Discussion 6

May 16, 2007
Matthew Tong
• How was the midterm?
• How’s the project?
• Questions?
• How’s life?
• Hopefully all reading the board?
• Getting emails from me/Gary?
  – Anyone with a preferred email?
Why logic?

• I think it’s possible to forget that beneath all these formalisms and toy problems lies one of the more powerful tools in computer science
NL vs. Logic: Ambiguity and Precision

**NL:**

- x is at the bank.
  - river bank?
  - financial institution?

**Logic:**

- x is running-InMotion® x is changing location
- x is running-DeviceOperating® x is operating
- x is running-AsCandidate® x is a candidate

**Reasoning:** Figuring out what must be true, given what is known. Requires precision of meaning.
NL vs. Logic: Calculus of Meaning

Logic: Well-understood operators enable reasoning:

Logical constants: not, and, or, all, some

Not (All men are taller than all women).

All men are taller than 12”.

Some women are taller than 12”.

Not (All A are F than all B).

All A are F than x.

Some B are F than x.
Why logic?

• In programming, you always need to formalize things
  – For example, for non overlapping rectangles:
    • $X_1 > 0, Y_1 > 0, X_2 > 0, Y_2 > 0$
    • $X_1 + \text{width} < \text{width limit}$, …
    • …
  – Logic provides a formal syntax for expressing semantic content
Why logic?

- Computers can manipulate these formalized systems to:
  - Derive facts about the world or domain of interest
  - Plan or act rationally
  - Or even just basically work with entities of interest

- Not necessarily automated reasoning (e.g. resolution), but more sophisticated tests (e.g. expert systems)
## Logic programming

Sound bite: computation as inference on logical KBs

<table>
<thead>
<tr>
<th>Logic programming</th>
<th>Ordinary programming</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Identify problem</td>
<td>Identify problem</td>
</tr>
<tr>
<td>2. Assemble information</td>
<td>Assemble information</td>
</tr>
<tr>
<td>3. Tea break</td>
<td>Figure out solution</td>
</tr>
<tr>
<td>4. Encode information in KB</td>
<td>Program solution</td>
</tr>
<tr>
<td>5. Encode problem instance as facts</td>
<td>Encode problem instance as data</td>
</tr>
<tr>
<td>6. Ask queries</td>
<td>Apply program to data</td>
</tr>
<tr>
<td>7. Find false facts</td>
<td>Debug procedural errors</td>
</tr>
</tbody>
</table>

Should be easier to debug $\text{Capital(\text{NewYork, US})}$ than $x := x + 2$!
## Logics in general

<table>
<thead>
<tr>
<th>Language</th>
<th>Ontological Commitment</th>
<th>Epistemological Commitment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Propositional logic</td>
<td>facts</td>
<td>true/false/unknown</td>
</tr>
<tr>
<td>First-order logic</td>
<td>facts, objects, relations</td>
<td>true/false/unknown</td>
</tr>
<tr>
<td>Temporal logic</td>
<td>facts, objects, relations, times</td>
<td>true/false/unknown</td>
</tr>
<tr>
<td>Probability theory</td>
<td>facts</td>
<td>degree of belief</td>
</tr>
<tr>
<td>Fuzzy logic</td>
<td>facts + degree of truth</td>
<td>known interval value</td>
</tr>
</tbody>
</table>
Knowledge bases

Knowledge base = set of sentences in a formal language

Declarative approach to building an agent (or other system):
Tell it what it needs to know

Then it can Ask itself what to do—answers should follow from the KB

Agents can be viewed at the knowledge level
i.e., what they know, regardless of how implemented

Or at the implementation level
i.e., data structures in KB and algorithms that manipulate them
Logic in general

Logics are formal languages for representing information such that conclusions can be drawn.

Syntax defines the sentences in the language.

Semantics define the “meaning” of sentences; i.e., define truth of a sentence in a world.

E.g., the language of arithmetic:

\( x + 2 \geq y \) is a sentence; \( x^2 + y > \) is not a sentence.

\( x + 2 \geq y \) is true iff the number \( x + 2 \) is no less than the number \( y \).

