Lecture 16
Bottom Up and Top Down Proof Procedure
(Ch 5.2.2)
Announcements

• After today you will be able to do up to Q3 in the assignment
Lecture Overview

- Recap Lecture 15
  - Bottom-Up Proof Procedure
    - Soundness
    - Completeness
  - Top-Down (TD) Proof Procedure
  - TD as Search
  - Datalog (time permitting)
Where Are We?

Environment

Problem Type

Deterministic

Stochastic

Constraint Satisfaction

Static

Query

Sequential

Search

Logics

Strips

Belief Nets

Decision Nets

Markov Processes

Representation

Reasoning Technique

Arc Consistency

Search

Vars + Constraints

Variable Elimination

Variable Elimination

Value Iteration

Back to static problems, but with richer representation
Logic: a framework for representation & reasoning

- When we **represent a domain** about which we have only partial (but certain) information, we need to represent.....
  - Objects, properties, sets, groups, actions, events, time, space, ...
- All these can be represented as
  - Objects
  - Relationships between objects
- Logic is the language to express knowledge about the world this way

We will start with a simple logic

Primitive elements are **propositions**: Boolean variables that can be 
{true, false}

Two kinds of statements:
- that a proposition is true
- that a proposition is true if one or more other propositions are true
To Define a Logic We Need

- **Syntax**: specifies the symbols used, and how they can be combined to form legal sentences.
  - Knowledge base is a set of sentences in the language.
- **Semantics**: specifies the meaning of symbols and sentences.
- **Reasoning theory or proof procedure**: a specification of how an answer can be produced.
  - **Sound**: only generates correct answers with respect to the semantics.
  - **Complete**: Guaranteed to find an answer if it exists.
## Propositional Definite Clauses: Syntax

### Definition (atom)
An **atom** is a symbol starting with a lower case letter.  

**Examples:**
- `p_1;`  
- `live_I_1`

### Definition (body)
A **body** is an atom or is of the form \( b_1 \land b_2 \) where \( b_1 \) and \( b_2 \) are bodies.  

**Examples:**
- `p_1 \land p_2;`  
- `ok_w_1 \land live_w_0`

### Definition (definite clause)
A **definite clause** is

- an atom or  
- a rule of the form `h ← b` where \( h \) is an atom ("head") and \( b \) is a body. (Read this as "\( h \) if \( b \)".)  

**Examples:**
- `p_1 ← p_2;`  
- `live_w_0 ← live_w_1 \land up_s_2`

### Definition (KB)
A **knowledge base (KB)** is a set of definite clauses.
atoms

definite clauses, KB

rules
Propositional Definite Clauses: Semantics

Definition (interpretation)
An interpretation $I$ assigns a truth value to each atom.

Definition (truth values of statements)
- A body $b_1 \land b_2$ is true in $I$ if and only if $b_1$ is true in $I$ and $b_2$ is true in $I$.
- A rule $h \leftarrow b$ is false in $I$ if and only if $b$ is true in $I$ and $h$ is false in $I$.
- A knowledge base $KB$ is true in $I$ if and only if every clause in $KB$ is true in $I$. 
PDC Semantics: Knowledge Base (KB)

- A knowledge base $KB$ is true in $I$ if and only if every clause in $KB$ is true in $I$.

<table>
<thead>
<tr>
<th>p</th>
<th>q</th>
<th>r</th>
<th>s</th>
</tr>
</thead>
<tbody>
<tr>
<td>$I_1$</td>
<td>false</td>
<td>true</td>
<td>true</td>
</tr>
</tbody>
</table>

$KB_1$: $p$

$KB_2$: $r$
$s \leftarrow p$
$q \leftarrow p \land s$

$KB_3$: $r$
$q$
$s \leftarrow q$

Only $KB_2$ above is True in $I_1$
### Propositional Definite Clauses: Semantics

**Definition (interpretation)**
An interpretation $I$ assigns a truth value to each atom.

**Definition (truth values of statements)**
- A body $b_1 \land b_2$ is true in $I$ if and only if $b_1$ is true in $I$ and $b_2$ is true in $I$.
- A rule $h \leftarrow b$ is false in $I$ if and only if $b$ is true in $I$ and $h$ is false in $I$.

- A knowledge base $KB$ is true in $I$ if and only if every clause in $KB$ is true in $I$.

