Chapters 5: Propositions and Inference

DIT410/TIN172 Artificial Intelligence

Peter Ljunglöf
modifed from slides by Poole & Mackworth

1 April, 2015

Outline

1. *Propositions (5.1–5.2)*
2. *Proofs (5.2.2)*
   - Bottom-up proof procedure (5.2.2.1)
   - Top-down proof procedure (5.2.2.2)
3. *Complete knowledge assumption (5.5)*

Outline

For when I am presented with a false theorem, I do not need to examine or even to know the demonstration, since I shall discover its falsity *a posteriori* by means of an easy experiment, that is, by a calculation, costing no more than paper and ink, which will show the error no matter how small it is...

And if someone would doubt my results, I should say to him: "Let us calculate, Sir," and thus by taking to pen and ink, we should soon settle the question.

—Gottfried Wilhelm Leibniz [1677]

Syntax of propositional calculus

Propositions are built from simpler propositions using logical connectives. A proposition is either:

- an atomic proposition (also called atom, symbol or a boolean variable),
- or a compound proposition of the form:

$$\begin{aligned}
\neg p &: \text{the negation of } p \\
p \wedge q &: \text{the conjunction of } p \text{ and } q \\
p \vee q &: \text{the disjunction of } p \text{ and } q \\
p \rightarrow q &: \text{the implication of } q \text{ from } p \\
p \leftrightarrow q &: \text{the equivalence of } p \text{ and } q
\end{aligned}$$

The precedence of the connectives is in the above order.

Interpretations, models and propositions

- An *interpretation* is an assignment of values to all symbols.
- A *model* is an interpretation that satisfies the constraints.
- Often we don't want to just find a model, but want to know what is true in all models.
- A *proposition* is statement that is true or false in each interpretation.
- Propositions are built using *logical connectives*.

Semantics of propositional calculus

An interpretation is a function π that maps atoms to $\{\mathit{true}, \mathit{false}\}$:

- if $\pi(a) = \mathit{true}$ (false), we say atom a is true (false) in the interpretation
- we can also think of π as the set of atoms that map to true

The interpretation maps each proposition to a truth value. The truth value of a compound proposition is built using this *truth table:*

p	q	$\neg q$	$p \wedge q$	$p \vee q$	$p \rightarrow q$	$p \leftrightarrow q$
true	*true*	*false*	*true*	*true*	*true*	*true*
true	*false*	*true*	*false*	*true*	*false*	*false*
false	*true*		*false*	*true*	*true*	*false*
false	*false*		*false*	*false*	*true*	*true*

Simple language: Propositional definite clauses

- An **atom** is a symbol starting with a lower case letter
- A **body** is of the form $b_1 \wedge \cdots \wedge b_k$ where $b_1 \ldots b_k$ are atoms
  - k can be 1, in which case the body is a single atom
- A **definite clause** is an atom or is an implication of the form $b \rightarrow h$ where b is a body and h is an atom
  - usually we write the clause backwards, $h \leftarrow b$
- A **knowledge base** is a set of definite clauses

Definite Clauses

Which of the following are definite clauses?

(a) $happy \leftarrow sad$

(b) $blimsy$

(c) $old \wedge wise \leftarrow teenager$

(d) $happy \wedge sad$

(e) $glad \leftarrow happy \wedge sad$

(f) $green \vee blue \leftarrow \neg red$

(g) $glad \leftarrow happy \wedge sad \wedge mad \wedge bad$

(h) $glad \leftarrow happy \wedge rad \leftarrow sad \wedge mad \wedge bad$

(i) $happy \leftarrow happy$

Human's view of semantics

Step 1 Begin with a task domain.

Step 2 Choose atoms in the computer to denote propositions.

- These atoms have meaning to the KB designer.

Step 3 Tell the system knowledge about the domain.

Step 4 Ask the system questions.

- The system will answer whether the question is a logical consequence.

Step 5 Interpret the answers with the meaning associated with the atoms.

