Logic: Top-down proof procedure, Datalog, Big Picture

Alan Mackworth

UBC CS 322 - Logic 4
March 8, 2013

Textbook §5.2, §12.3
Lecture Overview

Recap: Bottom-up proof procedure is sound and complete

• Top-down Proof Procedure

• Datalog

• Logics: Big Picture
Logical consequence and BU proofs

Definition (logical consequence)
If KB is a set of clauses and g is a conjunction of atoms, g is a logical consequence of KB, written KB ⊧ g, if g is true in every model of KB.

Example: KB = \{h ← a, a, a ← c\}. Then KB ⊧ ?

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>c</th>
<th>h</th>
<th>h ← a</th>
<th>a</th>
<th>a ← c</th>
<th>Model of KB</th>
</tr>
</thead>
<tbody>
<tr>
<td>I₁</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>no</td>
</tr>
<tr>
<td>I₂</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>no</td>
</tr>
<tr>
<td>I₃</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>no</td>
</tr>
<tr>
<td>I₄</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>no</td>
</tr>
<tr>
<td>I₅</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td>no</td>
</tr>
<tr>
<td>I₆</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>yes</td>
</tr>
<tr>
<td>I₇</td>
<td>T</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td>no</td>
</tr>
<tr>
<td>I₈</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>yes</td>
</tr>
</tbody>
</table>

Which atoms are entailed?
### Logical consequence and BU proofs

**Definition (logical consequence)**

If $KB$ is a set of clauses and $g$ is a conjunction of atoms, $g$ is a logical consequence of $KB$, written $KB \models g$, if $g$ is true in every model of $KB$.

---

**Example:** $KB = \{h \leftarrow a, a, a \leftarrow c\}$. Then $KB \models ?$

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>c</th>
<th>h</th>
<th>h ← a</th>
<th>a</th>
<th>a ← c</th>
<th>Model of KB</th>
</tr>
</thead>
<tbody>
<tr>
<td>$I_1$</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>no</td>
</tr>
<tr>
<td>$I_2$</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>no</td>
</tr>
<tr>
<td>$I_3$</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>no</td>
</tr>
<tr>
<td>$I_4$</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>no</td>
</tr>
<tr>
<td>$I_5$</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td>no</td>
</tr>
<tr>
<td>$I_6$</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>yes</td>
</tr>
<tr>
<td>$I_7$</td>
<td>T</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td>no</td>
</tr>
<tr>
<td>$I_8$</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>yes</td>
</tr>
</tbody>
</table>

Which atoms are entailed? $KB \models a$ and $KB \models h$.
What does BU derive for the KB above?

Definition (logical consequence)
If KB is a set of clauses and g is a conjunction of atoms, g is a logical consequence of KB, written \( KB \vdash g \), if g is true in every model of KB.

Example: KB = \{h ← a, a, a ← c\}. Then KB \( \vdash a \) and KB \( \nvdash h \).

BU proof procedure

\[
\begin{align*}
C := {}; \\
\text{repeat} \\
\quad \text{select clause } h \leftarrow b_1 \land \ldots \land b_m \text{ in KB} \\
\quad \text{such that } b_i \in C \text{ for all } i, \text{ and } h \notin C; \\
\quad C := C \cup \{h\} \\
\text{until no more clauses can be selected. } KB \vdash_{\text{BU}} g \text{ if and only if } g \in C
\end{align*}
\]

What does BU derive for the KB above?
Logical consequence and BU proofs

Definition (logical consequence)
If KB is a set of clauses and g is a conjunction of atoms, g is a logical consequence of KB, written $\text{KB} \vdash g$, if g is true in every model of KB.

Example: $\text{KB} = \{h \leftarrow a, a, a \leftarrow c\}$. Then $\text{KB} \vdash a$ and $\text{KB} \vdash h$.