**Definition (model)**
A model of a knowledge base $KB$ is an interpretation in which $KB$ is true.

Similar to CSPs: a model of a set of clauses is an interpretation that makes all of the clauses true.
**Definition (model)**
A **model** of a knowledge base KB is an interpretation in which every clause in KB is true.

KB = \[
\begin{align*}
& \begin{align*}
& p \leftarrow q \\
& q
\end{align*} \\
& r \leftarrow s
\end{align*}
\]

Which of the interpretations below are models of KB?
All interpretations where KB is true: \(I_1, I_3, \) and \(I_4\)

<table>
<thead>
<tr>
<th></th>
<th>p</th>
<th>q</th>
<th>r</th>
<th>s</th>
<th>p ← q</th>
<th>q</th>
<th>r ← s</th>
<th>KB</th>
</tr>
</thead>
<tbody>
<tr>
<td>(I_1)</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>(I_2)</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td>(I_3)</td>
<td>T</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>(I_4)</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>T</td>
<td>T</td>
<td>T</td>
</tr>
<tr>
<td>(I_5)</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>F</td>
</tr>
</tbody>
</table>
What We Want to Do with Logic

1) Tell the system **knowledge** about a task domain.
   - This is your KB
   - which expresses **true statements** about the world

2) **Ask the system** whether new statements about the domain are true or false.
   - We want the system responses to be
     - **Sound**: only generates correct answers with respect to the semantics
     - **Complete**: Guaranteed to find an answer if it exists
For Instance

1) Tell the system **knowledge** about a task domain.

   light_l1.
   light_l2.
   ok_l1.
   ok_l2.
   ok_cb1.
   ok_cb2.
   live_outside

   live_l1 ← live_w0.
   live_w0 ← live_w1 ∧ up_s2.
   live_w0 ← live_w2 ∧ down_s2.
   live_w1 ← live_w3 ∧ up_s1.
   live_w2 ← live_w3 ∧ down_s1.
   live_l2 ← live_w4.
   live_w4 ← live_w3 ∧ up_s3.
   live_p1 ← live_w3.
   live_w3 ← live_w5 ∧ ok_cb1.
   live_p2 ← live_w6.
   live_w6 ← live_w5 ∧ ok_cb2.
   live_w5 ← live_outside.
   lit_l1 ← light_l1 ∧ live_l1 ∧ ok_l1.
   lit_l2 ← light_l2 ∧ live_l2 ∧ ok_l2.

2) Ask the system whether new statements about the domain are true or false

   • live_w4?
   • lit_l2?
PDCL Semantics: Logical Consequence

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

• In other words, KB ⊨ g if there is no interpretation in which KB is true and g is false

• We want a reasoning procedure that can find all and only the logical consequences of a knowledge base
  • mechanically derivable demonstration that a formula logically follows from a knowledge base.
  • Must be sound and complete
Recap: proofs, soundness, completeness

- A proof is a mechanically derivable demonstration that a formula logically follows from a knowledge base.

**Definition (derivability with a proof procedure)**
Given a proof procedure $P$, $KB \vdash_P g$ means $g$ can be derived from knowledge base $KB$ with proof procedure $P$.

**Definition (soundness)**
A proof procedure $P$ is **sound** if $KB \vdash_P g$ implies $KB \models g$.

sound: every atom $g$ that $P$ derives follows logically from $KB$

**Definition (completeness)**
A proof procedure $P$ is **complete** if $KB \models g$ implies $KB \vdash_P g$.

complete: every atom $g$ that logically follows from $KB$ is derived by $P$
Simple Proof Procedure

Simple proof procedure $S$

- Enumerate all interpretations
- For each interpretation $I$, check whether it is a model of $KB$
  - i.e., check whether all clauses in $KB$ are true in $I$
- $KB ⊢_S g$ if $g$ holds in all such models

*problem with this approach?*
Simple Proof Procedure

Simple proof procedure S

• Enumerate all interpretations
• For each interpretation I, check whether it is a model of KB
  ✓ i.e., check whether all clauses in KB are true in I
• KB ⊨ g if g holds in all such models

**Problem with this approach?**

• If there are n propositions in the KB, must check all the interpretations!
Simple Proof Procedure

Simple proof procedure $S$

- Enumerate all interpretations
- For each interpretation $I$, check whether it is a model of $KB$
  - i.e., check whether all clauses in $KB$ are true in $I$
- $KB \vdash_S g$ if $g$ holds in all such models

**problem with this approach?**

- If there are $n$ propositions in the $KB$, must check all the $2^n$ interpretations!