Role of semantics

In computer:

$$light1\_broken \leftarrow sw\_up \wedge power \wedge unlit\_light1.$$

$$sw\_up.$$

$$power \leftarrow lit\_light2.$$

$$unlit\_light1.$$

$$lit\_light2.$$

In user's mind:

- $light1\_broken$: light #1 is broken
- sw_up: switch is up
- $power$: there is power in the building
- $unlit\_light1$: light #1 isn't lit
- lit_light2: light #2 is lit

Conclusion: $light1\_broken$

- The computer doesn't know the meaning of the symbols
- The user can interpret the symbol using their meaning

Semantics

Simplified semantics for definite clauses:

- An interpretation I assigns a truth value to each atom.
- A body $b_1 \wedge \ldots \wedge b_k$ is *true* in I if all b_i's are *true* in I, and is *false* otherwise.
- A rule $h \leftarrow b$ is *false* in I if b is *true* in I and h is *false* in I. The rule is *true* otherwise.
- A knowledge base KB is *true* in I if and only if every clause in KB is *true* in I.

Models and logical consequence

- A **model** of a set of clauses is an interpretation in which all the clauses are *true*.
- Assuming that KB is a set of clauses and g is a body:
  - g is a **logical consequence** of KB, written $KB \models g$, if g is *true* in every model of KB.
- That is, $KB \models g$ if there is no interpretation in which KB is *true* and g is *false*.

Simple example

$$KB \quad = \quad \begin{cases} p \leftarrow q. \\ q. \\ r \leftarrow s. \end{cases}$$

	p	q	r	s	model?
I_1	*true*	*true*	*true*	*true*	
I_2	*false*	*false*	*false*	*false*	
I_3	*true*	*true*	*false*	*false*	
I_4	*true*	*true*	*true*	*false*	
I_5	*true*	*true*	*false*	*true*	

Which of p, q, r, s, t logically follow from KB?

Simple example

$$KB = \begin{cases} p \leftarrow q. \\ q. \\ r \leftarrow s. \end{cases}$$

	p	q	r	s	model?
I_1	*true*	*true*	*true*	*true*	is a model of KB
I_2	*false*	*false*	*false*	*false*	not a model of KB
I_3	*true*	*true*	*false*	*false*	is a model of KB
I_4	*true*	*true*	*true*	*false*	is a model of KB
I_5	*true*	*true*	*false*	*true*	not a model of KB

Which of p, q, r, s, t logically follow from KB?

$KB \models p$, $KB \models q$, $KB \not\models r$, $KB \not\models s$, $KB \not\models t$

User's view of semantics

Step 1 Choose a task domain: intended interpretation.

Step 2 Associate an atom with each proposition you want to represent.

Step 3 Tell the system clauses that are true in the intended interpretation: axiomatizing the domain.

Step 4 Ask questions about the intended interpretation.

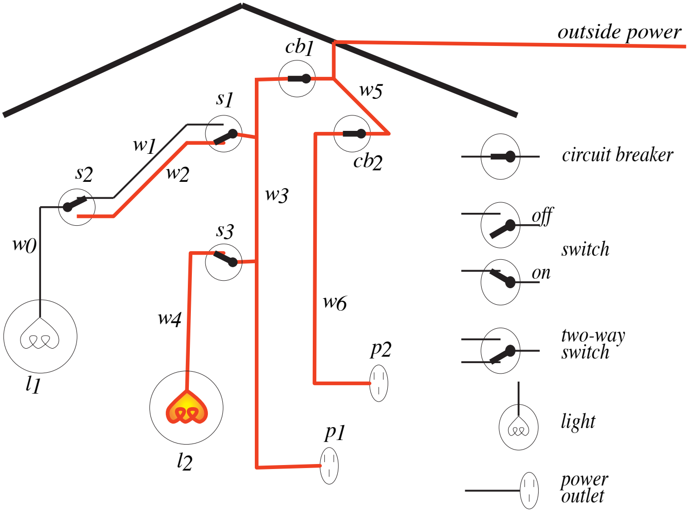
- If $KB \models g$, then g must be true in the intended interpretation.

Step 5 Users can interpret the answer using their intended interpretation of the symbols.

Computer's view of semantics

- The computer doesn't have access to the intended interpretation.
  - All it knows is the knowledge base KB.
- It can determine if a formula is a logical consequence of KB.
- If $KB \models g$ then g must be true in the intended interpretation.
- If $KB \not\models g$ then there is a model of KB in which g is false.
  - This could be the intended interpretation!

