Languages and Algorithms for Artificial Intelligence (Module 2)

Last update: 09 December 2023

Contents

1 Propositional logic

1.1 Syntax

Syntax Rules and symbols to define well-formed sentences.

The symbols of propositional logic are:

Proposition symbols p_0, p_1, ...

Connectives $\wedge \vee \rightarrow \leftrightarrow \neg \perp$ ( )

Well-formed formula The definition of a well-formed formula is recursive:

- An atomic proposition is a well-formed formula.
- If S is well-formed, $\neg S$ is well-formed.
- If S_1 and S_2 are well-formed, $S_1 \wedge S_2$ is well-formed.
- If S_1 and S_2 are well-formed, $S_1 \vee S_2$ is well-formed.

Note that the implication $S_1 \rightarrow S_2$ can be written as $\neg S_1 \vee S_2$.

The BNF definition of a formula is:

$$F := \texttt{atomic\_proposition} \,|\, F \wedge F \,|\, F \vee F \,|\, F \rightarrow F \,|\, F \leftrightarrow F \,|\, \neg F \,|\, (F)$$

1.2 Semantics

Semantics Rules to associate a meaning to well-formed sentences.

Model theory What is true.

Proof theory What is provable.

Interpretation Given a propositional formula F of n atoms $\{A_1, \ldots, A_n\}$, an interpretation $\mathcal{I}$ of F is is a pair (D, I) where:

- D is the domain. Truth values in the case of propositional logic.
- I is the interpretation mapping that assigns to the atoms $\{A_1, \ldots, A_n\}$ an element of D.

Note: given a formula F of n distinct atoms, there are 2^n district interpretations.

Model If F is true under the interpretation $\mathcal{I}$, we say that $\mathcal{I}$ is a model of F ($\mathcal{I} \models F$).

Valid formula A formula F is valid (tautology) iff it is true in all the possible interpretations. It is denoted as $\models F$.

Invalid formula A formula F is invalid iff it is not valid (:o).

In other words, there is at least an interpretation where F is false.

Inconsistent formula A formula F is inconsistent (unsatisfiable) iff it is false in all the possible interpretations.

Consistent formula A formula F is consistent (satisfiable) iff it is not inconsistent.

In other words, there is at least an interpretation where F is true.

Decidability A logic is decidable if there is a terminating method to decide if a formula is valid.

Propositional logic is decidable.

Truth table Useful to define the semantics of connectives.

- $\neg S$ is true iff S is false.
- $S_1 \wedge S_2$ is true iff S_1 is true and S_2 is true.
- $S_1 \vee S_2$ is true iff S_1 is true or S_2 is true.
- $S_1 \rightarrow S_2$ is true iff S_1 is false or S_2 is true.
- $S_1 \leftrightarrow S_2$ is true iff $S_1 \rightarrow S_2$ is true and $S_1 \leftarrow S_2$ is true.

Evaluation The connectives of a propositional formula are evaluated in the order:

$$\leftrightarrow, \rightarrow, \vee, \wedge, \neg$$

Formulas in parenthesis have higher priority.

Logical consequence Let $\Gamma = \{F_1, \ldots, F_n\}$ be a set of formulas (premises) and G a formula (conclusion). G is a logical consequence of Γ ($\Gamma \models G$) if in all the possible interpretations $\mathcal{I}$, if $F_1 \wedge \cdots \wedge F_n$ is true, G is true.

Logical equivalence Two formulas F and G are logically equivalent ($F \equiv G$) iff the truth values of F and G are the same under the same interpretation. In other words, $F \equiv G \iff F \models G \wedge G \models F$.

Common equivalences are:

Commutativity : $(P \wedge Q) \equiv (Q \wedge P)$ and $(P \vee Q) \equiv (Q \vee P)$

Associativity : $((P \wedge Q) \wedge R) \equiv (P \wedge (Q \wedge R))$ and $((P \vee Q) \vee R) \equiv (P \vee (Q \vee R))$

Double negation elimination : $\neg(\neg P) \equiv P$

Contraposition : $(P \rightarrow Q) \equiv (\neg Q \rightarrow \neg P)$

Implication elimination : $(P \rightarrow Q) \equiv (\neg P \vee Q)$

Biconditional elimination : $(P \leftrightarrow Q) \equiv ((P \rightarrow Q) \wedge (Q \rightarrow P))$

De Morgan : $\neg(P \wedge Q) \equiv (\neg P \vee \neg Q)$ and $\neg(P \vee Q) \equiv (\neg P \wedge \neg Q)$

Distributivity of $\wedge$ over $\vee$: $(P \wedge (Q \vee R)) \equiv ((P \wedge Q) \vee (P \wedge R))$

Distributivity of $\vee$ over $\wedge$: $(P \vee (Q \wedge R)) \equiv ((P \vee Q) \wedge (P \vee R))$

1.2.1 Normal forms

Negation normal form (NNF) A formula is in negation normal form iff negations appear only in front of atoms (i.e. not parenthesis).

