Today

See Russell and Norvig, chapter 7

- Propositional Logic ctd
- Inference algorithms

Reminder

- **Syntax**: proposition symbols, joined with $\neg, \wedge, \vee, \Rightarrow, \Leftrightarrow$.
- **Semantics**: truth values, logical consequence $KB \models F$
- special formulas:

valid:	true in **all** interpretations
satisfiable:	true in **some** interpretations
contradictory:	true in **no** interpretations

Inference by enumeration

Depth-first enumeration of all models is sound and complete

function $\mathrm{TT\text{-}E{\small NTAILS}?}(KB, \alpha)$ **returns** *true* or *false*
  symbols $\leftarrow$ a list of the proposition symbols in KB and α
  return $\mathrm{TT\text{-}C{\small HECK}\text{-}A{\small LL}}(KB, \alpha, symbols, [\,])$

function $\mathrm{TT\text{-}C{\small HECK}\text{-}A{\small LL}}(KB, \alpha, symbols, model)$ **returns** *true* or *false*
  if $\mathrm{E{\small MPTY}?}(symbols)$ **then**
    if $\mathrm{PL\text{-}T{\small RUE}?}(KB, model)$ **then return** $\mathrm{PL\text{-}T{\small RUE}?}(\alpha, model)$
    else return *true*
  else do
    $P \leftarrow \mathrm{F{\small IRST}}(symbols)$; $rest \leftarrow \mathrm{R{\small EST}}(symbols)$
    return $\mathrm{TT\text{-}C{\small HECK}\text{-}A{\small LL}}(KB, \alpha, rest, \mathrm{E{\small XTEND}}(P, true, model))$ **and**
      $\mathrm{TT\text{-}C{\small HECK}\text{-}A{\small LL}}(KB, \alpha, rest, \mathrm{E{\small XTEND}}(P, false, model))$

Logical equivalence

Two sentences are logically equivalent iff true in same models:
$\alpha \equiv \beta$ if and only if $\alpha \models \beta$ and $\beta \models \alpha$

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

Proof methods

Proof methods divide into (roughly) two kinds:

Application of inference rules
- Legitimate (sound) generation of new sentences from old
- Proof = a sequence of inference rule applications
  Can use inference rules as operators in a standard search alg.

Model checking
truth table enumeration (always exponential in n)
improved backtracking, heuristic search in model space
(sound but incomplete)
e.g., min-conflicts-like hill-climbing algorithms

Forward and backward chaining

Horn Form (restricted)
KB = *conjunction* of *Horn clauses* also called definite clauses
Horn clause =
- ♢ proposition symbol; or
- ♢ (conjunction of symbols) $\Rightarrow$ symbol

E.g., $C \wedge (B \Rightarrow A) \wedge (C \wedge D \Rightarrow B)$

Modus Ponens (for Horn Form): complete for Horn KBs

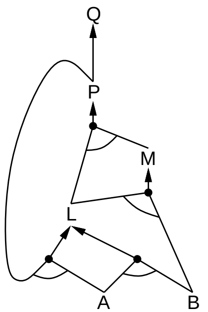
$$\frac{\alpha_1, \ldots, \alpha_n, \qquad \alpha_1 \wedge \cdots \wedge \alpha_n \Rightarrow \beta}{\beta}$$

Can be used with forward chaining or backward chaining.
These algorithms are very natural and run in *linear* time

Forward chaining

Idea: fire any rule whose premises are satisfied in the KB,
add its conclusion to the KB, until query is found

$P \Rightarrow Q$
$L \wedge M \Rightarrow P$
$B \wedge L \Rightarrow M$
$A \wedge P \Rightarrow L$
$A \wedge B \Rightarrow L$
A
B

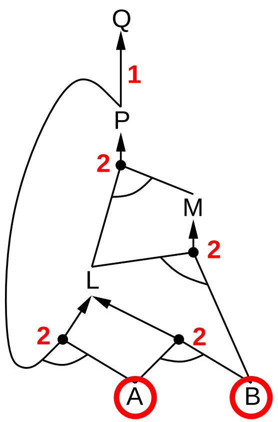


Forward chaining algorithm

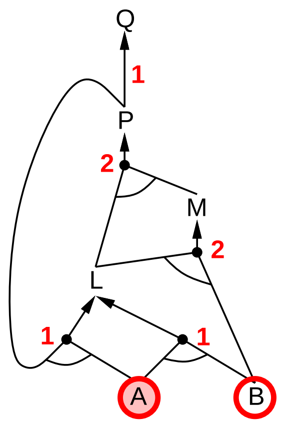
```
function PL-FC-ENTAILS?(KB, q) returns true or false
   local variables: count, table, indexed by clause, initially no. of premises
                    inferred, table, indexed by symbol, each entry initially false
                    agenda, list of symbols, initially symbols known to be true

   while agenda is not empty do
      p ← POP(agenda)
      unless inferred[p] do
         inferred[p] ← true
         for each Horn clause c in whose premise p appears do
            decrement count[c]
            if count[c] = 0 then do
               if HEAD[c] = q then return true
               PUSH(HEAD[c], agenda)
   return false
```

