

☺ Propositional logic allows partial/disjunctive/negated information

☺ Propositional logic is compositional:
  - meaning of $B_{1,1} \wedge P_{1,2}$ is derived from meaning of $B_{1,1}$ and of $P_{1,2}$

☺ Meaning in propositional logic is context-independent
  - (unlike natural language, where meaning depends on context)

☹ Propositional logic has very limited expressive power
  - (unlike natural language)
  - E.g., cannot say "pits cause breezes in adjacent squares“
    - except by writing one sentence for each square
    - E.g p11 -> p12 ^ p21 ^ …
    - p33 -> p32 ^ p23 ^ …

Outline

- First Order Logic (FOL)
- Syntax and semantics of FOL
- Using FOL
- Knowledge engineering in FOL
- Inference using FOL
- Prolog

First-order logic

- Propositional logic assumes the world contains facts – that are *true* or *false*
- First-order logic (like natural language) assumes the world contains
  - Objects: people, houses, numbers, colors, baseball games, wars, …
  - Relations: red, round, prime, brother of, bigger than, part of, comes between, …
  - Functions: father of, best friend, one more than, plus, …

First Order Logic Syntax

```
S := <Sentence> ;
<Sentence> := <AtomicSentence> |
              <Sentence> <Connective> <Sentence> |
              <Quantifier> <Variable>,... <Sentence> |
              "NOT" <Sentence>
<AtomicSentence> := <Predicate> "(" <Term>, ... ")" |
              <Term> "=" <Term>;
<Term> := <Function> "(" <Term>, ... ")" |<Constant> | <Variable>;
<Connective> := "AND" | "OR" | "IMPLIES" | "EQUIVALENT";
<Quantifier> := "EXISTS" | "FORALL" ;
<Constant> := "A" | "X1" | "John" | ... ; (uppercase)
<Variable> := "a" | "x" | "s" | ... ; (lowercase)
<Predicate> := "Before" | "HasColor" | "Raining" | ... ;
<Function> := "Mother" | "LeftLegOf" | ... ;
```

Constants, Functions, Predicates

- Constant symbols, which represent individuals in the world
  - Mary
  - 3
  - Green
- Function symbols, which map individuals to individuals
  - father-of(Mary) = John
  - color-of(Sky) = Blue
- Predicate symbols, which map individuals to truth values
  - greater(5,3) = true
  - green(Grass) = true
  - color(Grass, Green) = true

Variables, Connectives and Quantifiers

- Variable symbols
  - They can hold any value
  - E.g., x, y, foo
- Connectives
  - Same as in PL: not (¬), and (^), or (v), implies (→), if and only if (⇔)
- Quantifiers
  - Universal (Ax)
  - Existential (Ex)

Universal quantification

(∀x) P(x) means that P holds for all values of x in the domain associated with that variable

- e.g.
  Everyone at NEU is smart:
  ∀x At(x,NEU) ⇒ Smart(x)

  All dolphins are mammals
  (∀x) dolphin(x) ⇒ mammal(x)

Existential quantification

∃*x P(x)* means that P holds for some value of x in the domain associated with that variable

- Someone at NEU is smart:
  - ∃*x* At(x,NEU) ∧ Smart(x)
- Some Mammals Lay eggs:
  - ∃*x* mammal(x) ^ lays-eggs(x)
- Permits one to make a statement about some object without naming it

Sentences, Terms and Atoms

- A term (denoting a real-world individual) is a constant symbol, a variable symbol, or an n-place function of n terms.
  - x and f(x1, ..., xn) are terms, where each xi is a term.
  - A term with no variables is a ground term
- An atom (which has value true or false) is either
  - an n-place predicate of n terms, or,
  - ¬P, P V Q, P ^ Q, P =>Q, P ⇔ Q where P and Q are atoms
- A sentence is an atom, or, if P is a sentence and x is a variable,
  - then (Ax)P and (Ex)P are sentences
- A well-formed formula (wff) is a sentence containing no "free" variables. That is, all variables are "bound" by universal or existential quantifiers.
  - (Ax)P(x,y) has x bound as a universally quantified variable, but y is free.