Conjunctive normal form (CNF) A formula F is in conjunctive normal form iff:

- it is in negation normal form;

- it has the form $F := F_1 \wedge F_2 \cdots \wedge F_n$, where each F_i (clause) is a disjunction of literals.

Example.

$(\neg P \vee Q) \wedge (\neg P \vee R)$ is in CNF.

$\neg (P \vee Q) \wedge (\neg P \vee R)$ is not in CNF (not in NNF).

Disjunctive normal form (DNF) A formula F is in disjunctive normal form iff:
- it is in negation normal form;
- it has the form $F := F_1 \vee F_2 \cdots \vee F_n$, where each F_i is a conjunction of literals.

1.3 Reasoning

Reasoning method Systems to work with symbols.

Given a set of formulas Γ, a formula F and a reasoning method E, we denote with $\Gamma \vdash^E F$ the fact that F can be deduced from Γ using the reasoning method E.

Sound A reasoning method E is sound iff:

$$(\Gamma \vdash^E F) \rightarrow (\Gamma \models F)$$

Complete A reasoning method E is complete iff:

$$(\Gamma \models F) \rightarrow (\Gamma \vdash^E F)$$

Deduction theorem Given a set of formulas $\{F_1, \ldots, F_n\}$ and a formula G:

$$(F_1 \wedge \cdots \wedge F_n) \models G \iff \models (F_1 \wedge \cdots \wedge F_n) \rightarrow G$$

Proof.

$\rightarrow$) By hypothesis $(F_1 \wedge \cdots \wedge F_n) \models G$.

So, for each interpretation $\mathcal{I}$ in which $(F_1 \wedge \cdots \wedge F_n)$ is true, G is also true. Therefore, $\mathcal{I} \models (F_1 \wedge \cdots \wedge F_n) \rightarrow G$.

Moreover, for each interpretation $\mathcal{I}'$ in which $(F_1 \wedge \cdots \wedge F_n)$ is false, $(F_1 \wedge \cdots \wedge F_n) \rightarrow G$ is true. Therefore, $\mathcal{I}' \models (F_1 \wedge \cdots \wedge F_n) \rightarrow G$.

In conclusion, $\models (F_1 \wedge \cdots \wedge F_n) \rightarrow G$.

$\leftarrow$) By hypothesis $\models (F_1 \wedge \cdots \wedge F_n) \rightarrow G$. Therefore, for each interpretation where $(F_1 \wedge \cdots \wedge F_n)$ is true, G is also true.

In conclusion, $(F_1 \wedge \cdots \wedge F_n) \models G$.

$\square$

Refutation theorem Given a set of formulas $\{F_1, \ldots, F_n\}$ and a formula G:

$$(F_1 \wedge \cdots \wedge F_n) \models G \iff F_1 \wedge \cdots \wedge F_n \wedge \neg G \text{ is inconsistent}$$

Note: this theorem is not accepted in intuitionistic logic.

Proof. By definition, $(F_1 \wedge \cdots \wedge F_n) \models G$ iff for every interpretation where $(F_1 \wedge \cdots \wedge F_n)$ is true, G is also true. This requires that there are no interpretations where $(F_1 \wedge \cdots \wedge F_n)$ is true and G false. In other words, it requires that $(F_1 \wedge \cdots \wedge F_n \wedge \neg G)$ is inconsistent. $\square$

1.3.1 Natural deduction

Proof theory Set of rules that allows to derive conclusions from premises by exploiting syntactic manipulations.

Natural deduction Set of rules to introduce or eliminate connectives. We consider a subset $\{\wedge, \to, \bot\}$ of functionally complete connectives.

Natural deduction can be represented using a tree like structure:

$$[\text{hypothesis}]$$
$$\vdots$$
$$\frac{\text{premise}}{\text{conclusion}} \text{ rule name}$$

The conclusion is true when the hypothesis are able to prove the premise. Another tree can be built on top of premises to prove them.

