Decision Procedures for Artificial Intelligence

INF656L

Alexandre Chapoutot and Sergio Mover

ENSTA Paris

2022-2023

Course Outline

Main goals of the course:

- Know the main principles behind logical agents in AI and System verification;
- To be able to model decision problems using logical formulas;
- Know the solving algorithms for SAT and SMT solvers.

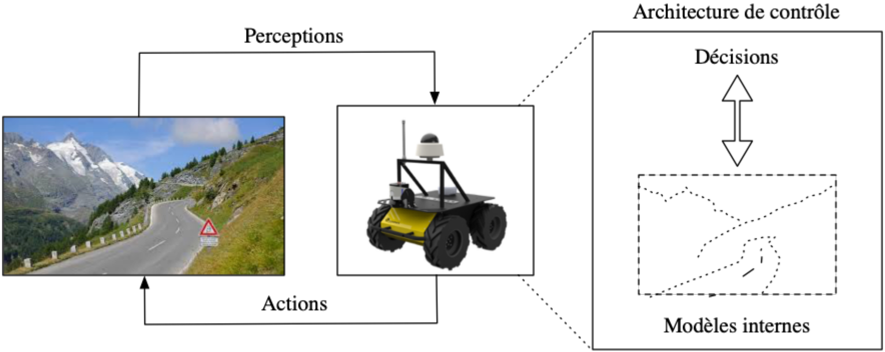
Remarks:

- consider unquantified logical formula
- consider exhaustive semantic methods

Intelligent agents

Examples

Mobile robot: Perception – Decision – Action
Software model agent: Belief – Desire – Intention



In this course

internal model is based on logical formula

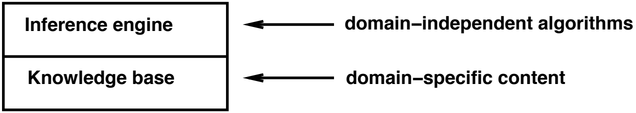
Motivations

We are interested in

- having an approach to *automate the reasoning*

in order to produce autonomous agent.

In particular, we will consider on *knowledge-based agent*. It is based on two elements:



- Knowledge base (set of sentences in a formal language) *i.e.* what the agent knows
- Inference engine *i.e.* a capability to deduce new information

Simple knowledge-based agent

```
function KB-AGENT(percept) returns an action
    static: KB, a knowledge base
            t, a counter, initially 0, indicating time

    TELL(KB, MAKE-PERCEPT-SENTENCE(percept, t))
    action ← ASK(KB, MAKE-ACTION-QUERY(t))
    TELL(KB, MAKE-ACTION-SENTENCE(action, t))
    t ← t + 1
    return action
```

The agent shall:

- Represent states (symbolic representation of the world) and actions
- Incorporate new perceptions
- Update internal representations of the world
- Deduce appropriate actions

Propositional logic will help to perform all these actions

Lecture 1: Propositional logic

1. Propositional logic
   - Syntax and Semantics
   - Satisfiability, Validity and Entailment
   - Normal forms
   - Tseitin's transform
   - Model checking
   - Theorem proving
   - Encoding

Syntax of PL

The Backus-Naur Form (BNF) of the grammar PL is:

$$P ::= \top \mid \bot \mid \text{id} \mid \neg P \mid P_1 \wedge P_2 \mid P_1 \vee P_2 \mid P_1 \implies P_2 \mid P_1 \iff P_2$$

We denote by $\mathcal{PL}$ the set of all well written PL formula.
Vocabulary

Atom is of two kinds

- **truth symbols** $\bot$, F, 0 (false) or $\top$, T, 1 (true)
- **ident** or **propositional variables** p, q, r, s, ... taken into a countable set $\mathcal{V}$

Literal is an atom α or its negation $\neg\alpha$

Formula are made of atom associated to **logical connectors**: $\neg$ (negation), $\wedge$ (conjunction), $\vee$ (disjunction), $\implies$ (implication), and $\iff$ (if and only if).