Exercises

1. Every gardener likes the sun.
   (Ax) gardener(x) → likes(x,Sun)
2. You can fool some of the people all of the time.
   (Ex)(At) (person(x) ^ time(t)) → can-fool(x,t)
3. You can fool all of the people some of the time.
   (Ax)(Et) (person(x) ^ time(t) → can-fool(x,t)
4. All purple mushrooms are poisonous.
   (Ax) (mushroom(x) ^ purple(x)) → poisonous(x)
5. No purple mushroom is poisonous.
   ¬(Ex) purple(x) ^ mushroom(x) ^ poisonous(x)
   (Ax) (mushroom(x) ^ purple(x)) → ¬poisonous(x)
6. There are exactly two purple mushrooms.
   (Ex)(Ey) mushroom(x) ^ purple(x) ^ mushroom(y) ^ purple(y) ^ ¬(x=y) ^ (A z) (mushroom(z) ^ purple(z)) → ((x=z) V (y=z))
7. Clinton is not tall.
   ¬tall(Clinton)

Working with Quantifiers

- Universal quantifiers are often used with "implies" to form "rules":
  (Ax) student(x) → smart(x) means "All students are smart"
- Universal quantification is rarely used to make blanket statements about every individual in the world:
  (Ax)student(x) ^ smart(x) means
  "Everyone in the world is a student and is smart"
- Existential quantifiers are usually used with "and" to specify a list of properties about an individual:
  (Ex) student(x) ^ smart(x) means "There is a student who is smart"
- A common mistake is to represent this English sentence as the FOL sentence:
  (Ex) student(x) → smart(x)
  But what happens when there is a person who is not a student?

Quantifier Scope

- Switching the order of universal quantifiers does not change the meaning:
  (Ax)(Ay)P(x,y) → (Ay)(Ax) P(x,y)
- Similarly, you can switch the order of existential quantifiers:
  (Ex)(Ey)P(x,y) → (Ey)(Ex) P(x,y)
- Switching the order of universals and existentials does change meaning:
  (Ax)(Ey) likes(x,y)
  Everyone likes someone
  (Ey)(Ax) likes(x,y)
  Someone is liked by everyone

Connections between For-All and There-Exists

- We can relate sentences involving A and E using De Morgan's laws:

(Ax) ¬P(x) → ¬ (Ex) P(x)
¬(Ax) P(x) → (Ex) ¬P(x)
(Ax) P(x) → ¬ (Ex) ¬P(x)
(Ex) P(x) → ¬ (Ax) ¬P(x)

Quantified Inference Rules

- Universal instantiation
  Ax P(x) : P(A)
- Universal generalization
  Ax P(x) : P(A) ^ P(B) …
- Existential instantiation
  Ex P(x) : P(F) (skolem constant F)
- Existential generalization
  Ex P(x) : P(A)

Notations

- Different symbols for and, or, not, implies, ...
- – A E => ⇔ ^ V ¬
- – p v (q ^ r)
- – p + (q * r)
- Prolog
  cat(X) :- furry(X), meows (X), has(X, claws)
- Lispy notations

```
(forall ?x (implies (and (furry ?x)
                         (meows ?x)
                         (has ?x claws))
                    (cat ?x)))
```

Summary so far

- First-order logic:
  – objects and relations are semantic primitives
  – syntax: constants, functions, predicates, equality, quantifiers
- Increased expressive power: sufficient to define wumpus world

INFERENCE IN FIRST-ORDER LOGIC

Universal instantiation (UI)

- Every instantiation of a universally quantified sentence is entailed by it:

$$\frac{\forall v\ \alpha}{\text{Subst}(\{v/g\}, \alpha)}$$

for any variable *v* and ground term *g*

- E.g., $\forall x\ King(x) \wedge Greedy(x) \Rightarrow Evil(x)$ yields:

King(John) ∧ *Greedy(John)* ⇒ *Evil(John)*
King(Richard) ∧ *Greedy(Richard)* ⇒ *Evil(Richard)*
King(Father(John)) ∧ *Greedy(Father(John))* ⇒ *Evil(Father(John))*
.
.
.