Goal of proof theory

- find sound and complete proof procedures that allow us to prove that a logical formula follows from a $KB$ avoiding to do the above
Lecture Overview

• Recap

→ Bottom-Up Proof Procedure
  • Soundness
  • Completeness

• Top-Down (TD) Proof Procedure
• TD as Search
• Datalog (time permitting)
One rule of derivation, a generalized form of modus ponens:

- If "h ← b_1 ∧ ... ∧ b_m" is a clause in the knowledge base, and each b_i has been derived, then h can be derived.

This rule also covers the case when m = 0.
Bottom-up (BU) proof procedure

\[ C := \{\}; \]
repeat
    \textbf{select} clause “\( h \leftarrow b_1 \land \ldots \land b_m \)” in KB such that \( b_i \in C \) for all \( i \), and \( h \notin C \);
    \[ C := C \cup \{ h \} \]
until no more clauses can be selected.

\( KB \vdash_{BU} G \) if \( G \subseteq C \) at the end of this procedure

The C at the end of BU procedure is a \textbf{fixed point}:

- Further applications of the rule of derivation will not change C!
Bottom-up proof procedure: example

\[
C := \emptyset;
\]
\[
\text{repeat}
\]
\[
\quad \text{select} \ \text{clause } h \leftarrow b_1 \wedge \ldots \wedge b_m \text{ in KB}
\]
\[
\quad \text{such that } b_i \in C \text{ for all } i, \text{ and } h \not\in C;
\]
\[
\quad C := C \cup \{h\}
\]
\[
\text{until no more clauses can be selected.}
\]

\[
a \leftarrow b \wedge c
\]
\[
a \leftarrow e \wedge f
\]
\[
b \leftarrow f \wedge k
\]
\[
c \leftarrow e
\]
\[
d \leftarrow k
\]
\[
e.
\]
\[
f \leftarrow j \wedge e
\]
\[
f \leftarrow c
\]
\[
j \leftarrow c
\]
Bottom-up proof procedure: example

C := {};
repeat
    select clause h ← b_1 ∧ ... ∧ b_m in KB
    such that b_i ∈ C for all i, and h ∉ C;
    C := C ∪ {h}
until no more clauses can be selected.

\( a ← b ∧ c \)
\( a ← e ∧ f \)
\( b ← f ∧ k \)
\( c ← e \)
\( d ← k \)
\( e. \)
\( f ← j ∧ e \)
\( f ← c \)
\( j ← c \)

\{\}
\{e\}
\{c,e\}
\{c,e,f\}
\{c,e,f,j\}
\{a,c,e,f,j\}

Done.
Bottom-up proof procedure: Example

KB

\[
\begin{align*}
z & \leftarrow f \land e \\
q & \leftarrow r \land g \land e \\
e & \leftarrow a \land b \\
a & \\
b & \\
r & \\
f & 
\end{align*}
\]

Which of the following is true?

A. \( KB \vdash_{BU} \{z, q, a\} \)
B. \( KB \vdash_{BU} \{r, z, b\} \)
C. \( KB \vdash_{BU} \{q, a\} \)
# Bottom-up proof procedure: Example

**KB**

\[
\begin{align*}
z & \leftarrow f \land e \\
q & \leftarrow r \land g \land e \\
e & \leftarrow a \land b \\
a & \\
b & \\
r & \\
f &
\end{align*}
\]

C := \{\};
repeat
    select clause \( h \leftarrow b_1 \land \ldots \land b_m \) in KB such that \( b_i \in C \) for all \( i \), and \( h \notin C \);
    \( C := C \cup \{h\} \)
until no more clauses can be selected.

\[\{a\}\]
\[\{a, b\}\]
\[\{a, b, r\}\]
\[\{a, b, r, f\}\]
\[\{a, b, r, f, e\}\]
\[\{a, b, r, f, e, z\}\]

**A.** \( KB \vdash_{BU} \{z, q, a\} \)

**B.** \( KB \vdash_{BU} \{r, f, b\} \)

**C.** \( KB \vdash_{BU} \{q, a\} \)

Done.
Lecture Overview

• Recap
• Bottom-Up Proof Procedure
  Soundness
  • Completeness
• Top-Down Proof Procedure
• TD as Search
• Datalog (time permitting)
Soundness of bottom-up proof procedure BU