Example: Electrical environment



Representing the electrical environment

$light\_l_1.$

$light\_l_2.$

$down\_s_1.$

$up\_s_2.$

$up\_s_3.$

$ok\_l_1.$

$ok\_l_2.$

$ok\_cb_1.$

$ok\_cb_2.$

$live\_outside.$

$lit\_l_1 \leftarrow live\_w_0 \wedge ok\_l_1$

$live\_w_0 \leftarrow live\_w_1 \wedge up\_s_2.$

$live\_w_0 \leftarrow live\_w_2 \wedge down\_s_2.$

$live\_w_1 \leftarrow live\_w_3 \wedge up\_s_1.$

$live\_w_2 \leftarrow live\_w_3 \wedge down\_s_1.$

$lit\_l_2 \leftarrow live\_w_4 \wedge ok\_l_2.$

$live\_w_4 \leftarrow live\_w_3 \wedge up\_s_3.$

$live\_p_1 \leftarrow live\_w_3.$

$live\_w_3 \leftarrow live\_w_5 \wedge ok\_cb_1.$

$live\_p_2 \leftarrow live\_w_6.$

$live\_w_6 \leftarrow live\_w_5 \wedge ok\_cb_2.$

$live\_w_5 \leftarrow live\_outside.$

Outline

1. *Propositions (5.1–5.2)*
2. *Proofs (5.2.2)*
   - Bottom-up proof procedure (5.2.2.1)
   - Top-down proof procedure (5.2.2.2)
3. *Complete knowledge assumption (5.5)*

Proofs

- A proof is a mechanically derivable demonstration that a formula logically follows from a knowledge base.
- $KB \vdash g$ means that g can be derived from knowledge base KB, using a proof procedure.
- Recall that $KB \models g$ means g is true in all models of KB.
- A proof procedure is sound if $KB \vdash g$ implies $KB \models g$.
- A proof procedure is complete if $KB \models g$ implies $KB \vdash g$.

Bottom-up proof procedure

- There is one single rule of derivation,
  a generalized form of *modus ponens*:

  If "$h \leftarrow b_1 \wedge \ldots \wedge b_m$" is a clause in the knowledge base, and each b_i has been derived, then h can be derived.

- This is called forward chaining on the clause.
  (This rule also covers the case when $m = 0$.)

Bottom-up proof procedure

After this procedure, $KB \vdash g$ iff $g \in C$:

> $C := \{\}$
> **repeat**
>   **select** clause "$h \leftarrow b_1 \wedge \ldots \wedge b_m$" 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

Example

$$a \leftarrow b \wedge c.$$
$$a \leftarrow e \wedge f.$$
$$b \leftarrow f \wedge k.$$
$$c \leftarrow e.$$
$$d \leftarrow k.$$
$$e.$$
$$f \leftarrow j \wedge e.$$
$$f \leftarrow c.$$
$$j \leftarrow c.$$

Soundness of bottom-up proof procedure

Assume there is a g such that $KB \vdash g$ and $KB \not\models g$.

- Then there must be a *first* atom added to C that isn't true in every model of KB. Call it h.
  - Let's call this model I, in which h isn't *true*.
- There must be a clause in KB of form:

$$h \leftarrow b_1 \wedge \ldots \wedge b_m$$

  Each b_i is true in I, but h is false in I. So this clause is false in I. Therefore I isn't a model of KB.

- Contradiction.

Therefore, if $KB \vdash g$ then $KB \models g$.

Fixed point

- The C generated at the end of the bottom-up algorithm is called a fixed point.
- Let I be the interpretation in which every element of the fixed point is true and every other atom is false.
- I is a model of KB.

  Proof: Suppose $h \leftarrow b_1 \wedge \ldots \wedge b_m$ in KB is false in I.
  Then h is false and each b_i is true in I.
  Thus h can be added to C.
  This is a contradiction to C being the fixed point.
- I is called a minimal model.

Completeness

Suppose $KB \models g$.

- Then g is true in all models of KB.
- Thus g is true in the minimal model.
- Thus g is in the fixed point.
- Thus g is generated by the bottom up algorithm.
- Thus $KB \vdash g$.

Therefore, if $KB \models g$ then $KB \vdash g$.

Top-down definite clause proof procedure

A query is a body that we want to determine if it is a logical consequence of KB.