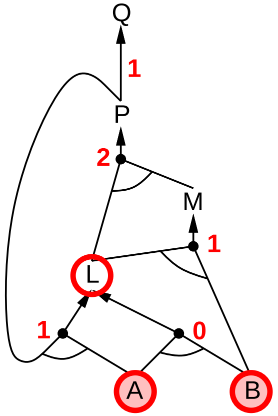
Forward chaining example



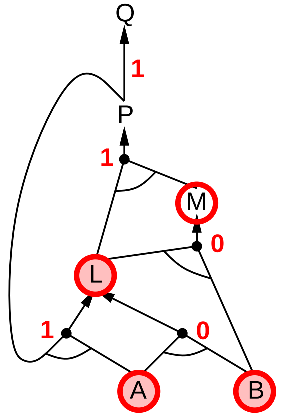
Forward chaining example



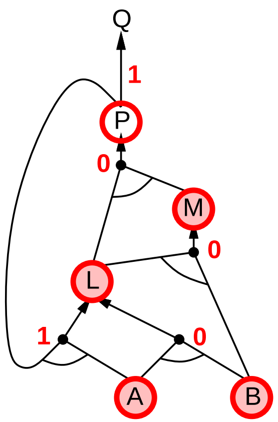
Forward chaining example



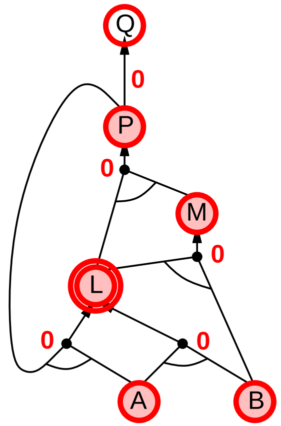
Forward chaining example



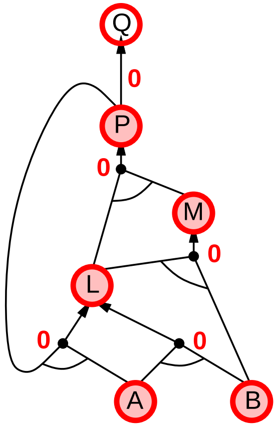
Forward chaining example



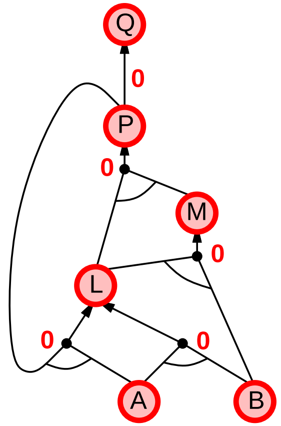
Forward chaining example



Forward chaining example



Forward chaining example



Proof of completeness

FC derives every atomic sentence that is entailed by KB

1. FC reaches a fixed point where no new atomic sentences are derived.
2. Consider the final state as a model m, assigning true/false to symbols.
3. Every clause in the original KB is true in m
   Proof: Suppose a clause $a_1 \land \ldots \land a_k \Rightarrow b$ is false in m.
   Then $a_1 \land \ldots \land a_k$ is true in m and b is false in m.
   Therefore the algorithm has not reached a fixed point!
4. Hence m is a model of KB.
5. If $KB \models q$, then q is true in *every* model of KB, including m.

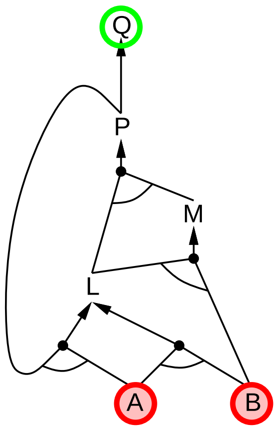
Backward chaining

Idea: work backwards from the query q:
 to prove q by BC,
  check if q is known already, or
  prove by BC all premises of some rule concluding q