\( x + 2 \geq y \) is true in a world where \( x = 7, \ y = 1 \).

\( x + 2 \geq y \) is false in a world where \( x = 0, \ y = 6 \).
Entailment means that one thing follows from another:

$$KB \models \alpha$$

Knowledge base $KB$ entails sentence $\alpha$
if and only if
$\alpha$ is true in all worlds where $KB$ is true

E.g., the KB containing “the Giants won” and “the Reds won”
entails “Either the Giants won or the Reds won”

E.g., $x + y = 4$ entails $4 = x + y$

Entailment is a relationship between sentences (i.e., syntax)
that is based on semantics

Note: brains process syntax (of some sort)
Models

Logicians typically think in terms of models, which are formally structured worlds with respect to which truth can be evaluated.

We say \( m \) is a model of a sentence \( \alpha \) if \( \alpha \) is true in \( m \).

\( M(\alpha) \) is the set of all models of \( \alpha \).

Then \( KB \models \alpha \) if and only if \( M(KB) \subseteq M(\alpha) \).

E.g. \( KB = \) Giants won and Reds won

\( \alpha = \) Giants won
Inference

KB ⊢_i α = sentence α can be derived from KB by procedure i

Consequences of KB are a haystack; α is a needle.
Entailment = needle in haystack; inference = finding it

Soundness: i is sound if
whenever KB ⊢_i α, it is also true that KB ⊨ α

Completeness: i is complete if
whenever KB ⊨ α, it is also true that KB ⊢_i α

Preview: we will define a logic (first-order logic) which is expressive enough to say almost anything of interest, and for which there exists a sound and complete inference procedure.

That is, the procedure will answer any question whose answer follows from what is known by the KB.
Propositional logic: Syntax

Propositional logic is the simplest logic—illustrates basic ideas

The proposition symbols $P_1, P_2$ etc are sentences

---

Propositional logic: Semantics

Each model specifies true/false for each proposition symbol

E.g. $P_{1,2} \quad P_{2,2} \quad P_{3,1}$

\[
\begin{array}{ccc}
\text{true} & \text{true} & \text{false}
\end{array}
\]

(With these symbols, 8 possible models, can be enumerated automatically.)
Truth tables for inference

<table>
<thead>
<tr>
<th>$B_{1,1}$</th>
<th>$B_{2,1}$</th>
<th>$P_{1,1}$</th>
<th>$P_{1,2}$</th>
<th>$P_{2,1}$</th>
<th>$P_{2,2}$</th>
<th>$P_{3,1}$</th>
<th>$R_1$</th>
<th>$R_2$</th>
<th>$R_3$</th>
<th>$R_4$</th>
<th>$R_5$</th>
<th>KB</th>
</tr>
</thead>
<tbody>
<tr>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>false</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>false</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>false</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>false</td>
<td>false</td>
<td>false</td>
<td>true</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>false</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>false</td>
<td>true</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>false</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>false</td>
<td>true</td>
<td>true</td>
<td>true</td>
<td>false</td>
<td>false</td>
<td>false</td>
</tr>
</tbody>
</table>

Enumerate rows (different assignments to symbols),
if KB is true in row, check that $\alpha$ is too
Logical equivalence

Two sentences are logically equivalent iff true in same models:

\( \alpha \equiv \beta \) if and only if \( \alpha \models \beta \) and \( \beta \models \alpha \)