Definition (soundness)
A proof procedure $P$ is sound if $\text{KB} \vdash_P g$ implies $\text{KB} \models g$.

sound: every atom $g$ that $P$ derives follows logically from KB

\begin{verbatim}
C := {}; 
repeat
  select clause $h \leftarrow b_1 \land \ldots \land b_m$ in KB 
  such that $b_i \in C$ for all $i$, and $h \not\in C$;
  C := C $\cup$ \{h\}
until no more clauses can be selected.
\end{verbatim}

What do we need to prove to show that BU is sound?
Soundness of bottom-up proof procedure BU

Definition (soundness)
A proof procedure P is sound if $\text{KB} \vdash_P g$ implies $\text{KB} \models g$.

sound: every atom $g$ that P derives follows logically from KB

C := {};
repeat
    select clause $h \leftarrow b_1 \land \ldots \land b_m$ in KB
    such that $b_i \in C$ for all i, and $h \not\in C$;
    C := C ∪ {h}
until no more clauses can be selected.

What do we need to prove to show that BU is sound?

If $g \in C$ at the end of BU procedure, then $g$ is true in all models of KB ($\text{KB} \models g$)
Soundness of bottom-up proof procedure BU

Proof by contradiction: Suppose there is a \( h \) such that

\[
KB \vdash_{BU} h \quad \text{but not} \quad KB \not\models h.
\]

1. Let \( h \) be the first atom added to \( C \) that is not true in every model of \( KB \).
   - In particular, suppose \( I \) is a model of \( KB \) in which \( h \) isn’t true.

2. Since \( h \) was added to \( C \), there must be a clause in \( KB \) of form

\[
h \leftarrow b_1 \land \ldots \land b_n
\]

where each \( b_j \) is already in \( C \) and thus true in every model of \( KB \), including \( I \).

3. Because \( h \) is false in \( I \), \( h \leftarrow b_1 \land \ldots \land b_n \) is false in \( I \).

4. Therefore \( I \) is not a model of \( KB \) \( \Rightarrow \) Contradiction with
Lecture Overview

• Recap
• Bottom-Up Proof Procedure
  • Soundness
  • Completeness
• Top-Down Proof Procedure
• TD as Search
• Datalog (time permitting)
Completeness of BU: general idea

• Generic completeness of proof procedure:
  \[
  \text{If } g \text{ is logically entailed by the KB } (KB \models g) \text{ then } g \text{ can be proved by the procedure } (KB \vdash_{BU} g)
  \]

Sketch of our proof for BU:
1. Suppose \( KB \not\models g \). Then \( g \) is true in all models of \( KB \).
2. Thus \( g \) is true in any particular model of \( KB \).
3. We will define a model (called \textit{minimal model}) so that if \( g \) is true in that model, \( g \) is proved by the bottom up algorithm.
4. Thus \( KB \vdash_{BU} g \).
We define a specific interpretation of our KB, in which
• every atom in C at the end of BU is true
• every other atom is false

This is called minimal model

EXAMPLE

All atoms = \{a, b, c, d, e, f, g\}

C = ?

A. \{a, b, c, f,\}
B. \{e, d, c, f,\}
C. \{a, c, d, e\}
D. \{e, d, c, b\}

KB
\[
\begin{align*}
a & \leftarrow e \land g. \\
b & \leftarrow f \land g. \\
c & \leftarrow e. \\
f & \leftarrow c \\
e. \\
d.
\end{align*}
\]
Completeness of BU: general idea

We define a specific interpretation of our KB, in which
- every atom in C at the end of BU is true
- every other atom is false

This is called minimal model

EXAMPLE

All atoms = \{a, b, c, d, e, f, g\}
C = ?
\{e, d, c, f,\} \quad \{e\}
\{e,d\}
\{e,d,c\}
\{e,d,c, f\}

\textbf{KB}
\begin{align*}
a & \leftarrow e \land g. \\
b & \leftarrow f \land g. \\
c & \leftarrow e. \\
f & \leftarrow c \\
e. \\
d. \\
\end{align*}
We define a specific model of our KB, in which

- every atom in C at the end of BU is true
- every other atom is false

This is called **minimal model**

All atoms = \{a, b, c, d, e, f, g\}

C = ?