Existential instantiation (EI)

- For any sentence α, variable *v*, and constant symbol *k* that does not appear elsewhere in the knowledge base:

$$\frac{\exists v\ \alpha}{\text{Subst}(\{v/k\}, \alpha)}$$

- E.g., $\exists x\ Crown(x) \wedge OnHead(x,John)$ yields:

$$Crown(C_1) \wedge OnHead(C_1,John)$$

provided C_1 is a new constant symbol, called a Skolem constant

Reduction to propositional inference

Suppose the KB contains just the following:

∀x King(x) ∧ Greedy(x) ⇒ Evil(x)
King(John)
Greedy(John)
Brother(Richard,John)

- Instantiating the universal sentence in all possible ways, we have:
  King(John) ∧ Greedy(John) ⇒ Evil(John)
  King(Richard) ∧ Greedy(Richard) ⇒ Evil(Richard)
  King(John)
  Greedy(John)
  Brother(Richard,John)

- The new KB is propositionalized: proposition symbols are

King(John), Greedy(John), Evil(John), King(Richard), etc.

Reduction contd.

- Every FOL KB can be propositionalized so as to preserve entailment
- (A ground sentence is entailed by new KB iff entailed by original KB)
- Idea: propositionalize KB and query, apply resolution, return result

Problems with propositionalization

- Propositionalization seems to generate lots of irrelevant sentences.
- E.g., from:

∀x King(x) ∧ Greedy(x) ⇒ Evil(x)
King(John)
∀y Greedy(y)
Brother(Richard,John)

- it seems obvious that *Evil*(*John*), but propositionalization produces lots of facts such as *Greedy*(*Richard*) that are irrelevant
- With *p* *k*-ary predicates and *n* constants, there are $p \cdot n^k$ instantiations.

Unification

Unification – takes two similar sentences and computes the substitution that makes them look the same

- Unify(α,β) = θ if subs(α,θ) = subs(β,θ)

p	q	θ
Knows(John,x)	Knows(John,Jane)	
Knows(John,x)	Knows(y,OJ)	
Knows(John,x)	Knows(y,Mother(y))	
Knows(John,x)	Knows(x,OJ)	

Unification

Unification – takes two similar sentences and computes the substitution that makes them look the same

- Unify(α,β) = θ if subs(α,θ) = subs(β,θ)

p	q	θ
Knows(John,x)	Knows(John,Jane)	{x/Jane}}
Knows(John,x)	Knows(y,OJ)	
Knows(John,x)	Knows(y,Mother(y))	
Knows(John,x)	Knows(x,OJ)	

Unification

Unification – takes two similar sentences and computes the substitution that makes them look the same

- Unify(α,β) = θ if subs(α,θ) = subs(β,θ)

p	q	θ
Knows(John,x)	Knows(John,Jane)	{x/Jane}}
Knows(John,x)	Knows(y,OJ)	{x/OJ,y/John}}
Knows(John,x)	Knows(y,Mother(y))	
Knows(John,x)	Knows(x,OJ)	

Unification

Unification – takes two similar sentences and computes the substitution that makes them look the same

- Unify(α,β) = θ if subs(α,θ) = subs(β,θ)

p	q	θ
Knows(John,x)	Knows(John,Jane)	{x/Jane}}
Knows(John,x)	Knows(y,OJ)	{x/OJ,y/John}}
Knows(John,x)	Knows(y,Mother(y))	{y/John,x/Mother(John)}}
Knows(John,x)	Knows(x,OJ)	

Unification

Unification – takes two similar sentences and computes the substitution that makes them look the same

- Unify(α,β) = θ if subs(α,θ) = subs(β,θ)

p	q	θ
Knows(John,x)	Knows(John,Jane)	{x/Jane}}
Knows(John,x)	Knows(y,OJ)	{x/OJ,y/John}}
Knows(John,x)	Knows(y,Mother(y))	{y/John,x/Mother(John)}}
Knows(John,x)	Knows(x,OJ)	{fail}

Unification

- To unify *Knows(John,x)* and *Knows(y,z)*,

  θ = {y/John, x/z } or θ = {y/John, x/John, z/John}

- The first unifier is more general than the second.
- There is a single most general unifier (MGU) that is unique up to renaming of variables.