\[
\begin{align*}
(\alpha \land \beta) & \equiv (\beta \land \alpha) \quad \text{commutativity of } \land \\
(\alpha \lor \beta) & \equiv (\beta \lor \alpha) \quad \text{commutativity of } \lor \\
((\alpha \land \beta) \land \gamma) & \equiv (\alpha \land (\beta \land \gamma)) \quad \text{associativity of } \land \\
((\alpha \lor \beta) \lor \gamma) & \equiv (\alpha \lor (\beta \lor \gamma)) \quad \text{associativity of } \lor \\
\neg(\neg \alpha) & \equiv \alpha \quad \text{double-negation elimination} \\
(\alpha \Rightarrow \beta) & \equiv (\neg \beta \Rightarrow \neg \alpha) \quad \text{contraposition} \\
(\alpha \Rightarrow \beta) & \equiv (\neg \alpha \lor \beta) \quad \text{implication elimination} \\
(\alpha \leftrightarrow \beta) & \equiv ((\alpha \Rightarrow \beta) \land (\beta \Rightarrow \alpha)) \quad \text{biconditional elimination} \\
\neg(\alpha \land \beta) & \equiv (\neg \alpha \lor \neg \beta) \quad \text{De Morgan} \\
\neg(\alpha \lor \beta) & \equiv (\neg \alpha \land \neg \beta) \quad \text{De Morgan} \\
(\alpha \land (\beta \lor \gamma)) & \equiv ((\alpha \land \beta) \lor (\alpha \land \gamma)) \quad \text{distributivity of } \land \text{ over } \lor \\
(\alpha \lor (\beta \land \gamma)) & \equiv ((\alpha \lor \beta) \land (\alpha \lor \gamma)) \quad \text{distributivity of } \lor \text{ over } \land
\end{align*}
\]
Validity and satisfiability

A sentence is valid if it is true in all models,
e.g., True, $A \lor \neg A$, $A \Rightarrow A$, $(A \land (A \Rightarrow B)) \Rightarrow B$

Validity is connected to inference via the Deduction Theorem:
$KB \models \alpha$ if and only if $(KB \Rightarrow \alpha)$ is valid

A sentence is satisfiable if it is true in some model
e.g., $A \lor B$, $C$

A sentence is unsatisfiable if it is true in no models
e.g., $A \land \neg A$

Satisfiability is connected to inference via the following:
$KB \models \alpha$ if and only if $(KB \land \neg \alpha)$ is unsatisfiable
i.e., prove $\alpha$ by reductio ad absurdum
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.
- Typically require translation of sentences into a normal form

Model checking
  
  truth table enumeration (always exponential in $n$)
  
  improved backtracking, e.g., Davis–Putnam–Logemann–Loveland
  
  heuristic search in model space (sound but incomplete)
  
  e.g., min-conflicts-like hill-climbing algorithms
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 program?

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

Conjunctive Normal Form (CNF—universal) conjunction of disjunctions of literals clauses

E.g., \((A \lor \neg B) \land (B \lor \neg C \lor \neg D)\)

Resolution inference rule (for CNF): complete for propositional logic

\[
\ell_1 \lor \cdots \lor \ell_k, \quad m_1 \lor \cdots \lor m_n
\]

\[
\ell_1 \lor \cdots \lor \ell_{i-1} \lor \ell_{i+1} \lor \cdots \lor \ell_k \lor m_1 \lor \cdots \lor m_{j-1} \lor m_{j+1} \lor \cdots \lor m_n
\]

where \(\ell_i\) and \(m_j\) are complementary literals. E.g.,

\[
\frac{P_{1,3} \lor P_{2,2}, \quad \neg P_{2,2}}{P_{1,3}}
\]

Resolution is sound and complete for propositional logic

Resolution algorithm

Proof by contradiction, i.e., show \(KB \land \neg \alpha\) unsatisfiable
Summary

Logical agents apply **inference** to a **knowledge base**
   to derive new information and make decisions

Basic concepts of logic:
- **syntax**: formal structure of **sentences**
- **semantics**: truth of sentences wrt models
- **entailment**: necessary truth of one sentence given another
- **inference**: deriving sentences from other sentences
- **soundness**: derivations produce only entailed sentences
- **completeness**: derivations can produce all entailed sentences

Wumpus world requires the ability to represent partial and negated information, reason by cases, etc.