<table>
<thead>
<tr>
<th>C := {}</th>
<th>BU proof procedure</th>
</tr>
</thead>
<tbody>
<tr>
<td>repeat</td>
<td></td>
</tr>
<tr>
<td>select clause $h \leftarrow b_1 \land \ldots \land b_m$ in KB such that $b_i \in C$ for all i, and $h \notin C$;</td>
<td></td>
</tr>
<tr>
<td>$C := C \cup {h}$</td>
<td></td>
</tr>
<tr>
<td>until no more clauses can be selected. $\text{KB} \vdash_{\text{BU}} g$ if and only if $g \in C$</td>
<td></td>
</tr>
</tbody>
</table>

What does BU derive for the KB above?
Trace: $\{a\}, \{a,h\}$. Thus $\text{KB} \vdash_{\text{BU}} a$ and $\text{KB} \vdash_{\text{BU}} h$.
Exactly the logical consequences!
Summary for bottom-up proof procedure BU

- Proved last time
  - BU is sound: it derives only atoms that logically follow from KB
  - BU is complete: it derives all atoms that logically follow from KB

- Together: it derives exactly the atoms that logically follow from KB!
  - That’s why the results for ⊨ and ⊨_{BU} matched for the example above

- And, it is efficient!
  - Outer loop linear in the number of clauses in KB
    - Each clause is used maximally once by BU
Learning Goals Up To Here

• PDCL syntax & semantics
  – Verify whether a logical statement belongs to the language of propositional definite clauses
  – Verify whether an interpretation is a model of a PDCL KB.
  – Verify when a conjunction of atoms is a logical consequence of a knowledge base

• Bottom-up proof procedure
  • Define/read/write/trace/debug the Bottom Up (BU) proof procedure
  • Prove that the BU proof procedure is sound and complete
Lecture Overview

• Recap: Bottom-up proof procedure is sound and complete

Top-down Proof Procedure

• Datalog

• Logics: Big Picture
Bottom-up vs. Top-down

**Bottom-up**

$g$ is proved if $g \in C$

When does BU look at the query $g$?

- In every loop iteration
- Never
- At the end
- At the beginning
**Bottom-up vs. Top-down**

- **Key Idea of top-down:** search backward from a query $g$ to determine if it can be derived from $KB$.

**Bottom-up**

- $KB \rightarrow C$
- $g$ is proved if $g \in C$

**Top-down**

- Query $g$
- $KB \rightarrow$ answer
- We’ll see how $g$ is proved

When does BU look at the query $g$?
- Never
- It derives the same $C$ regardless of the query

TD performs a backward search starting at $g$
Top-down Ground Proof Procedure

Idea: search backward from a query

An answer clause is of the form:

\[ \text{yes} \leftarrow a_1 \land \ldots \land a_m \]

where \( a_1, \ldots, a_m \) are atoms

We express the query as an answer clause

- E.g. query \( q_1 \land \ldots \land q_k \) is expressed as \( \text{yes} \leftarrow q_1 \land \ldots \land q_k \)

Basic operation: SLD Resolution of an answer clause

\[ \text{yes} \leftarrow c_1 \land \ldots \land c_{i-1} \land c_i \land c_{i+1} \ldots \land c_m \]

on an atom \( c_i \) with another clause

\[ c_i \leftarrow b_1 \land \ldots \land b_p \]

yields the clause

\[ \text{yes} \leftarrow c_1 \land \ldots \land c_{i-1} \land b_1 \land \ldots \land b_p \land c_{i+1} \ldots \land c_m \]
Rules of derivation in top-down and bottom-up

Top-down:
SLD Resolution

\[
\text{yes} \leftarrow c_1 \land \ldots \land c_{i-1} \land c_i \land c_{i+1} \ldots \land c_m \\
\text{yes} \leftarrow c_1 \land \ldots \land c_{i-1} \land b_1 \land \ldots \land b_p \land c_{i+1} \ldots \land c_m
\]

Bottom-up:
Generalized modus ponens

\[
h \leftarrow b_1 \land \ldots \land b_m \\
h \leftarrow b_1 \land \ldots \land b_m
\]
Example for (successful) SLD derivation

\[ a \leftarrow b \land c. \]
\[ c \leftarrow e. \]
\[ f \leftarrow j \land e. \]

\[ \begin{align*}
1 & \quad a \leftarrow e \land f. \\
2 & \quad f. \\
3 & \quad b \leftarrow f \land k. \\
3 & \quad e. \\
3 & \quad j \leftarrow c.
\end{align*} \]

\[ d \leftarrow k \]

**Query:** \(?a\)

\[ \begin{align*}
\gamma_0: & \quad \text{yes} \leftarrow a \\
\gamma_1: & \quad \text{yes} \leftarrow e \land f \\
\gamma_2: & \quad \text{yes} \leftarrow e \\
\gamma_3: & \quad \text{yes} \leftarrow
\end{align*} \]

Done. The question was “Can we derive \(a\)?”