  MGU = { y/John, x/z }

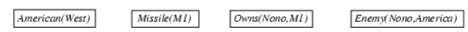
Example knowledge base

- 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

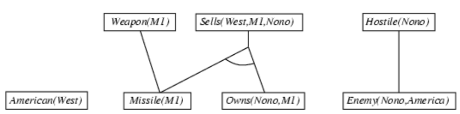
Example knowledge base contd.

... it is a crime for an American to sell weapons to hostile nations:
American(x) ∧ Weapon(y) ∧ Sells(x,y,z) ∧ Hostile(z) ⇒ Criminal(x)
Nono … has some missiles, i.e., ∃x Owns(Nono,x) ∧ Missile(x):
Owns(Nono,M_1) and Missile(M_1)
… all of its missiles were sold to it by Colonel West
Missile(x) ∧ Owns(Nono,x) ⇒ Sells(West,x,Nono)
Missiles are weapons:
Missile(x) ⇒ Weapon(x)
An enemy of America counts as "hostile“:
Enemy(x,America) ⇒ Hostile(x)
West, who is American …
American(West)
The country Nono, an enemy of America …
Enemy(Nono,America)

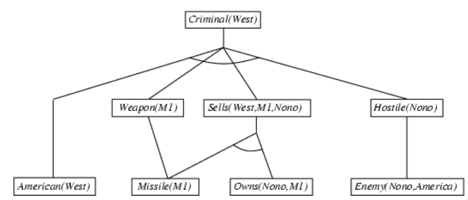
Forward chaining proof



Forward chaining proof



Forward chaining proof



Properties of forward chaining

- Sound and complete for first-order definite clauses
- Datalog = first-order definite clauses + no functions
- FC terminates for Datalog in finite number of iterations
- May not terminate in general if α is not entailed
- This is unavoidable: entailment with definite clauses is semidecidable

Efficiency of forward chaining

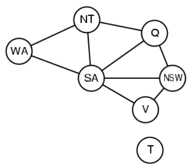
Incremental forward chaining: no need to match a rule on iteration *k* if a premise wasn't added on iteration *k-1*
⇒ match each rule whose premise contains a newly added positive literal

Matching itself can be expensive:
Database indexing allows O(1) retrieval of known facts

– e.g., query *Missile(x)* retrieves *Missile(M_1)*

Forward chaining is widely used in deductive databases

Hard matching example

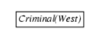


Diff(wa,nt) ∧ *Diff(wa,sa)* ∧ *Diff(nt,q)* ∧
Diff(nt,sa) ∧ *Diff(q,nsw)* ∧ *Diff(q,sa)* ∧
Diff(nsw,v) ∧ *Diff(nsw,sa)* ∧ *Diff(v,sa)* ⇒
Colorable()

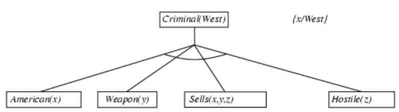
Diff(Red,Blue) *Diff (Red,Green)*
Diff(Green,Red) *Diff(Green,Blue)*
Diff(Blue,Red) *Diff(Blue,Green)*

- *Colorable*() is inferred iff the CSP has a solution
- CSPs include 3SAT as a special case, hence matching is NP-hard