Forward, backward chaining are linear-time, complete for Horn clauses
Resolution is complete for propositional logic

Propositional logic lacks expressive power
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 $B_{1,1} \land P_{1,2}$ 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
Whereas propositional logic assumes world contains facts, first-order logic (like natural language) assumes the world contains

- **Objects**: people, houses, numbers, theories, Ronald McDonald, colors, baseball games, wars, centuries . . .
- **Relations**: red, round, bogus, prime, multistoried . . ., brother of, bigger than, inside, part of, has color, occurred after, owns, comes between, . . .
- **Functions**: father of, best friend, third inning of, one more than, end of . . .
Truth in first-order logic

Sentences are true with respect to a **model** and an **interpretation**

Model contains $\geq 1$ objects (**domain elements**) and relations among them

Interpretation specifies referents for:
- constant symbols $\rightarrow$ objects
- predicate symbols $\rightarrow$ relations
- function symbols $\rightarrow$ functional relations

An atomic sentence $\text{predicate}(\text{term}_1, \ldots, \text{term}_n)$ is true iff the objects referred to by $\text{term}_1, \ldots, \text{term}_n$ are in the relation referred to by $\text{predicate}$
Models for FOL: Lots!

Entailment in propositional logic can be computed by enumerating models

We can enumerate the FOL models for a given KB vocabulary:

For each number of domain elements $n$ from 1 to $\infty$
  For each $k$-ary predicate $P_k$ in the vocabulary
    For each possible $k$-ary relation on $n$ objects
      For each constant symbol $C$ in the vocabulary
        For each choice of referent for $C$ from $n$ objects . . .

Computing entailment by enumerating FOL models is not easy!
Universal quantification

\( \forall \) \{variables\} \ (sentence)  

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 \( P \)

\[ (At(KingJohn, Berkeley) \Rightarrow Smart(KingJohn)) \]
\[ \land (At(Richard, Berkeley) \Rightarrow Smart(Richard)) \]
\[ \land (At(Berkeley, Berkeley) \Rightarrow Smart(Berkeley)) \]
\[ \land \ldots \]
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

\[ \exists \langle \text{variables} \rangle \ \langle \text{sentence} \rangle \]

Someone at Stanford is smart:
\[ \exists x \ \text{At}(x, \text{Stanford}) \wedge \text{Smart}(x) \]

\[ \exists x \ P \] 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 \( P \)

\[ (\text{At}(\text{King John}, \text{Stanford}) \wedge \text{Smart}(\text{King John})) \]
\[ \lor (\text{At}(\text{Richard}, \text{Stanford}) \wedge \text{Smart}(\text{Richard})) \]
\[ \lor (\text{At}(\text{Stanford}, \text{Stanford}) \wedge \text{Smart}(\text{Stanford})) \]
\[ \lor \ldots \]
Another common mistake to avoid

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

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

$$\exists x \ A_t(x, Stanford) \Rightarrow Smart(x)$$

is true if there is anyone who is not at Stanford!
Properties of quantifiers

\( \forall x \ \forall y \) is the same as \( \forall y \ \forall x \) (why??)

\( \exists x \ \exists y \) is the same as \( \exists y \ \exists x \) (why??)

\( \exists x \ \forall y \) is not the same as \( \forall y \ \exists x \)

\( \exists x \ \forall y \ \text{Loves}(x, y) \)

"There is a person who loves everyone in the world"

\( \forall y \ \exists x \ \text{Loves}(x, y) \)

"Everyone in the world is loved by at least one person"

Quantifier duality: each can be expressed using the other

\( \forall x \ \text{Likes}(x, \text{IceCream}) \quad \neg \exists x \ \neg \text{Likes}(x, \text{IceCream}) \)

\( \exists x \ \text{Likes}(x, \text{Broccoli}) \quad \neg \forall x \ \neg \text{Likes}(x, \text{Broccoli}) \)
Knowledge base for the wumpus world

“Perception”
\[ \forall b, g, t \ Percept([\text{Smell}, b, g], t) \Rightarrow \text{Smelt}(t) \]
\[ \forall s, b, t \ Percept([s, b, \text{Glitter}], t) \Rightarrow \text{AtGold}(t) \]

Reflex: \[ \forall t \ \text{AtGold}(t) \Rightarrow \text{Action(Grab, t)} \]

Reflex with internal state: do we have the gold already?
\[ \forall t \ \text{AtGold}(t) \land \neg \text{Holding(Gold, t)} \Rightarrow \text{Action(Grab, t)} \]

\text{Holding(Gold, t)} cannot be observed
\[ \Rightarrow \text{keeping track of change is essential} \]
Reduction to propositional inference