Syntax of PL – 2

We define the relative precedence of the logical operators from highest to lowest as follows

$$\neg, \wedge, \vee, \implies, \iff$$

Moreover, $\implies$ and $\iff$ are associative to the right, so

$$p \implies q \implies r \quad \equiv \quad p \implies (q \implies r)$$

Example

- $F : (p \wedge q \implies p \vee \neg q) \equiv (p \wedge q) \implies (p \vee (\neg q))$

Semantics of PL

An **interpretation** I is a (partial) mapping from $\mathcal{V}$ to truth value $\{0, 1\} = \mathbb{B}$.

Example

$$I : \{p \mapsto 0; q \mapsto 1; r \mapsto 1; \ldots\}$$

Definition

The semantics or the meaning of a PL formula F is its truth value with respect to a given interpretation I

We will denote by $[\![\cdot]\!]_{\text{PL}}$ the semantic function of PL formula, *i.e.*,

$$[\![\cdot]\!]_{\text{PL}} : \mathcal{PL} \to (\mathcal{V} \to \mathbb{B}) \to \mathbb{B}$$

Inductive definition of $[\![]\!]_{\mathrm{PL}}$

Following the structure of a PL formula F, the semantic function $[\![]\!]_{\mathrm{PL}}$ is defined as followed

- if $F \equiv \bot$ then $[\![F]\!]_{\mathrm{PL}}(I) = 0$ for all I
- if $F \equiv \top$ then $[\![F]\!]_{\mathrm{PL}}(I) = 1$ for all I
- if $F \equiv \mathrm{id} \in \mathcal{V}$ then $[\![F]\!]_{\mathrm{PL}}(I) = I(\mathrm{id})$
- if $F \equiv \neg F_1$ then $[\![F]\!]_{\mathrm{PL}}(I) = \text{not } [\![F_1]\!]_{\mathrm{PL}}(I)$
- if $F \equiv F_1 \wedge F_2$ then $[\![F]\!]_{\mathrm{PL}}(I) = [\![F_1]\!]_{\mathrm{PL}}(I) \text{ and } [\![F_2]\!]_{\mathrm{PL}}(I)$
- if $F \equiv F_1 \vee F_2$ then $[\![F]\!]_{\mathrm{PL}}(I) = [\![F_1]\!]_{\mathrm{PL}}(I) \text{ or } [\![F_2]\!]_{\mathrm{PL}}(I)$
- if $F \equiv F_1 \implies F_2$ then $[\![F]\!]_{\mathrm{PL}}(I) = [\![\neg F_1 \vee F_2]\!]_{\mathrm{PL}}(I)$
- if $F \equiv F_1 \iff F_2$ then $[\![F]\!]_{\mathrm{PL}}(I) = [\![(F_1 \implies F_2) \wedge (F_2 \implies F_1)]\!]_{\mathrm{PL}}(I)$

Remark there is a difference between syntax and semantics notations.

Semantics of Boolean operator

The semantics of PL formula is based on the semantics of Boolean operator. It is given by the **Truth Tables**
p, q are propositional variables

p	q	not p	p and q	p or q
0	0	1	0	0
0	1	1	0	1
1	1	0	1	1
1	0	0	0	1

And we use rules as

- $p \implies q \equiv \neg p \vee q$
- $p \iff q \equiv (p \implies q) \wedge (q \implies p)$

to define the Truth value of $\implies$ and $\iff$

Example

Inputs:

- PL formula $F : (p \implies q) \wedge p$
- Interpretation $I : \{p \mapsto 1; q \mapsto 0\}$

Output: Truth value

$$
\begin{aligned}
[\![F]\!]_{\mathrm{PL}}(I) &= [\![(p \implies q) \wedge p]\!]_{\mathrm{PL}}(I) \\
&= [\![(p \implies q)]\!]_{\mathrm{PL}}(I) \text{ and } [\![p]\!]_{\mathrm{PL}}(I) \\
&= [\![(\neg p \vee q)]\!]_{\mathrm{PL}}(I) \text{ and } 1 \\
&= ([\![\neg p]\!]_{\mathrm{PL}}(I) \text{ or } [\![q]\!]_{\mathrm{PL}}(I)) \text{ and } 1 \\
&= (\text{not } [\![p]\!]_{\mathrm{PL}}(I) \text{ or } 0) \text{ and } 1 \\
&= ((\text{not } 1) \text{ or } 0) \text{ and } 1 \\
&= (0 \text{ or } 0) \text{ and } 1 \\
&= 0 \text{ and } 1 \\
&= 0
\end{aligned}
$$