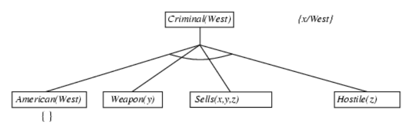
Backward chaining example



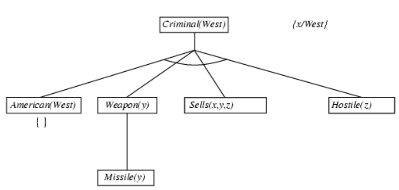
Backward chaining example



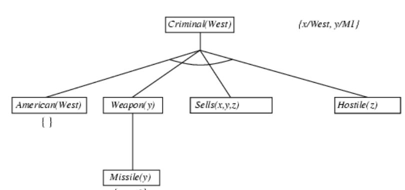
Backward chaining example



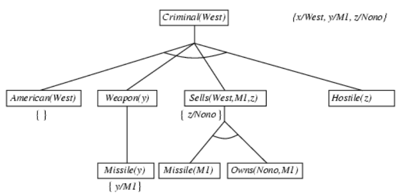
Backward chaining example



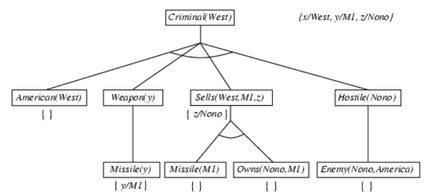
Backward chaining example



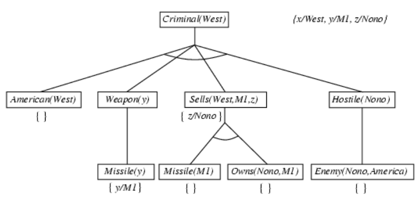
Backward chaining example



Backward chaining example



Backward chaining example



Properties of backward chaining

- Depth-first recursive proof search: space is linear in size of proof
- Incomplete due to infinite loops
  - $\Rightarrow$ fix by checking current goal against every goal on stack
- Inefficient due to repeated subgoals (both success and failure)
  - $\Rightarrow$ fix using caching of previous results (extra space)
- Widely used for logic programming

Logic programming: Prolog

- Algorithm = Logic + Control (Programming Logic)
- Prolog Programs are a set of definite clauses that are written in a notation that is somewhat different from standard first-order logic
  - Variables – Uppercase
  - Constants – Lowercase
  - Commas separate conjuncts in a clause
  - % is a comment in Prolog
  - Clauses are written backwards from what we are used to

American(x) ∧ Weapon(y) ∧ Sells(x,y,z) ∧ Hostile(z) ⇒ Criminal(x)

```
criminal(X) :- american(X), weapon(Y), sells(X,Y,Z), hostile(Z).
```

DCG: Definite Clause Grammar

```
sentence --> noun_phrase, verb_phrase.
noun_phrase --> det, noun.
verb_phrase --> verb, noun_phrase.
det --> [the].
det --> [a].
noun --> [cat].
noun --> [bat].
verb --> [eats].
```

Prolog

f(a).
f(b).
g(a).
g(b).
h(b).
k(X):- f(X), g(X), h(X).

k(X). (which constant satisfies it?)

Prolog

- Appending two lists to produce a third:

```
append([],Y,Y).
append([X|L],Y,[X|Z]) :- append(L,Y,Z).
```

- query: `append(A,B,[1,2]) ?`
- answers: `A=[]    B=[1,2]`

  `A=[1]   B=[2]`

  `A=[1,2] B=[]`

Criminal Scenario in Prolog

```
% it is a crime to sell weapons to hostile nations:
criminal(X):-american(X),weapon(Y),sells(X,Y,Z),hostile(Z).
% Nono ... has some missiles,
owns(nono,m1).
missile(m1).
% all of its missiles were sold to it by Colonel West
sells(west,X,nono) :- missile(X), owns(nono,X).
% Missiles are weapons
weapon(X) :- missile(X).
% An enemy of America counts as ``hostile'':
hostile(X) :- enemy(X,america).
% The country Nono, an enemy of America ...
enemy(nono,america).
% West, who is American ...
american(west).
```