The answer is “Yes, we can”
SLD Derivations

• An answer is an answer clause with $m = 0$.

  yes ← .

• A successful derivation from $KB$ of query $\, ?q_1 \land \ldots \land q_k$ is a sequence of answer clauses $\gamma_0, \gamma_1, \ldots, \gamma_n$ such that
  ▪ $\gamma_0$ is the answer clause $\, yes ← q_1 \land \ldots \land q_k$.
  ▪ $\gamma_i$ is obtained by resolving $\gamma_{i-1}$ with a clause in $KB$, and
  ▪ $\gamma_n$ is an answer $\, yes ←$

• An unsuccessful derivation from $KB$ of query $\, ?q_1 \land \ldots \land q_k$
  ▪ We get to something like $\, yes ← b_1 \land \ldots \land b_k$, where
    there is no clause in $KB$ with any of the $b_i$ as its head
Top-down Proof Procedure for PDCL

To solve the query \(? q_1 \land ... \land q_k:\)

\[\text{ac}:= \text{yes} \leftarrow \text{body, where body is } q_1 \land ... \land q_k\]

repeat

select \(q_i \in \text{body};\)

choose clause \(C \in \text{KB}, C \text{ is } q_i \leftarrow b_c;\)

replace \(q_i \) in body by \(b_c\)

until \(\text{ac} \) is an answer (fail if no clause with \(q_i\) as head)

select: any choice will work

(“Don’t care non-determinism”)

choose: truly non-deterministic, must pick the right one

(“Don’t know non-determinism”)
Example for failing SLD derivation

\[
\begin{align*}
\text{Query: } ?a \\
\gamma_0: & \text{ yes } \leftarrow a \\
\gamma_1: & \text{ yes } \leftarrow e \land f \\
\gamma_2: & \text{ yes } \leftarrow e \land k \\
\gamma_3: & \text{ yes } \leftarrow k
\end{align*}
\]

“Can we derive a?”
“This time we failed”

There is no rule with \(k\) as its head, thus … fail
Correspondence between BU and TD proofs

If the following is a top-down (TD) derivation in a given KB, what would be the bottom-up (BU) derivation of the same query?

TD derivation

```
yes ← a.
yes ← b ∧ f.
yes ← b ∧ g ∧ h.
yes ← c ∧ d ∧ g ∧ h.
yes ← d ∧ g ∧ h.
yes ← g ∧ h.
yes ← h.
yes ← .
```
Correspondence between BU and TD proofs

If the following is a top-down (TD) derivation in a given KB, what would be the bottom-up (BU) derivation of the same query?

TD derivation

yes ← a.
yes ← b ∧ f.
yes ← b ∧ g ∧ h.
yes ← c ∧ d ∧ g ∧ h.
yes ← d ∧ g ∧ h.
yes ← g ∧ h.
yes ← h.
yes ← .

BU derivation

{}{h}{g,h}{d,g,h}{c,d,g,h}{b,c,d,g,h}{b,c,d,f,g,h}{a,b,c,d,f,g,h}
Is the Top-down procedure sound and complete?

• Yes, since there is a 1:1 correspondence between top-down and bottom-up proofs
  – The two methods derive exactly the same atoms (if the SLD resolution picks the successful derivations)
Search Graph for Top-down proofs

Query: \(?a \land d\).

\begin{align*}
    a & \leftarrow b \land c. & a & \leftarrow g. \\
    a & \leftarrow h. & b & \leftarrow j. \\
    b & \leftarrow k. & d & \leftarrow m. \\
    d & \leftarrow p. & f & \leftarrow m. \\
    f & \leftarrow p. & g & \leftarrow m. \\
    g & \leftarrow f. & k & \leftarrow m. \\
    h & \leftarrow m. & p & .
\end{align*}

What kind of search is SLD resolution?

Breadth-first search  Depth-first-search
Search Graph for Top-down proofs

Query: \(?a \land d\).

\begin{align*}
a & \leftarrow b \land c. & a & \leftarrow g. \\
a & \leftarrow h. & b & \leftarrow j. \\
b & \leftarrow k. & d & \leftarrow m. \\
d & \leftarrow p. & f & \leftarrow m. \\
f & \leftarrow p. & g & \leftarrow m. \\
g & \leftarrow f. & k & \leftarrow m. \\
h & \leftarrow m. & p. & \\
\end{align*}

What kind of search is SLD resolution?