Introduction Usually used to prove the conclusion by splitting it.

$$\frac{\psi \quad \varphi}{\varphi \wedge \psi} \wedge \text{I} \qquad\qquad \begin{array}{c} [\varphi] \\ \vdots \\ \psi \\ \hline \varphi \to \psi \end{array} \to \text{I}$$

Elimination Usually used to exploit hypothesis and derive a conclusion.

$$\frac{\varphi \wedge \psi}{\varphi} \wedge \text{E} \qquad\qquad \frac{\varphi \wedge \psi}{\psi} \wedge \text{E} \qquad\qquad \frac{\varphi \quad \varphi \to \psi}{\psi} \to \text{E}$$

Ex falso sequitur quodlibet From contradiction, anything follows. This can be used when we have two contradicting hypothesis.

$$\frac{\bot}{\varphi} \bot$$

Reductio ad absurdum Assume the opposite and prove a contradiction (not accepted in intuitionistic logic).

$$\begin{array}{c} [\neg\varphi] \\ \vdots \\ \frac{\bot}{\varphi} \end{array} \text{RAA}$$

2 First order logic

2.1 Syntax

The symbols of propositional logic are:

Constants Known elements of the domain. Do not represent truth values.

Variables Unknown elements of the domain. Do not represent truth values.

Function symbols Function $f^{(n)}$ applied on n constants to obtain another constant.

Predicate symbols Function $P^{(n)}$ applied on n constants to obtain a truth value.

Connectives $\forall\ \exists\ \wedge\ \vee\ \rightarrow\ \neg\ \leftrightarrow\ \top\ \bot\ (\ )$

Using the basic syntax, the following constructs can be defined:

Term Denotes elements of the domain.

$$t := \texttt{constant} \,|\, \texttt{variable} \,|\, f^{(n)}(t_1, \ldots, t_n)$$

Proposition Denotes truth values.

$$P := \top \,|\, \bot \,|\, P \wedge P \,|\, P \vee P \,|\, P \rightarrow P \,|\, P \leftrightarrow P \,|\, \neg P \,|\, \forall x.P \,|\, \exists x.P \,|\, (P) \,|\, P^{(n)}(t_1, \ldots, t_n)$$

Well-formed formula The definition of well-formed formula in first order logic extends the one of propositional logic by adding the following conditions:

- If S is well-formed, $\exists X.S$ is well-formed. Where X is a variable.
- If S is well-formed, $\forall X.S$ is well-formed. Where X is a variable.

Free variables The universal and existential quantifiers bind their variable within the scope of the formula. Let $F_v(F)$ be the set of free variables in a formula F, F_v is defined as follows:

- $F_v(p(t)) = \bigcup \texttt{vars}(t)$
- $F_v(\top) = F_v(\bot) = \varnothing$
- $F_v(\neg F) = F_v(F)$
- $F_v(F_1 \wedge F_2) = F_v(F_1 \vee F_2) = F_v(F_1 \rightarrow F_2) = F_v(F_1) \cup F_v(F_2)$
- $F_v(\forall X.F) = F_v(\exists X.F) = F_v(F) \smallsetminus \{X\}$

Closed formula/Sentence Proposition without free variables.

Theory Set of sentences.

Ground term/Formula Proposition without variables.

2.2 Semantics

Interpretation An interpretation in first order logic $\mathcal{I}$ is a pair (D, I):

- D is the domain of the terms.
- I is the interpretation function such that:
    - $I(f) : D^n \to D$ for every n-ary function symbol.
    - $I(p) \subseteq D^n$ for every n-ary predicate symbol.

Variable evaluation Given an interpretation $\mathcal{I} = (D, I)$ and a set of variables $\mathcal{V}$, a variable is evaluated through $\eta : \mathcal{V} \to D$.

Model Given an interpretation $\mathcal{I}$ and a formula F, $\mathcal{I}$ models F ($\mathcal{I} \models F$) when $\mathcal{I}, \eta \models F$ for every variable evaluation η.

A sentence S is:

Valid S is satisfied by every interpretation ($\forall \mathcal{I} : \mathcal{I} \models S$).

Satisfiable S is satisfied by some interpretations ($\exists \mathcal{I} : \mathcal{I} \models S$).

Falsifiable S is not satisfied by some interpretations ($\exists \mathcal{I} : \mathcal{I} \not\models S$).

Unsatisfiable S is not satisfied by any interpretation ($\forall \mathcal{I} : \mathcal{I} \not\models S$).

Logical consequence A sentence T_1 is a logical consequence of T_2 ($T_2 \models T_1$) if every model of T_2 is also model of T_1:

$$\mathcal{I} \models T_2 \to \mathcal{I} \models T_1$$

Theorem 2.2.1. It is undecidable to determine if a first order logic formula is a tautology.