Suppose the KB contains just the following:

\[ \forall x \ King(x) \land Greedy(x) \Rightarrow Evil(x) \]
\[ King(John) \]
\[ Greedy(John) \]
\[ Brother(Richard, John) \]

Instantiating the universal sentence in all possible ways, we have

\[ King(John) \land Greedy(John) \Rightarrow Evil(John) \]
\[ King(Richard) \land Greedy(Richard) \Rightarrow Evil(Richard) \]
\[ King(John) \]
\[ Greedy(John) \]
\[ Brother(Richard, John) \]

The new KB is propositionalized: proposition symbols are

\[ King(John), Greedy(John), Evil(John), King(Richard) \text{ etc.} \]
Keeping track of change

Facts hold in situations, rather than eternally
E.g., Holding(Gold, Now) rather than just Holding(Gold)

Situation calculus is one way to represent change in FOL:
Add a situation argument to each non-eternal predicate
E.g., Now in Holding(Gold, Now) denotes a situation

Situations are connected by the Result function
Result(a, s) is the situation that results from doing a in s
Describing actions I

“Effect” axiom—describe changes due to action
\[ \forall s \ AtGold(s) \Rightarrow Holding(Gold, Result(Grab, s)) \]

“Frame” axiom—describe non-changes due to action
\[ \forall s \ HaveArrow(s) \Rightarrow HaveArrow(Result(Grab, s)) \]

Frame problem: find an elegant way to handle non-change
   (a) representation—avoid frame axioms
   (b) inference—avoid repeated “copy-overs” to keep track of state

Qualification problem: true descriptions of real actions require endless caveats—what if gold is slippery or nailed down or . . .

Ramification problem: real actions have many secondary consequences—what about the dust on the gold, wear and tear on gloves, . . .
Problems with propositionalization

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

\[ \forall x \ King(x) \land Greedy(x) \Rightarrow Evil(x) \]
\[ King(John) \]
\[ \forall y \ Greedy(y) \]
\[ Brother(Richard, John) \]

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

With function symbols, it gets much much worse!
Unification

We can get the inference immediately if we can find a substitution \( \theta \) such that \( \text{King}(x) \) and \( \text{Greedy}(x) \) match \( \text{King}(\text{John}) \) and \( \text{Greedy}(y) \)

\[ \theta = \{x/\text{John}, y/\text{John}\} \] works

\( \text{UNIFY}(\alpha, \beta) = \theta \) if \( \alpha \theta = \beta \theta \)

\[
\begin{align*}
p & & q & & \theta \\
\text{Knows}(\text{John}, x) & & \text{Knows}(\text{John}, \text{Jane}) & & \{x/\text{Jane}\} \\
\text{Knows}(\text{John}, x) & & \text{Knows}(y, \text{OJ}) & & \{x/\text{OJ}, y/\text{John}\} \\
\text{Knows}(\text{John}, x) & & \text{Knows}(y, \text{Mother}(y)) & & \{y/\text{John}, x/\text{Mother}(\text{John})\} \\
\text{Knows}(\text{John}, x) & & \text{Knows}(x, \text{OJ}) & & \text{fail}
\end{align*}
\]

Standardizing apart eliminates overlap of variables, e.g., \( \text{Knows}(z_{17}, \text{OJ}) \)
Generalized Modus Ponens (GMP)

\[ \frac{p_1', \ p_2', \ \ldots, \ p_n', \ (p_1 \land p_2 \land \ldots \land p_n \Rightarrow q)}{q\theta} \]  
where \( p_i'\theta = p_i\theta \) for all \( i \)

\( p_1' \) is \( \text{King}(\text{John}) \)  \hspace{1cm} \( p_1 \) is \( \text{King}(x) \)
\( p_2' \) is \( \text{Greedy}(y) \)  \hspace{1cm} \( p_2 \) is \( \text{Greedy}(x) \)
\( \theta \) is \( \{x/\text{John}, y/\text{John}\} \)  \hspace{1cm} q is \( \text{Evil}(x) \)
\( q\theta \) is \( \text{Evil}(\text{John}) \)

GMP used with KB of definite clauses \((\text{exactly} \ one \ positive \ literal)\)
All variables assumed universally quantified
Forward chaining algorithm