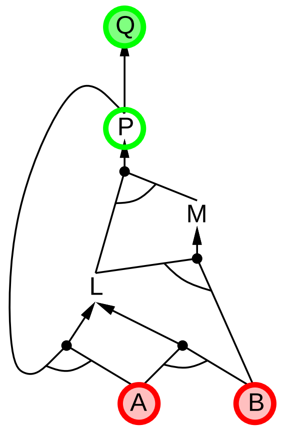
Avoid loops: check if new subgoal is already on the goal stack

Avoid repeated work: check if new subgoal
 1) has already been proved true, or
 2) has already failed

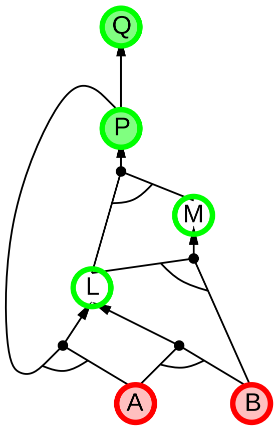
Backward chaining example



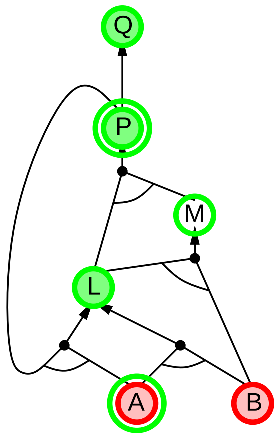
Backward chaining example



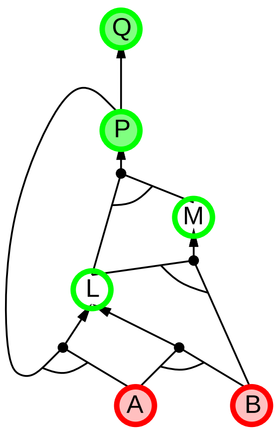
Backward chaining example



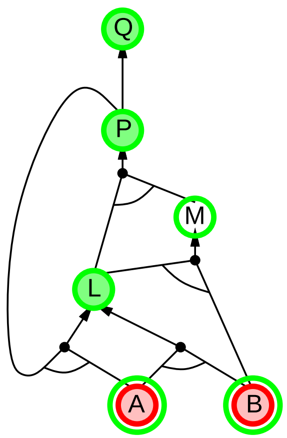
Backward chaining example



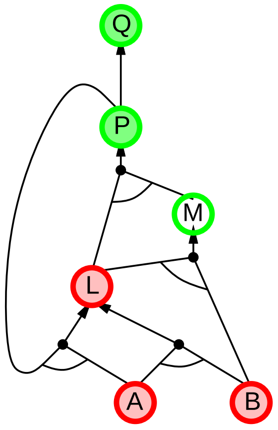
Backward chaining example



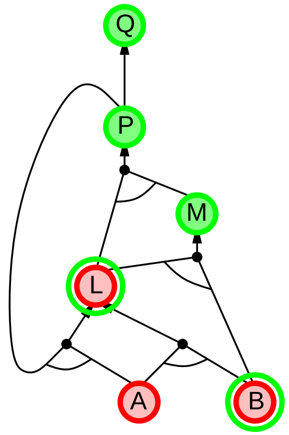
Backward chaining example



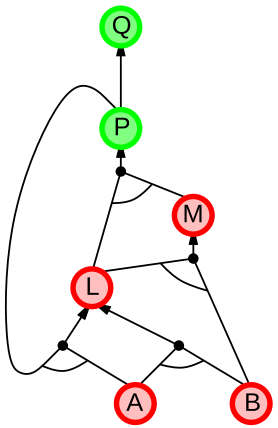
Backward chaining example



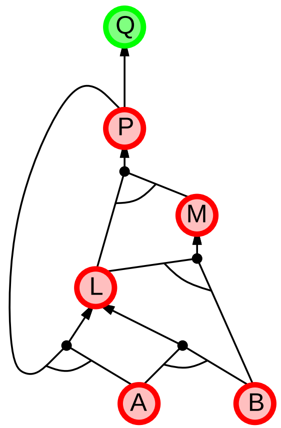
Backward chaining example



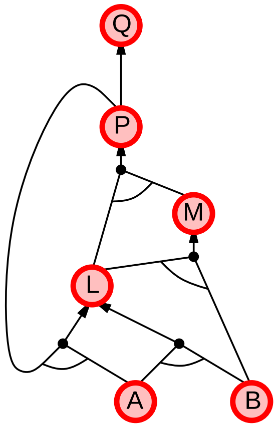
Backward chaining example



Backward chaining example



Backward chaining example



Forward vs. backward chaining

FC is data-driven, cf. automatic, unconscious processing,
e.g., object recognition, routine decisions

May do lots of work that is irrelevant to the goal

BC is goal-driven, appropriate for problem-solving,
e.g., Where are my keys? How do I get into a PhD programme?