It’s a depth-first-search. Failing resolutions are paths where the search has to backtrack.
Search Graph for Top-down proofs

Query: \( ?a \land d \).

\[
\begin{align*}
  a & \leftarrow b \land c. \\
  a & \leftarrow h. \\
  b & \leftarrow k. \\
  d & \leftarrow p. \\
  f & \leftarrow p. \\
  g & \leftarrow f. \\
  h & \leftarrow m. \\
\end{align*}
\]

Q: Can we use heuristics?
A: Yes! E.g. number of atoms in the answer clause

Admissible? Yes No
Search Graph for Top-down proofs

Query: $?a \land d$.

$$a \leftarrow b \land c.$$ $$a \leftarrow h.$$ $$a \leftarrow g.$$ $$b \leftarrow j.$$ $$b \leftarrow k.$$ $$b \leftarrow j.$$ $$b \leftarrow k.$$ $$d \leftarrow m.$$ $$d \leftarrow p.$$ $$d \leftarrow m.$$ $$f \leftarrow m.$$ $$f \leftarrow p.$$ $$f \leftarrow m.$$ $$g \leftarrow m.$$ $$g \leftarrow f.$$ $$g \leftarrow m.$$ $$k \leftarrow m.$$ $$h \leftarrow m.$$ $$h \leftarrow m.$$ $$p.$$ $$p.$$ $$p.$$ 

**Q:** Can we use heuristics?

**A:** Yes! *E.g. number of atoms in the answer clause*

**Admissible?** Yes, you need at least these many SLD steps to get an answer.
Inference as Standard Search

• **Constraint Satisfaction (Problems):**
  - **State:** assignments of values to a subset of the variables
  - **Successor function:** assign values to a “free” variable
  - **Goal test:** set of constraints
  - **Solution:** possible world that satisfies the constraints
  - **Heuristic function:** none (all solutions at the same distance from start)

• **Planning :**
  - **State:** full assignment of values to features
  - **Successor function:** states reachable by applying valid actions
  - **Goal test:** partial assignment of values to features
  - **Solution:** a sequence of actions
  - **Heuristic function:** relaxed problem! E.g. “ignore delete lists”

• **Inference (Top-down/SLD resolution)**
  - **State:** answer clause of the form \( \text{yes} \leftarrow q_1 \land \ldots \land q_k \)
  - **Successor function:** all states resulting from substituting first atom \( a \) with \( b_1 \land \ldots \land b_m \) if there is a clause \( a \leftarrow b_1 \land \ldots \land b_m \)
  - **Goal test:** is the answer clause empty (i.e. \( \text{yes} \leftarrow \)) ?
  - **Solution:** the proof, i.e. the sequence of SLD resolutions
  - **Heuristic function:** number of atoms in the query clause
Lecture Overview

• Recap: Bottom-up proof procedure is sound and complete

• Top-down Proof Procedure

Datalog

• Logics: Big Picture
Representation and Reasoning in complex domains

- Expressing knowledge with **propositions** can be quite limiting

\[
\begin{align*}
  & up(s_2) \\
  & up(s_3) \\
  & ok(cb_1) \\
  & ok(cb_2) \\
  & live(w_1) \\
  & connected(w_1, w_2)
\end{align*}
\]

- It is often **natural** to consider **individuals** and their properties

\[
\begin{align*}
  & up(s_2) \\
  & up(s_3) \\
  & ok(cb_1) \\
  & ok(cb_2) \\
  & live(w_1) \\
  & connected(w_1, w_2)
\end{align*}
\]

E.g. there is no notion that  

**\textit{w}_1 \textit{is the same in live}_w_1**  

and in **\textit{connected}_w_1_w_2**

Now there is a notion that  

**\textit{w}_1 \textit{is the same in live}(w_1)**  

and in **\textit{connected}(w_1, w_2)**
Datalog: What do we gain?