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

repeat until new is empty
    new ← {} 
    for each sentence r in KB do 
        (p₁ ∧ … ∧ pₙ ⇒ q) ← STANDARDIZE-Apart(r) 
        for each θ such that (p₁ ∧ … ∧ pₙ)θ = (p'₁ ∧ … ∧ p'ₙ)θ 
            for some p'₁, …, p'ₙ in KB 
                q' ← SUBST(θ, q) 
                if q' is not a renaming of a sentence already in KB or new then do 
                    add q' to new 
                    φ ← UNIFY(q', α) 
                    if φ is not fail then return φ 
                add new to KB 
    return false
Properties of forward chaining

Sound and complete for first-order definite clauses
(proof similar to propositional proof)

Datalog = first-order definite clauses + no functions (e.g., crime KB)
FC terminates for Datalog in poly iterations: at most $p \cdot n^k$ literals

May not terminate in general if $\alpha$ is not entailed

This is unavoidable: entailment with definite clauses is semidecidable
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 {θ}
q' ← Subst(θ, First(goals))
for each sentence r in KB
    where Standardize-Apart(r) = (p₁ ∧ ... ∧ pₙ ⇒ q)
    and θ' ← Unify(q, q') succeeds
    new goals ← [p₁, ..., pₙ | Rest(goals)]
    answers ← FOL-BC-Ask(KB, new-goals, Compose(θ', θ)) ∪ answers
return answers
Properties of backward chaining

Depth-first recursive proof search: space is linear in size of proof

Incomplete due to infinite loops
  ⇒ fix by checking current goal against every goal on stack

Inefficient due to repeated subgoals (both success and failure)
  ⇒ fix using caching of previous results (extra space!)

Widely used (without improvements!) for logic programming
Prolog systems

Basis: backward chaining with Horn clauses + bells & whistles
Widely used in Europe, Japan (basis of 5th Generation project)
Compilation techniques ⇒ approaching a billion LIPS

Program = set of clauses = head :- literal₁, ... literalₙ.

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

Efficient unification by open coding
Efficient retrieval of matching clauses by direct linking
Depth-first, left-to-right backward chaining
Built-in predicates for arithmetic etc., e.g., X is Y*Z+3
Closed-world assumption ("negation as failure")
   e.g., given alive(X) :- not dead(X).
   alive(joe) succeeds if dead(joe) fails
Resolution: brief summary

Full first-order version:

\[
\ell_1 \lor \cdots \lor \ell_k, \quad m_1 \lor \cdots \lor m_n
\]

\[
(\ell_1 \lor \cdots \lor \ell_{i-1} \lor \ell_{i+1} \lor \cdots \lor \ell_k \lor m_1 \lor \cdots \lor m_{j-1} \lor m_{j+1} \lor \cdots \lor m_n) \theta
\]

where UNIFY(\(\ell_i, \neg \text{m}_j\)) = \(\theta\).

For example,

\[
\neg \text{Rich}(x) \lor \text{Unhappy}(x)
\]

\[
\text{Rich(Ken)}
\]

\[
\text{Unhappy(Ken)}
\]

with \(\theta = \{x/\text{Ken}\}\)

Apply resolution steps to \(CNF(KB \land \neg \alpha)\); complete for FOL
Conversion to CNF

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
\[ \forall x \ [\neg \forall y \ \neg Animal(y) \lor Loves(x, y)] \lor [\exists y \ Loves(y, x)] \]

2. Move \( \neg \) inwards: \( \neg \forall x, p \equiv \exists x \ \neg p, \ \neg \exists x, p \equiv \forall x \ \neg p: \)
\[ \forall x \ [\exists y \ \neg (\neg Animal(y) \lor Loves(x, y))] \lor [\exists y \ Loves(y, x)] \]
\[ \forall x \ [\exists y \ \neg Animal(y) \land \neg Loves(x, y)] \lor [\exists y \ Loves(y, x)] \]
\[ \forall x \ [\exists y \ Animal(y) \land \neg Loves(x, y)] \lor [\exists y \ Loves(y, x)] \]
Conversion to CNF contd.

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 \( \lor \):

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