Satisfiability, Validity

Definition: Satisfiability

A PL formula F is **satisfiable** iff $\exists I$ such that $[\![F]\!]_{\mathrm{PL}}(I) = 1$. F is **unsatisfiable** iff $\forall I,\ [\![F]\!]_{\mathrm{PL}}(I) = 0$ or $\nexists I,\ [\![F]\!]_{\mathrm{PL}}(I) = 1$

Definition: Validity

F is **valid** or a **theorem**, iff $\forall I,\ [\![F]\!]_{\mathrm{PL}}(I) = 1$

A strong link between validity and satisfiability

$$F \text{ is valid iff } \neg F \text{ is unsatisfiable}$$

Definition: Equivalence

F_1 and F_2 are equivalent, *i.e.*, $F_1 \equiv F_2$
iff $\forall I,\ [\![F_1]\!]_{\mathrm{PL}}(I) = [\![F_2]\!]_{\mathrm{PL}}(I)$

Entailment or Logical consequence

We denote by $\mathcal{M}(a)$ the set of all models of a *i.e.*

$$\mathcal{M}(a) = \{I : [\![a]\!]_{\mathrm{PL}}(I) = 1\}$$

We can express logical consequence or entailment denoted by

$$a \models b \qquad \text{iff} \qquad \mathcal{M}(a) \subseteq \mathcal{M}(b)$$

In other words, a entails b iff b is true in all interpretations which make a true.

Example

In arithmetic, we have

$$x = 0 \models xy = 0$$

For Knowledge-Based Agent

What we want is KB $\models g$ (g is goal for a robot)

Entailment as satisfiability

For any PL formula a and b, to show

$$a \models b$$

we can

- from a validity point of view

$$a \models b \quad \text{iff} \quad (a \implies b) \text{ is valid}$$

- from an unsatisfiability point of view (proof by contradiction)

$$a \models b \quad \text{iff} \quad (a \wedge \neg b) \text{ is unsatisfiable}$$

We will see two simple procedures: **model checking** and **resolution method**.

Vocabulary

A Clause

is a disjunction of literals, *e.g.*,

$$K \equiv a \vee \neg b \vee c \vee d$$

which is also denoted by $K = \{a, \neg b, c, d\}$

A cube

is a conjunction of literals, *e.g.*,

$$C \equiv \ell_1 \wedge \ell_2 \wedge \cdots \wedge \ell_n$$

also denoted by $C = \{\ell_1, \ell_2, \ldots, \ell_n\}$

Normal forms – NNF

Negative Normal Form (NNF)

Negation connectors only appear in front of literals and $\neg$, $\vee$, and $\wedge$ are the only available connectors.

Construction rules:

- $\neg\neg F \rightsquigarrow F, \quad \neg\top \rightsquigarrow \bot, \quad \neg\bot \rightsquigarrow \top$
- $\neg(F_1 \vee F_2) \rightsquigarrow \neg F_1 \wedge \neg F_2$
- $\neg(F_1 \wedge F_2) \rightsquigarrow \neg F_1 \vee \neg F_2$
- $F_1 \implies F_2 \rightsquigarrow \neg F_1 \vee F_2$
- $F_1 \iff F_2 \rightsquigarrow (F_1 \implies F_2) \wedge (F_2 \implies F_1)$

Example of NNF transformation

Transformation of $F : \neg(P \implies \neg(P \wedge Q))$ in NNF

- $F' : \neg(\neg P \vee \neg(P \wedge Q))$
- $F'' : \neg\neg P \wedge \neg\neg(P \wedge Q)$
- $F''' : P \wedge P \wedge Q$

Hence F''' is equivalent to F but in NNF

Normal forms – DNF

Disjunctive normal form (DNF)

Disjunction of conjunctions of literals (or disjunction of cubes)

$$\bigvee_i \bigwedge_j \ell_{ij} \quad \text{forall literals} \quad \ell_{ij}$$

Constructions rules:

- Transform F into NNF
- $(F_1 \vee F_2) \wedge F_3 \rightsquigarrow (F_1 \wedge F_3) \vee (F_2 \wedge F_3)$
- $F_1 \wedge (F_2 \vee F_3) \rightsquigarrow (F_1 \wedge F_2) \vee (F_1 \wedge F_3)$

Example of DNF transformation

Transformation of $F : (Q_1 \vee \neg\neg Q_2) \wedge (\neg R_1 \implies R_2)$ in DNF

- $F' : (Q_1 \vee Q_2) \wedge (R_1 \vee R_2)$
- $F'' : (Q_1 \wedge (R_1 \vee R_2)) \vee (Q_2 \wedge (R_1 \vee R_2))$
- $F''' : (Q_1 \wedge R_1) \vee (Q_1 \wedge R_2) \vee (Q_2 \wedge R_1) \vee (Q_2 \wedge R_2)$

Hence F''' is equivalent to F but in DNF

Normal forms – CNF

Conjunctive normal form (CNF)

Conjunction of disjunctions of literals (or conjunction of clauses)

$$\bigwedge_i \bigvee_j \ell_{ij} \quad \text{forall literals} \quad \ell_{ij}$$

Constructions rules:

- Transform F into NNF
- $(F_1 \wedge F_2) \vee F_3 \rightsquigarrow (F_1 \vee F_3) \wedge (F_2 \vee F_3)$
- $F_1 \vee (F_2 \wedge F_3) \rightsquigarrow (F_1 \vee F_2) \wedge (F_1 \vee F_3)$

Example of CNF transformation

Transformation of $F : (Q_1 \wedge \neg\neg Q_2) \vee (\neg R_1 \implies R_2)$ in CNF

- $F' : (Q_1 \wedge Q_2) \vee (R_1 \vee R_2)$
- $F'' : (Q_1 \vee (R_1 \vee R_2)) \wedge (Q_2 \vee (R_1 \vee R_2))$

Hence F'' is equivalent to F but in CNF

Normal form and Tseitin

Regular rules (de Morgan, etc.) to transform a PL formula in CNF can increase the number of terms (exponentially).

Example

- Input: a DNF formula

$$(x_1 \wedge y_1) \vee (x_2 \wedge y_2) \vee (x_3 \wedge y_3)$$

  the equivalent CNF is

$$(x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee y_3) \wedge (x_1 \vee x_3 \vee y_2) \wedge (x_1 \vee y_2 \vee y_3) \wedge \\ (x_2 \vee x_3 \vee y_1) \wedge (x_2 \vee y_1 \vee y_3) \wedge (x_3 \vee y_1 \vee y_2) \wedge (y_1 \vee y_2 \vee y_3)$$

Consequence

We need an other method to transform logical formula into CNF

Equisatifiable formula

Tseitin transformation produces a CNF formula F' from F which grows linearly but it will be only equisatifiable, *i.e.*, F is satisfiable iff F' is satisfiable.

The idea is to add new variables:

- $A \wedge B \equiv X$ and the CNF formula is produced
  $(\neg A \vee \neg B \vee X) \wedge (A \vee \neg X) \wedge (B \vee \neg X)$
  Derivation:
  - X is true when A and B are true, X is false when either A or B is false
  - So $(X \implies (A \wedge B)) \wedge (\neg X \implies (\neg A \vee \neg B))$
  - $(\neg X \vee (A \wedge B)) \wedge (X \vee \neg A \vee \neg B)$
- $A \vee B \equiv X$ and the CNF formula is produced
  $(A \vee B \vee \neg X) \wedge (\neg A \vee X) \wedge (\neg B \vee X)$

Consequence

We have a linear growing formula in the number of operators but the number of variables increases