Equivalence A sentence T_1 is equivalent to T_2 if $T_1 \models T_2$ and $T_2 \models T_1$.

Theorem 2.2.2. The following statements are equivalent:

1. $F_1, \ldots, F_n \models G$.

2. $(\bigwedge_{i=1}^n F_i) \to G$ is valid.

3. $(\bigwedge_{i=1}^n F_i) \wedge \neg G$ is unsatisfiable.

2.3 Substitution

Substitution A substitution $\sigma : \mathcal{V} \to \mathcal{T}$ is a mapping from variables to terms. It is written as $\{X_1 \mapsto t_1, \ldots, X_n \mapsto t_n\}$.

The application of a substitution is the following:

- $p(t_1, \ldots, t_n)\sigma = p(t_1\sigma, \ldots, t_n\sigma)$
- $f(t_1, \ldots, t_n)\sigma = fp(t_1\sigma, \ldots, t_n\sigma)$
- $\perp\sigma = \perp$ and $\top\sigma = \top$
- $(\neg F)\sigma = (\neg F\sigma)$
- $(F_1 \star F_2)\sigma = (F_1\sigma \star F_2\sigma)$ for $\star \in \{\wedge, \vee, \to\}$

- $(\forall X.F)\sigma = \forall X'(F\sigma[X \mapsto X'])$ where X' is a fresh variable (i.e. does not appear in F).

- $(\exists X.F)\sigma = \exists X'(F\sigma[X \mapsto X'])$ where X' is a fresh variable.

Unifier A substitution σ is a unifier for $e_1, \ldots, e_n$ if $e_1\sigma = \cdots = e_n\sigma$.

    Most general unifier A unifier σ is the most general unifier (MGU) for $\bar{e} = e_1, \ldots, e_n$ if every unifier τ for $\bar{e}$ is an instance of σ ($\tau = \sigma\rho$ for some substitution ρ). In other words, σ is the smallest substitution to unify $\bar{e}$.

3 Logic programming

3.1 Syntax

A logic program has the following components (defined using BNF):

Atom $A := p(t_1, \ldots, t_n)$ for $n \geq 0$

Goal $G := \top \mid \bot \mid A \mid G_1 \wedge G_2$

Horn clause A clause with at most one positive literal.

$$K := A \leftarrow G$$

In other words, A and all the literals in G are positive as $A \leftarrow G = A \vee \neg G$.

Program $P := K_1 \ldots K_m$ for $m \geq 0$

3.2 Semantics

3.2.1 State transition system

State Pair $\langle G, \theta \rangle$ where G is a goal and θ is a substitution.

    Intial state $\langle G, \varepsilon \rangle$

    Successful final state $\langle \top, \theta \rangle$

    Failed final state $\langle \bot, \varepsilon \rangle$

Derivation A sequence of states. A derivation can be:

    Successful If the final state is successful.

    Failed If the final state is failed.

    Infinite If there is an infinite sequence of states.

Given a derivation, a goal G can be:

    Successful There is a successful derivation starting from $\langle G, \varepsilon \rangle$.

    Finitely failed All the derivations starting from $\langle G, \varepsilon \rangle$ are failed.

Computed answer substitution Given a goal G and a program P, if there exists a successful derivation $\langle G, \varepsilon \rangle \mapsto *\langle \top, \theta \rangle$, then the substitution θ is the computed answer substitution of G.

Transition Starting from the state $\langle A \wedge G, \theta \rangle$ of a program P, a transition on the atom A can result in:

    Unfold If there exists a clause $(B \leftarrow H)$ in P and a (most general) unifier β for $A\theta$ and B, then we have a transition: $\langle A \wedge G, \theta \rangle \mapsto \langle H \wedge G, \theta\beta \rangle$.

    In other words, we want to prove that $A\theta$ holds. To do this, we search for a clause that has as conclusion $A\theta$ and add its premise to the things to prove. If a unification is needed to match $A\theta$, we add it to the substitutions of the state.

Failure If there are no clauses $(B \leftarrow H)$ in P with a unifier for $A\theta$ and B, then we have a transition: $\langle A \wedge G, \theta \rangle \mapsto \langle \bot, \varepsilon \rangle$.

Non-determinism A transition has two types of non-determinism:

>**Don't-care** Any atom in $(A \wedge G)$ can be chosen to determine the next state. This affects the length of the derivation (infinite in the worst case).

>**Don't-know** Any clause $(B \rightarrow H)$ in P with an unifier for $A\theta$ and B can be chosen. This determines the output of the derivation.

Selective linear definite resolution Approach to avoid non-determinism when constructing a derivation.