Idea: search backward from the query.

- An answer clause is of the form:

$$yes \leftarrow a_1 \wedge \cdots \wedge a_i \wedge \cdots \wedge a_m$$

- SLD resolution of this answer clause on atom a_i with the clause:

$$a_i \leftarrow b_1 \wedge \cdots \wedge b_p$$

  results in the answer clause:

$$yes \leftarrow a_1 \wedge \cdots \wedge a_{i-1} \wedge b_1 \wedge \cdots \wedge b_p \wedge a_{i+1} \wedge \cdots \wedge a_m$$

Derivations

- An **answer** is an answer clause with $m = 0$.
  - i.e., it is the answer clause "$yes \leftarrow$"
- A **derivation** of query "$?q_1 \wedge \ldots \wedge q_k$" from KB is a sequence of answer clauses $\gamma_0, \gamma_1, \ldots, \gamma_n$ such that
  - γ_0 is the answer clause "$yes \leftarrow q_1 \wedge \ldots \wedge q_k$",
  - γ_i is obtained by resolving γ_{i-1} with a clause in KB, and
  - γ_n is an answer.

Top-down definite clause interpreter

To solve the query $?q_1 \wedge \ldots \wedge q_k$:

> $ac :=$ "$yes \leftarrow q_1 \wedge \ldots \wedge q_k$"
> **repeat**
> > **select** atom a_i from the body of ac
> > **choose** clause C from KB with a_i as head
> > replace a_i in the body of ac by the body of C
>
> **until** ac is an answer

Nondeterministic choice

The top-down interpreter uses two kinds of nondeterministic choice:

don't-care If one selection doesn't lead to a solution, there is no point trying other alternatives.

- "**select** atom a_i from the body of ac"

don't-know If one choice doesn't lead to a solution, there might be other choices that will.

- "**choose** clause C from KB with a_i as head"

Example: Successful derivation

$a \leftarrow b \wedge c$	$a \leftarrow e \wedge f$	$b \leftarrow f \wedge k$
$c \leftarrow e$	$d \leftarrow k$	e
$f \leftarrow j \wedge e$	$f \leftarrow c$	$j \leftarrow c$

Query: $?a$

Example: Successful derivation

$$
\begin{array}{lll}
a \leftarrow b \wedge c & a \leftarrow e \wedge f & b \leftarrow f \wedge k \\
c \leftarrow e & d \leftarrow k & e \\
f \leftarrow j \wedge e & f \leftarrow c & j \leftarrow c
\end{array}
$$

Query: $?a$

$$
\begin{array}{lll}
\gamma_0 : & yes \leftarrow a & (a \leftarrow e \wedge f) \\
\gamma_1 : & yes \leftarrow e \wedge f & (e) \\
\gamma_2 : & yes \leftarrow f & (f \leftarrow c) \\
\gamma_3 : & yes \leftarrow c & (c \leftarrow e) \\
\gamma_4 : & yes \leftarrow e & (e) \\
\gamma_5 : & yes \leftarrow &
\end{array}
$$

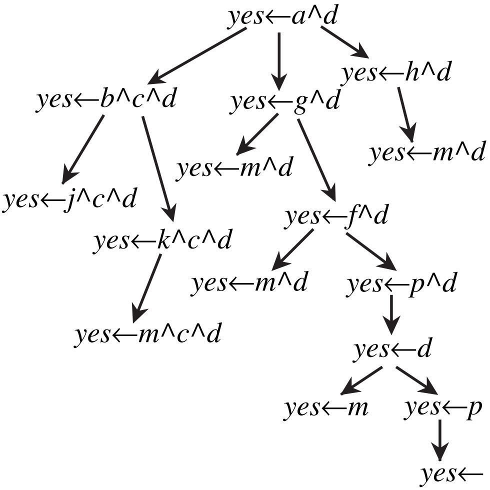
Example: Failing derivation

$$
\begin{array}{lll}
a \leftarrow b \wedge c & a \leftarrow e \wedge f & b \leftarrow f \wedge k \\
c \leftarrow e & d \leftarrow k & e \\
f \leftarrow j \wedge e & f \leftarrow c & j \leftarrow c
\end{array}
$$