\{e, d, c, f,\}  \{e\}  \{e,d\}  \{e,d,c\}  \{e,d,c, f\}

**KB**

\[a \leftarrow e \land g.\]
\[b \leftarrow f \land g.\]
\[c \leftarrow e.\]
\[f \leftarrow c\]
\[e.\]
\[d.\]
We define a specific *interpretation* of our KB, in which
- every atom in C at the end of BU is true
- every other atom is false

This is called **minimal model**

All atoms = \{a, b, c, d, e, f, g\}
C = \{e, d, c, f,\}
Minimal Model =

\[
\begin{align*}
    a & \leftarrow e \land g. \\
    b & \leftarrow f \land g. \\
    c & \leftarrow e. \\
    f & \leftarrow c \\
    e. \\
    d.
\end{align*}
\]
We define a specific *interpretation* of our KB, in which
• every atom in C at the end of BU is true
• every other atom is false

This is called **minimal model**

All atoms = \{a, b, c, d, e, f, g\}
C = \{e, d, c, f,\}
Minimal Model =
da=F, b=F, c=T, d = T, e=T, f=T, g=F
Completeness of BU: general idea

We define a specific interpretation of our KB, in which
- every atom in C at the end of BU is true
- every other atom is false

This is called minimal model

All atoms = \{a, b, c, d, e, f, g\}

C = \{e, d, c, f,\}

Minimal Model =

\begin{align*}
a &= \text{F}, \\
b &= \text{F}, \\
c &= \text{T}, \\
d &= \text{T}, \\
e &= \text{T}, \\
f &= \text{T}, \\
g &= \text{F}.
\end{align*}

Using this interpretation, we'll then show that, if \( \text{KB} \models \not g \), then \( g \) must be in C, that is

If \( g \) is true in all models of KB (KB \( \not \models g \))
then \( g \in C \) at the end of BU procedure (KB \( \models \text{BU} g \))
First, we prove that: **MM is a model of KB**

Proof by contradiction: assume that MM is not a model of KB.

- Then there must exist some clause in KB which is false in MM.
  - Like every clause in KB, it is of the form $h \leftarrow b_1 \land \ldots \land b_m$ (with $m \geq 0$).
  - $h \leftarrow b_1 \land \ldots \land b_m$ can only be false in MM if each $b_i$ is true in MM and $h$ is false in MM.
    - Since each $b_i$ is true in MM, each $b_i$ must be in C as well.
    - BU would add $h$ to C, so $h$ would be true in MM. **Contradiction!**
- Thus, MM is a model of KB

---

**Definition**

The **minimal model MM** is the interpretation in which
- every element of BU’s fixed point C is true
- every other atom is false.
Completeness of bottom-up procedure

If $g$ is true in all models of $KB$ ($KB \models g$) then $g \in C$ at the end of BU procedure ($KB \models_{BU} g$)

Direct proof based on minimal model:
- Suppose $KB \not\models g$. Then $g$ is true in all models of $KB$.
- Thus $g$ is true in the minimal model.
- Thus $g \in C$ at the end of BU procedure.
- Thus $KB \models_{BU} g$. Done. $KB \not\models g$ implies $KB \models_{BU} g$
Summary for bottom-up proof procedure BU

- 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

- And, it is efficient!
  - Linear in the number of clauses in KB
    ✓ Each clause is used maximally once by BU
Let’s consider these two alternative proof procedures for PDCL....

X. \[ C_X = \{ \text{All clauses in KB with empty bodies} \} \]

Y. \[ C_Y = \{ \text{All atoms in the knowledge base} \} \]

A. Both X and Y are sound and complete

B. Both Y and X are neither sound nor complete

C. X is sound only and Y is complete only

D. X is complete only and Y is sound only
Let’s consider these two alternative proof procedures for PDCL....

X. \(C_X = \{\text{All clauses in KB with empty bodies}\}\)

Returns atoms that are indeed logical consequences (sound), but misses all those derived from the application of rules with non-empty bodies (not complete)

Y. \(C_Y = \{\text{All atoms in the knowledge base}\}\)

Returns all the logical consequences (complete), but also returns atoms that are not (not sound),

\[KB\]
\[a \leftarrow e \land g.\]
\[b \leftarrow f \land g.\]
\[c \leftarrow e.\]
\[f \leftarrow c\]
\[e.\]
\[d.\]

X is sound only and Y is complete only
Let $g$ be the query

**Bottom-up**

- $KB$ → $C$

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

When does BU use the information that $G$ is the query?
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**

$KB \rightarrow C$

Query $g$ is proven if $g \in C$

When does BU uses the info that $g$ is the query?