• An extension of propositional definite clause (PDC) logic
  – We now have **variables**
  – We now have **relationships** between variables
  – We can express knowledge that holds for a set of individuals, writing more powerful clauses by introducing variables, such as:

\[
\text{live}(W) \leftarrow \text{wire}(W) \land \text{connected_to}(W, W_1) \\
\quad \land \text{wire}(W_1) \land \text{live}(W_1).
\]

  – We can ask **generic queries**,
    • E.g. “which wires are connected to \( w_1 \)?”

\[
? \text{connected_to}(W, w_1)
\]
**Datalog: a relational rule language**

Datalog expands the syntax of PDCL.

<table>
<thead>
<tr>
<th><strong>A variable</strong></th>
<th>is a symbol starting with an upper case letter</th>
<th>Examples: X, Y</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>A constant</strong></td>
<td>is a symbol starting with lower-case letter or a sequence of digits.</td>
<td>Examples: alan, w1</td>
</tr>
<tr>
<td><strong>A term</strong></td>
<td>is either a variable or a constant.</td>
<td>Examples: X, Y, alan, w1</td>
</tr>
<tr>
<td><strong>A predicate symbol</strong></td>
<td>is a symbol starting with a lower-case letter.</td>
<td>Examples: live, connected, part-of, in</td>
</tr>
</tbody>
</table>
An **atom** is a symbol of the form $p$ or $p(t_1 \ldots t_n)$ where $p$ is a predicate symbol and $t_i$ are terms

Examples: sunny, in(alan,X)

A **definite clause** is either an atom (a fact) or of the form:

$$h \leftarrow b_1 \land \ldots \land b_m$$

where $h$ and the $b_i$ are atoms (Read this as `h if b.`)

Example: in(X,Z) $\leftarrow$ in(X,Y) $\land$ part-of(Y,Z)

A **knowledge base** is a set of definite clauses
Datalog Semantics

- Role of semantics is still to connect symbols and sentences in the language with the target domain. Main difference:
  - need to create correspondence both between terms and individuals, as well as between predicate symbols and relations

We won’t cover the formal definition of Datalog semantics, but if you are interested see 12.3.1 and 12.3.2 in textbook.
Datalog: Top Down Proof Procedure

- Extension of Top-Down procedure for PDCL. How do we deal with variables?
  - Idea:
    - Find clauses with heads that match the query
    - Substitute variable in the clause with the matching constant
  - Example:
    - Query: yes ← in(alan, cs_building).
    - in(X,Y) ← part_of(Z,Y) & in(X,Z).
      with Y = cs_building
      yes ← part_of(Z,cs_building), in(alan, Z).

- We will not cover the formal details of this process, called *unification*. See P&M Section 12.4.2, p. 511 for the details.
Example proof of a Datalog query

\[
\begin{align*}
\text{in(alan, r123).} \\
\text{part_of(r123,cs_building).} \\
\text{in(X,Y) ← part_of(Z,Y) & in(X,Z).}
\end{align*}
\]

**Query:** yes ← in(alan, cs_building).

\[
\begin{align*}
\text{yes ← part_of(Z,cs_building), in(alan, Z).}
\end{align*}
\]

Using clause: in(X,Y) ← part_of(Z,Y) & in(X,Z), with Y = cs_building

Using clause: part_of(r123,cs_building) with Z = r123

Using clause: in(alan, r123).

Using clause: in(alan, r123).

yes ←.

yes ← part_of(Z, r123), in(alan, Z).

Using clause: in(X,Y) ← part_of(Z,Y) & in(X,Z). With Z = alan

No clause with matching head: part_of(Z,r123).

fail
Tracing Datalog proofs in Alspace


- Question 4 of assignment 3 asks you to use this applet
Query:  in(alan, X1).
    yes(X1) ← in(alan, X1).

What would the answer(s) be?
Datalog: queries with variables

in(alan, r123).
part_of(r123, cs_building).
in(X,Y) ← part_of(Z,Y) & in(X,Z).

Query:  in(alan, X1).
         yes(X1) ← in(alan, X1).

What would the answer(s) be?
  yes(r123).
  yes(cs_building).