Complexity of BC can be *much less* than linear in size of KB

Pros and cons of propositional logic

Propositional logic is *declarative*:
pieces of syntax correspond to facts

Propositional logic allows partial/disjunctive/negated information
(unlike most data structures and databases)

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

First-order logic

Whereas propositional logic assumes world contains *facts*,
first-order logic (like natural language) assumes the world contains

- Objects: people, houses, numbers, theories, Jacques Chirac, colours, soduko games, wars, centuries . . .
- Relations: red, round, bogus, prime, multistoried, brother of, bigger than, inside, part of, has colour, occurred after, owns, comes between, . . .
- Functions: father of, best friend, second innings of, one more than, beginning of . . .

Syntax of FOL: Basic elements

Constants	$KingJohn,\ 2,\ UN, \ldots$
Predicates	$Brother,\ >, \ldots$
Functions	$Sqrt,\ LeftLegOf, \ldots$
Variables	$x,\ y,\ a,\ b, \ldots$
Connectives	$\wedge\ \vee\ \neg\ \Rightarrow\ \ \Leftrightarrow$
Equality	$=$
Quantifiers	$\forall\ \exists$

Atomic sentences

Atomic sentence $=$ $predicate(term_1, \ldots, term_n)$
or $term_1 = term_2$

Term $=$ $function(term_1, \ldots, term_n)$
or $constant$ or $variable$

E.g., $Brother(KingJohn, RichardTheLionheart)$

$$>(Length(LeftLegOf(Richard)), Length(LeftLegOf(KingJohn)))$$

For human consumption, often write $>(X, Y)$ as $X > Y$.

Complex sentences

Complex sentences are made from atomic sentences using connectives

$$\neg S, \quad S_1 \wedge S_2, \quad S_1 \vee S_2, \quad S_1 \Rightarrow S_2, \quad S_1 \Leftrightarrow S_2$$

E.g. $Sibling(KingJohn, Richard) \Rightarrow Sibling(Richard, KingJohn)$
$>(1,2) \ \vee\ \leq(1,2)$
$>(1,2) \ \wedge\ (\neg >(1,2))$

Universal quantification

$\forall \langle variables \rangle\ \langle sentence \rangle$

Everyone at Berkeley is smart:
$\forall x\ At(x, Berkeley) \Rightarrow Smart(x)$

$\forall x\ P$ is true in a model m iff P is true with x being each possible object in the model

Roughly speaking, equivalent to the conjunction of instantiations of P

$$\begin{array}{ll} & At(KingJohn, Berkeley) \Rightarrow Smart(KingJohn) \\ \wedge & At(Richard, Berkeley) \Rightarrow Smart(Richard) \\ \wedge & At(Berkeley, Berkeley) \Rightarrow Smart(Berkeley) \\ \wedge & \ldots \end{array}$$

A common mistake to avoid

Typically, $\Rightarrow$ is the main connective with $\forall$

Common mistake: using $\land$ as the main connective with $\forall$:

$$\forall x \;\; At(x, Berkeley) \land Smart(x)$$

means "Everyone is at Berkeley and everyone is smart"

Existential quantification

$\exists \langle variables \rangle \;\; \langle sentence \rangle$

Someone at Stanford is smart:
$\exists x \;\; At(x, Stanford) \land Smart(x)$

$\exists x \;\; P$ is true in a model m iff P is true with x being *some* possible object in the model

Roughly speaking, equivalent to the disjunction of instantiations of P

$$\begin{array}{ll} & At(KingJohn, Stanford) \land Smart(KingJohn) \\ \lor & At(Richard, Stanford) \land Smart(Richard) \\ \lor & At(Stanford, Stanford) \land Smart(Stanford) \\ \lor & \ldots \end{array}$$

Another common mistake to avoid

Typically, $\land$ is the main connective with $\exists$

Common mistake: using $\Rightarrow$ as the main connective with $\exists$:

$$\exists x \;\; At(x, Stanford) \;\Rightarrow\; Smart(x)$$

is true if there is anyone who is not at Stanford!

Summary

Propositional & first-order logic:
- syntax, semantics, entailment, . . .
- Forward, backward chaining are linear-time, complete for Horn clauses

First-order logic:
- objects and relations are semantic primitives
- syntax: constants, functions, predicates, equality, quantifiers
- much increased expressive power