- Only at the end
- It derives the same $C$ regardless of the query

**Top-down**

Query $g$

$KB \rightarrow$ answer

TD performs a backward search starting at $g$
Lecture Overview

• Recap
• Bottom-Up Proof Procedure
  • Soundness
  • Completeness

Top-Down (TD) Proof Procedure
• TD as Search
• Datalog (time permitting)
Top-down Proof Procedure for PDCL

- Idea: search **backward** from a query to determine if it is a logical consequence of $KB$.
- An **answer clause** is of the form
  
  $$
  yes \leftarrow a_1 \land \ldots \land a_{i-1} \land a_i \land a_{i+1} \land \ldots \land a_m
  $$

- We express a query $q_1 \land q_2 \land \ldots \land q_m$ as an answer clause
  
  $$
  yes \leftarrow q_1 \land \ldots \land q_k
  $$

- Basic operation: **SLD Resolution** of an answer clause
  
  $$
  yes \leftarrow a_1 \land \ldots \land a_{i-1} \land a_i \land a_{i+1} \land \ldots \land a_m
  $$
  
  on atom $a_i$ with the clause:

  $$
  a_i \leftarrow b_1 \land \ldots \land b_p
  $$

  yields the clause

  $$
  yes \leftarrow a_1 \land \ldots \land a_{i-1} \land b_1 \land \ldots \land b_p \land a_{i+1} \land \ldots \land a_m
  $$
Example

- Rule of derivation: the SLD Resolution of clause

\[ yes \leftarrow a_1 \land ... \land a_{i-1} \land a_i \land a_{i+1} ... \land a_m \]

on atom \( a_i \) with the clause:

\[ a_i \leftarrow b_1 \land ... \land b_p \]

is the answer clause

\[ yes \leftarrow a_1 \land ... \land a_{i-1} \land b_1 \land ... \land b_p \land a_{i+1} ... \land a_m \]

Example

\[ yes \leftarrow b \land c. \]

\[ b \leftarrow k \land f. \]

SLD resolution

\[ yes \leftarrow k \land f \land c \]

\[ yes \leftarrow e \land f. \]

\[ e. \]

\[ yes \leftarrow f \]
Derivations

• An answer is an answer clause with $m = 0$.
  \[ \text{yes} \leftarrow . \]

• A successful derivation from $KB$ of query
  \[ ? \ q_1 \land ... \land q_k \]
  is a sequence of answer clauses $\gamma_0, \gamma_1, .., \gamma_n$ such that
  \begin{itemize}
  \item $\gamma_0$ is the answer clause \[ \text{yes} \leftarrow q_1 \land ... \land q_k. \]
  \item $\gamma_i$ is obtained by resolving $\gamma_{i-1}$ with a clause in $KB$, and
  \item $\gamma_n$ is an answer. \[ \text{yes} \leftarrow . \]
  \end{itemize}

• An unsuccessful derivation from $KB$ of query \[ ? \ q_1 \land ... \land q_k \]
  \begin{itemize}
  \item We get to something like \[ \text{yes} \leftarrow b_1 \land ... \land b_k. \]
  \item There is no clause in $KB$ with any of the $b_i$ as its head
  \end{itemize}
To solve the query \( \mathbf{q_1 \wedge \ldots \wedge q_k} \):

- **ac**: yes \(\leftarrow\) body, where body is \(\mathbf{q_1 \wedge \ldots \wedge q_k}\)

**repeat**

- **select** \(q_i \in\) body;
- **choose** clause \(Cl \in KB\), \(Cl\) is \(q_i \leftarrow b_c\);
- **replace** \(q_i\) in body by \(b_c\)