Example of Tseitin transformation

Transformation of $F : (A \wedge B) \vee C$ in CNF

1. $A \wedge B$ is transformed into

$$x_1, (\neg A \vee \neg B \vee x_1) \wedge (A \vee \neg x_1) \wedge (B \vee \neg x_1)$$

   and we have to treat $x_1 \vee C$

2. $x_1 \vee C$ is transformed into

$$x_2, (x_1 \vee C \vee \neg x_2) \wedge (\neg x_1 \vee x_2) \wedge (\neg C \vee x_2) \wedge \\ (\neg A \vee \neg B \vee x_1) \wedge (A \vee \neg x_1) \wedge (B \vee \neg x_1)$$

3. and at the end, the CNF is

$$x_2 \wedge (x_1 \vee C \vee \neg x_2) \wedge (\neg x_1 \vee x_2) \wedge (\neg C \vee x_2) \wedge \\ (\neg A \vee \neg B \vee x_1) \wedge (A \vee \neg x_1) \wedge (B \vee \neg x_1)$$

Algorithm for Tseitin's transformation

This can be defined by a recursive function `tseitin`

tseitin(f) is defined by

- tseitin(p) $= (p, \emptyset)$
- tseitin($\neg f$): let $(p', c') =$ tseitin(f) then return $(\neg p', c')$
- tseitin($f_1 \vee f_2$): let $(p_1, c_1) =$ tseitin(f_1), $(p_2, c_2) =$ tseitin(f_2) and p a new literal then return $p, (\neg p \vee p_1 \vee p_2) \wedge (p \vee \neg p_1) \wedge (p|\neg p_2) \wedge c_1 \wedge c_2)$
- tseitin($f_1 \wedge f_2$): let $(p_1, c_1) =$ tseitin(f_1), $(p_2, c_2) =$ tseitin(f_2) and p a new literal then return $(p, (p \vee \neg p_1 \vee \neg p_2) \wedge (\neg p \vee p_1) \wedge (\neg p \vee p2) \wedge c_1 \wedge c_2)$
- tseitin($f_1 \implies f_2$): let $(p_1, c_1) =$ tseitin(f_1), $(p_2, c_2) =$ tseitin(f_2) and p a new literal then return $(p, (\neg p \vee \neg p_1 \vee p_2) \wedge (p \vee p_1) \wedge (p \vee \neg p_2) \wedge c_1 \wedge c_2)$
- tseitin($f_1 \Leftrightarrow f_2$): let $(p_1, c_1) =$ tseitin(f_1), $(p_2, c_2) =$ tseitin(f_2) and p a new literal then return $(p, (\neg p \vee \neg p_1 \vee p_2) \wedge (\neg p \vee p_1 \vee \neg p_2) \wedge (p \vee p_1 \vee p_2) \wedge (p \vee \neg p_1 \vee \neg p_2) \wedge c_1 \wedge c_2)$

The final CNF is given by

$$\text{Let} \quad (l, c) = \text{tseitin}(f) \quad \text{then the CNF is} \quad \ell \wedge c$$

How to assert validity/satisfiability

We want to prove $\alpha \models \beta$

- **Model checking**
  - Enumerate all the valuation of α and check if produce true value
  - For entailment, check that all models of α is also a model for β
- **Theorem proving**
  - Search for a sequence of proof steps goind from α to β (inference rule)
  - For example, **Modus ponens** rule from $P \land (P \implies Q)$ we can infer Q

A simple decision procedure

Based on Truth Table a simple algorithm to decide if a PL formula F is satisfiable can be given.

```
let rec sat f =
  if f = ⊤ then true
  else if f = ⊥ then false
  else
    let p = choose(vars(f)) in
    (sat f{p ↦ ⊤}) or (sat f{p ↦ ⊥})
```

See Lecture 2 for an improved algorithm (CDCL)

Resolution method

We can see this method as a generalization of the modus *ponens rule*.

A refutation method

To prove a formula F, we refute $\neg F$ that is $\neg F$ is unsatisfiable.

Resolution method – rules

Let two clauses

$$C = \{\mathrm{x}, \ell_1, \ldots, \ell_n\}$$
$$C' = \{\bar{\mathrm{x}}, \ell'_1, \ldots, \ell'_m\}$$

From resolution principle, the clause $C \wedge C'$ is equi-satifiable of

$$R = \{\ell_1, \ldots, \ell_n, \ell'_1, \ldots, \ell'_m\}$$

R is named the **resolvent** of C and C' in regards of x

Remark C and C' cannot be true in the same time because of x. The satisfiability of $C \wedge C'$ depends on the truth value of other variables.