You can trace the SLD derivation for this query in the AIspace Deduction Applet, using file http://cs.ubc.ca/~mack/CS322/in-part-of.pl
One important Datalog detail

- In its SLD resolution proof, Datalog always chooses the first clause with a matching head it finds in KB
- What does that mean for recursive function definitions?
  - The clause(s) defining your base case(s) have to appear first in KB
  - Otherwise, you can get infinite recursions
  - This is similar to recursion in imperative and functional programming languages
- Datalog is a subset of Prolog (Programming in Logic)
Learning Goals For Logic

• PDCL syntax & semantics
  – Verify whether a logical statement belongs to the language of propositional definite clauses
  – Verify whether an interpretation is a model of a PDCL KB.
  – Verify when a conjunction of atoms is a logical consequence of a KB

• Bottom-up proof procedure
  – Define/read/write/trace/debug the Bottom Up (BU) proof procedure
  – Prove that the BU proof procedure is sound and complete

• Top-down proof procedure
  – Define/read/write/trace/debug the Top-down (SLD) proof procedure (as a search problem)

• Datalog
  – Represent simple domains in Datalog
  – Apply the Top-down proof procedure in Datalog
Lecture Overview

- Recap: Bottom-up proof procedure is sound and complete
- Top-down Proof Procedure
- Datalog

Logics: Big Picture
Logics: Big picture

- Propositional Definite Clause Logics (PDCL)
- Propositional Logics
- Description Logics
- Ontologies
- Semantic Web
- Information Extraction
- Cognitive Architectures
- Production Systems
- Video Games
- Summarization
- Tutoring Systems
- Hardware Verification
- Software Verification
- Product Configuration
- Satisfiability Testing (SAT)
- From CSP module
- Semantics and Proof Theory
- Soundness & Completeness
- Datalog
- First-Order Logics
- From CSP module

PDCL

Semantics and Proof Theory

Soundness & Completeness

Datalog

First-Order Logics

Satisfiability Testing (SAT)

From CSP module

Hardware Verification

Software Verification

Product Configuration

you know you know
Logics: Big picture

• We only covered rather simple logics
  – There are much more powerful representation and reasoning systems based on logics e.g. full first order logic (with negation, disjunction and function symbols), second-order logics, non-monotonic logics, modal logics, ...

• There are many important applications of logic
  – For example, software agents roaming the web on our behalf
    • Based on a more structured representation: the semantic web
    • This is just one example for how logics are used
Example problem: automated travel agent

• Examples for typical queries
  – How much is a typical flight to Mexico for a given date?
  – What’s the cheapest vacation package to some place in the Caribbean in a given week?
    • Plus, the hotel should have a white sandy beach and scuba diving

• If webpages are based on basic HTML
  – Humans need to scout for the information and integrate it
  – Computers are not reliable enough (yet?)
    • Natural language processing can be powerful (see Watson and Siri!)
    • But some information may be in pictures (beach), or implicit in the text, so simple approaches like Watson and Siri still don’t get all the info.
More structured representation: the Semantic Web

• Beyond HTML pages only made for humans
• Languages and formalisms based on logics that allow websites to include information in a more structured format
  – Goal: software agents that can roam the web and carry out sophisticated tasks on our behalf.
  – This is different than searching content for keywords and popularity!
• For further material, see P&M text, Chapter 13 and the Introduction to the Semantic Web tutorial given at 2011 Semantic Technology Conference http://www.w3.org/People/Ivan/CorePresentations/SWTutorial/ (This is the best technical intro; Herman keeps it up to date.)
Examples of ontologies for the Semantic Web

• “Ontology”: logic-based representation of the world

• eClassOwl: eBusiness ontology
  – for products and services
  – 75,000 classes (types of individuals) and 5,500 properties

• National Cancer Institute’s ontology: 58,000 classes

• Open Biomedical Ontologies Foundry: several ontologies
  – including the Gene Ontology to describe
    • gene and gene product attributes in any organism or protein sequence
    • annotation terminology and data

• OpenCyc project: a 150,000-concept ontology including
  – Top-level ontology
    • describes general concepts such as numbers, time, space, etc
  – Hierarchical composition: superclasses and subclasses
  – Many specific concepts such as “OLED display”, “iPhone”
Course Overview

Environment

Deterministic

- Arc Consistency
- Search

Stochastic

- Bayesian Networks
  - Variable Elimination
- Decision Networks
  - Variable Elimination
- Markov Processes
  - Value Iteration

Problem Type

- Static
  - Logic
  - Constraint Satisfaction
- Sequential
  - Planning

Course Module

Representation

- Reasoning Technique

Reasoning Technique

Uncertainty

Decision Theory

Course Module

This concludes the logic module