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

**select**: any choice will work

**choose**: have to pick the right one
Example: successful derivation

\[ a \leftarrow b \land c. \]
\[ \gamma_0: \text{yes} \leftarrow a \]

\[ c \leftarrow e. \]
\[ \gamma_1: \text{yes} \leftarrow e \land f \]

\[ f \leftarrow j \land e. \]
\[ \gamma_2: \text{yes} \leftarrow e \land c \]

\[ 1 \quad a \leftarrow e \land f. \]
\[ \gamma_3: \text{yes} \leftarrow c \]

\[ 2 \quad f \leftarrow c. \]
\[ \gamma_4: \text{yes} \leftarrow e \]

\[ b \leftarrow f \land k. \]
\[ \gamma_5: \text{yes} \leftarrow c. \]

\[ 3 \quad d \leftarrow k \]
\[ 4 \quad c \leftarrow e. \]
\[ 5 \quad e. \]

Query: ?a

\[ \gamma_0: \text{yes} \leftarrow a \]
\[ \gamma_1: \text{yes} \leftarrow e \land f \]
\[ \gamma_2: \text{yes} \leftarrow e \land c \]
\[ \gamma_3: \text{yes} \leftarrow c \]
\[ \gamma_4: \text{yes} \leftarrow e \]
\[ \gamma_5: \text{yes} \leftarrow c. \]

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

The answer is “Yes, we can.”
Example: failing derivation

Query: ?a

\[ \begin{align*}
\gamma_0: & \quad \text{yes} \leftarrow a \\
\gamma_1: & \quad \text{yes} \leftarrow b \land c \\
\gamma_2: & \quad \text{yes} \leftarrow f \land k \land c \\
\gamma_3: & \quad \text{yes} \leftarrow c \land k \land c \\
\gamma_4: & \quad \text{yes} \leftarrow e \land k \land c \\
\gamma_5: & \quad \text{yes} \leftarrow k \land c \\
\gamma_6: & \quad \text{yes} \leftarrow k \land e \\
\gamma_7: & \quad \text{yes} \leftarrow k
\end{align*} \]

There is no rule with k as its head, thus ... fail
Rules of derivation in top-down and bottom-up

Top-down:

SLD Resolution

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

Bottom-up:

Generalized modus ponens

\[
\begin{align*}
b & \leftarrow b_1 \land \ldots \land b_m \\
\text{h} & \leftarrow b_1 \land \ldots \land b_m
\end{align*}
\]
Lecture Overview

- Recap
- Bottom-Up Proof Procedure
  - Soundness
  - Completeness
- Top-Down (TD) Proof Procedure
  TD as Search
- Datalog (time permitting)
SLD resolution as search

- SLD resolution can be seen as a search
  - from a query stated as an answer clause
  - to an answer
- Through the space of all possible answer clauses
### Where Are We?

#### Environment
- **Deterministic**
  - Arc Consistency
- **Stochastic**
  - Value Iteration

#### Problem Type
- **Static**
  - Constraint Satisfaction
  - Logics
    - STRIPS
  - Belief Nets
    - Variable Elimination
- **Sequential**
  - Planning
  - Search
  - Decision Nets
    - Variable Elimination
  - Markov Processes
    - Value Iteration

#### Reasoning Technique
- Search
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”

• Query (Top-down/SLD resolution)
  • **State**: answer clause of the form \( \text{yes} \leftarrow a_1 \land \ldots \land a_k \)
  • **Successor function**: state resulting from substituting first atom \( a_1 \) with \( b_1 \land \ldots \land b_m \) if there is a clause \( a_1 \leftarrow b_1 \land \ldots \land b_m \)
  • **Goal test**: is the answer clause empty (i.e. \( \text{yes} \leftarrow \)) ?
  • **Solution**: the proof, i.e. the sequence of SLD resolutions
  • **Heuristic function**: ??????
KB

\[ a \leftarrow b \land c. \]
\[ a \leftarrow h. \]
\[ b \leftarrow k. \]
\[ d \leftarrow p. \]
\[ f \leftarrow p. \]
\[ g \leftarrow f. \]
\[ h \leftarrow m. \]

\[ b \leftarrow j. \]
\[ d \leftarrow m. \]
\[ f \leftarrow m. \]
\[ g \leftarrow m. \]
\[ k \leftarrow m. \]

Prove: \[ ? \leftarrow a \land d. \]

Why are these dead ends in the search space?
KB

Prove: ? ← a ∧ d.

Why are these dead ends in the search space?

- the successor function resolves the first atom in the body of the answer clause
- But j and m cannot be resolved
Learning Goals For Logic so Far

• 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
  - Define it as a search problem