Selection rule Method to select the atom in the goal to unfold (eliminates don't-care non-determinism).

SLD tree Search tree constructed using all the possible clauses according to a selection rule and visited following a search strategy (eliminates don't know non-determinism).

Theorem 3.2.1 (Soundness). Given a program P, a goal G and a substitution θ, if θ is a computed answer substitution, then $P \models G\theta$.

Theorem 3.2.2 (Completeness). Given a program P, a goal G and a substitution θ, if $P \models G\theta$, then it exists a computed answer substitution σ such that $G\theta = G\sigma\beta$.

Theorem 3.2.3. If a computed answer substitution can be obtained using a selection rule r, it can be obtained also using a different selection rule r'.

Prolog SLD

Selection rule Select the leftmost atom.

Tree search strategy Search following the order of definition of the clauses.

This results in a left-to-right, depth-first search of the SLD tree. Note that this may end up in a loop.

4 Prolog

See `Fundamentals of AI and KR (module 2)`.

5 Constraint programming

Class of problems

    Constraint satisfaction problem (CSP) Defined by:

- A finite set of variables $X_1, \ldots, X_n$.
- A domain for each variable $D(X_1), \ldots, D(X_n)$.
- A set of constraints $\{C_1, \ldots, C_m\}$

    A solution is an assignment to all the variables while satisfying the constraints.

    Constraint optimization problem (COP) Extension of a constraint satisfaction problem with an objective function with domain D:

$$f : D(X_1) \times \cdots \times D(X_n) \to D$$

    A solution is a CSP solution that optimizes f.

Class of languages

    Constraint logic programming (CLP) Add constraints and solvers to logic programming. Generally more efficient than plain logic programming.

    Imperative languages Add constraints and solvers to imperative languages (e.g. MiniZinc).

5.1 Constraint logic programming (CLP)

5.1.1 Syntax

Atom $A := p(t_1, \ldots, t_n)$, for $n \geq 0$. p is a predicate.

Constraint $C := c(t_1, \ldots, t_n) \mid C_1 \wedge C_2$, for $n \geq 0$. c is an atomic constraint.

Goal $G := \top \mid \bot \mid A \mid C \mid G_1 \wedge G_2$

Constraint logic clause $K := A \leftarrow G$

Constraint logic program $P := K_1 \ldots K_m$, for $m \geq 0$

5.1.2 Semantics

Transition system

    State Pair $\langle G, C \rangle$ where G is a goal and C is a constraint.

        Initial state $\langle G, \top \rangle$

        Successful final state $\langle \top, C \rangle$ with $C \neq \bot$

        Failed final state $\langle G, \bot \rangle$

Transition Starting from the state $\langle A \wedge G, C \rangle$ of a program P, a transition on the atom A can result in:

Unfold If there exists a clause $(B \leftarrow H)$ in P and an assignment $(B \doteq A)$ such that $((B \doteq A) \wedge C)$ is still valid, then we have a transition $\langle A \wedge G, C \rangle \mapsto \langle H \wedge G, (B \doteq A) \wedge C \rangle$.

In other words, we want to develop an atom A and the current constraints are denoted as C. We look for a clause whose head equals A, applying an assignment if needed. If this is possible, we transit from solving A to solving the body of the clause and add the assignment to the set of active constraints.

Failure If there are no clauses $(B \leftarrow H)$ with a valid assignment $((B \doteq A) \wedge C)$, then we have a transition $\langle A \wedge G, C \rangle \mapsto \langle \bot, \bot \rangle$.

Moreover, starting from the state $\langle C \wedge G, D_1 \rangle$ of a program P, a transition on the constraint C can result in:

Solve If $(C \wedge D_1) \iff D_2$ holds, then we have a transition $\langle C \wedge G, D_1 \rangle \mapsto \langle G, D_2 \rangle$.

In other words, we want to develop a constraint C and the current constraints are denoted as D_1. If $(C \wedge D_1)$ is valid, we call it D_2 and continue solving the rest of the goal constrained to D_2.

Non-determinism As in logic programming, there is don't-care and don't-know non-determinism. A SLD search tree is also used.

Derivation strategies

Generate-and-test Strategy adopted by logic programs. Every possible assignment to the variables are generated and tested.

The workflow is the following:

1. Determine domain.
2. Make an assignment to each variable.
3. Test the constraints.

Constrain-and-generate Strategy adopted by constraint logic programs. Exploit facts to reduce the search space.

The workflow is the following:

1. Determine domain.
2. Restrict the domain following the constraints.
3. Make an assignment to each variable.