Resolution method – rules cont'

Definition: Fusion

$$\{\ell_1, \ell_1, \ell_2, \dots, \ell_n\}$$

is equi-satifiable to

$$\{\ell_1, \ell_2, \dots, \ell_n\}$$

Definition: Tautology

$$\{\bar{\ell}_1, \ell_1, \ell_2, \dots, \ell_n\}$$

is equi-satifiable to $\{\top\}$

Definition: Subsomption

$$(C \vee C') \wedge C$$

is equi-satifiable to C

Examples

Modus ponens

$$\{\neg P, Q\} \wedge P$$
$$Q$$

General rules

$$\{\neg S, \neg P, Q, R\} \wedge \{Q, P, T\}$$
$$\{\neg S, Q, R, T\}$$

A small refutation proof

$$\{\neg P, \neg Q, R\} \wedge \{\neg R\} \wedge \{\neg P, Q\} \wedge \{P\}$$
$$\{\neg P, \neg Q\} \wedge \{\neg P, Q\} \wedge \{P\}$$
$$\{\neg P\} \wedge \{P\}$$
$$\{\}$$

Unsatisfiability proof with resolution method

Resolution method can be used to eliminate variable one at a time.

The principle is the following: repeat

- Choose a variable x
- Compute resolvent for each couple of clauses $\{x, A\}$ and $\{\bar{x}, B\}$
- Remove all clauses containing x or $\bar{x}$

If the empty clause is found then the formula is UNSAT otherwise the formula is SAT.

The resolution method is complete and correct, *i.e.* it is a decision procedure. **But** very bad complexity on high dimensional problem, algorithm spends a lot of time to choose variables to compute resolvent

What is an encoding?

- SAT solvers only consider CNF propositional formulas as input
- To solve combinatorial problems with SAT solvers, constraints have to be represented in this language
- An encoding of a problem P (*i.e.* a set of constraints) into a propositional formula F that expresses P,
  so that there is a bijection solutions to P and models of F

The n-Queens problem. $n = 2$

Let $P_{i,j}$ the proposition which is true when a queen is at column i and row j on the chess board.

We consider simplest problem of 2 queens on a board of dimension 2×2.

Stating the constraints:

- Only one queen per column

$$\begin{aligned} C &\equiv (P_{1,1} \text{ xor } P_{1,2}) \wedge (P_{2,1} \text{ xor } P_{2,2}) \\ &\equiv (P_{1,1} \vee P_{1,2}) \wedge (\neg P_{1,1} \vee \neg P_{1,2}) \wedge (P_{2,1} \vee P_{2,2}) \wedge (\neg P_{2,1} \vee \neg P_{2,2}) \end{aligned}$$

- Only one queen per row

$$\begin{aligned} C &\equiv (P_{1,1} \text{ xor } P_{2,1}) \wedge (P_{1,2} \text{ xor } P_{2,2}) \\ &\equiv (P_{1,1} \vee P_{2,1}) \wedge (\neg P_{1,1} \vee \neg P_{2,1}) \wedge (P_{1,2} \vee P_{2,2}) \wedge (\neg P_{1,2} \vee \neg P_{2,2}) \end{aligned}$$

- Only one queen per diagonal

$$\begin{aligned} C &\equiv (P_{1,1} \text{ xor } P_{2,2}) \wedge (P_{1,2} \text{ xor } P_{2,1}) \\ &\equiv (P_{1,1} \vee P_{2,2}) \wedge (\neg P_{1,1} \vee \neg P_{2,2}) \wedge (P_{1,2} \vee P_{2,1}) \wedge (\neg P_{1,2} \vee \neg P_{2,1}) \end{aligned}$$

Solution of the problem

A valuation of variables which makes $C \wedge R \wedge D$ true.

Problem solving with Propositional Logic

Design/Prove flow

Problem $\rightarrow$ PL formula $\rightarrow$ CNF $\rightarrow$ Inference engine $\rightarrow$ Yes/No

Encoding

Is a complete problem and many techniques have been developped