Query: $?a$

$$
\begin{array}{lll}
\gamma_0: & yes \leftarrow a & (a \leftarrow b \wedge c) \\
\gamma_1: & yes \leftarrow b \wedge c & (b \leftarrow f \wedge k) \\
\gamma_2: & yes \leftarrow f \wedge k \wedge c & (f \leftarrow c) \\
\gamma_3: & yes \leftarrow c \wedge k \wedge c & (c \leftarrow e) \\
\gamma_4: & yes \leftarrow e \wedge k \wedge c & (e) \\
\gamma_5: & yes \leftarrow k \wedge c & fail
\end{array}
$$

Search graph for SLD resolution



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

Outline

1. *Propositions (5.1–5.2)*
2. *Proofs (5.2.2)*
   - Bottom-up proof procedure (5.2.2.1)
   - Top-down proof procedure (5.2.2.2)
3. *Complete knowledge assumption (5.5)*

Complete knowledge assumption

- Often you want to assume that your knowledge is complete.
  - *Example*: you can state what switches are up and the agent can assume that the other switches are down.
  - *Example*: assume that a database of what students are enrolled in a course is complete.
- The definite clause language is monotonic:
  - adding clauses can't invalidate a previous conclusion.
- Under the *complete knowledge assumption*, the system is non-monotonic:
  - adding clauses can invalidate a previous conclusion.

Completion of a knowledge base

- Suppose the rules for atom a are,

$$a \leftarrow b_1 \quad \ldots \quad a \leftarrow b_n$$

  or equivalently $a \leftarrow b_1 \vee \cdots \vee b_n$.

- The *complete knowledge assumption* (CKA) says that if$a$ is true, one of the b_i must be true:

$$a \rightarrow b_1 \vee \cdots \vee b_n$$

- Under the CKA, the meaning of the clauses for a are:

$$a \leftrightarrow b_1 \vee \cdots \vee b_n$$

  which is called Clark's completion.

Clark's completion of a KB

- Clark's completion of a knowledge base consists of the completion of every atom.
- If you have an atom a with no clauses, the completion is $a \leftrightarrow \mathit{false}$.
- You can interpret negations in the body of clauses:
  - $\sim a$ means that a is false under the CKA
  - this is negation as failure

Bottom-up negation-as-failure interpreter

$C := \{\}$
repeat
  either
    select h such that there is a rule "$h \leftarrow b_1 \wedge \ldots \wedge b_m$" $\in KB$
      where every $b_i \in C$, and $h \notin C$
    $C := C \cup \{h\}$
  or
    select h such that for every rule "$h \leftarrow b_1 \wedge \ldots \wedge b_m$" $\in KB$
      either $\sim b_i \in C$ for some b_i
      or $g \in C$ for some $b_i = \sim g$
    $C := C \cup \{\sim h\}$
until no more selections are possible

Negation-as-failure example

$$p \leftarrow q \wedge \sim r.$$

$$p \leftarrow s.$$

$$q \leftarrow \sim s.$$

$$r \leftarrow \sim t.$$

$$t.$$

$$s \leftarrow w.$$

Top-down negation-as-failure proof procedure

If the proof for a fails, you can conclude $\sim a$.

Failure can be defined recursively:

- Suppose you have rules for atom a:

$$a \leftarrow b_1 \quad \ldots \quad a \leftarrow b_n$$

- If each body b_i fails, a fails.
- A body fails if one of the conjuncts in the body fails.
- Note that you need *finite* failure.
  - example $p \leftarrow p$

Default reasoning

- Birds fly.
- Emus and tiny birds don't.
- Hummingbirds are tiny birds that can fly.

$$
\begin{aligned}
\mathit{flies} &\leftarrow \mathit{bird} \wedge {\sim} \mathit{ab}_{\mathit{flying}} \\
\mathit{ab}_{\mathit{flying}} &\leftarrow \mathit{emu} \wedge {\sim} \mathit{ab}_{\mathit{emu}} \\
\mathit{ab}_{\mathit{flying}} &\leftarrow \mathit{tiny} \wedge {\sim} \mathit{ab}_{\mathit{tiny}} \\
\mathit{ab}_{\mathit{tiny}} &\leftarrow \mathit{hummingbird} \wedge {\sim} \mathit{ab}_{\mathit{hummingbird}}
\end{aligned}
$$