Logic Programming and Deductive Databases

Chapter 3: Clausal Logic

Prof. Dr. Stefan Brass

Martin-Luther-Universität Halle-Wittenberg

Summer 2021

http://www.informatik.uni-halle.de/~brass/lp21/

Objectives

After completing this chapter, you should be able to:

- explain the notions: literal, positive literal, negative literal, clause, empty clause, Horn clause, definite Horn clause.
- translate between clauses written as disjunctions and clauses written as rules.
- define and explain Herbrand interpretations.

  Including the notions Herbrand universe and Herbrand base.

  Compare Herbrand interpretaions with general interpretations.
- explain when it is sufficient to look only at Herbrand interpretations.
- transform formulas into sets of clauses.

  Including the technique of Skolemization to eliminate existential quantifiers.

Contents

Normal Forms (1)

Definition:

- A formula F is in Prenex Normal Form iff it is closed and has the form

$$\Theta_1 X_1 : s_1 \ \ldots \ \Theta_n X_n : s_n \ \ G$$

where $\{\Theta_1, \ldots, \Theta_n\} \subseteq \{\forall, \exists\}$ and G is quantifier-free.

- A formula F is in Disjunctive Normal Form iff it is in Prenex Normal Form, and G has the form

$$(G_{1,1} \wedge \cdots \wedge G_{1,k_1}) \ \vee \cdots \vee \ (G_{n,1} \wedge \cdots \wedge G_{n,k_n}),$$

where each $G_{i,j}$ is an atomic formula or a negated atomic formula.

Normal Forms (2)

Remark:

- Conjunctive Normal Form is like disjunctive normal form, but G must have the form

$$(G_{1,1} \vee \cdots \vee G_{1,k_1}) \wedge \cdots \wedge (G_{n,1} \vee \cdots \vee G_{n,k_n}).$$

Theorem:

- Under the assumption of non-empty domains, every formula can be equivalently translated into
  - prenex normal form,
  - disjunctive normal form, and
  - conjunctive normal form.

Contents

Literals, Clauses (1)

Definition:

- A literal is an atomic formula ("positive literal") or a negated atomic formula ("negative literal").
- A clause is a disjunction of zero or more literals.
  The empty clause is treated as false. All variables in a clause are treated as $\forall$-quantified.

  A clause is often seen as set of literals.
- A Horn clause is a clause with at most one positive literal.
  A definite Horn clause is a clause with exactly one positive literal.
- A logic program is a set of definite Horn clauses.

  Sometimes it is called "definite logic program" in order to distinguish it from later extensions.

Literals, Clauses (2)

- Clauses can be written as implications with the positive literals in the head, and negative literals (unnegated) in the body:

$$\underbrace{A_1 \vee \cdots \vee A_n}_{\text{Head}} \leftarrow \underbrace{B_1 \wedge \cdots \wedge B_m}_{\text{Body}}$$

  Formally, this is not a clause, but a formula that is equivalent to a clause.

  This kind of formula is also called a disjunctive rule.

- This corresponds to the disjunction

$$A_1 \vee \cdots \vee A_n \vee \neg B_1 \vee \cdots \vee \neg B_m.$$

- A definite Horn clause can be written as a rule with only one literal in the head:

$$A \leftarrow B_1 \wedge \cdots \wedge B_m.$$

  I.e. this kind of formula is called a rule or Prolog rule.

Literals, Clauses (3)

- A clause with only negative literals $\neg B_1 \vee \cdots \vee \neg B_m$ is written as: $\leftarrow B_1 \wedge \cdots \wedge B_m$.

  Here the head is the empty disjunction, which is understood as false: A disjunction is satisfied if at least one of its elements is true. If there are no elements, it can never be true.

- $\Phi \models G$ iff $\Phi \cup \{\neg\forall(G)\}$ is inconsistent. (See Chapter 2.)

- Refutation theorem provers, such as resolution, try to derive the empty clause "false" from $\Phi \cup \{\neg\forall(G)\}$.

- Therefore, a clause with only negative literals is often understood as proof goal / query: $\exists(B_1 \wedge \cdots \wedge B_m)$.

  This is what should be proven (giving values for the existentially quantified variables). In order to prove it, the negation is added to the program, and the inconsistency is shown by deriving the empty clause.

Skolemnization (1)

- In clauses, all variables are $\forall$-quantified.

  Therefore, no explicit quantifiers are needed.

- However, many logical formulas in prenex normal from do contain existential quantifiers.
- Skolemnization is a technique for removing existential quantifiers.
- The idea of Skolemnization is to introduce names (constants or function symbols) for the values that are required to exist.
- E.g. $\exists X : s\ p(X, a)$ is replaced by $p(c, a)$ with a new constant c of sort s.

Skolemnization (2)

- The new formula is not equivalent (it is a formula over a different signature), but it is consistent whenever the old formula is consistent.

  A model of $\exists X : s\ p(X, a)$ can be extended to a model of $p(c, a)$ by interpreting c as the value for X that makes $p(X, a)$ true.

  Conversely, if one has a model of $p(c, a)$, one can simply forget the interpretation of c (but keep the value in the domain), to get a model of $\exists X : s\ p(X, a)$ for the original signature.

- For refutation theorem provers, only the consistency is important, thus this is no restriction.

Skolemnization (3)

- Suppose that the existential quantifier $\exists X : s$ is in the scope of universal quantifiers $\forall Y_1 : s_1 \ldots \forall Y_n : s_n$.
- Then the value for X may depend on the values for $Y_1, \ldots, Y_n$.
- Thus, Skolemnization replaces each occurrence of the variable X by $f(Y_1, \ldots, Y_n)$ with a new function symbol $f : s_1 \times \cdots \times s_n \to s$.
- With Skolemnization, any formula can be translated into a set of clauses.

But one needs non-empty domains to get first prenex normal form.

Skolemnization (4)

Exercise:

- Consider the foreign key constraint:

$$\forall\, \mathtt{X}, \mathtt{Y}, \mathtt{Z}, \mathtt{D}\, \Big(\mathtt{emp}(\mathtt{X}, \mathtt{Y}, \mathtt{Z}, \mathtt{D}) \rightarrow \exists\, \mathtt{N}, \mathtt{L}\, \mathtt{dept}(\mathtt{D}, \mathtt{N}, \mathtt{L})\Big).$$

- Use the equivalence transformations from Chapter 2 to show that it is equivalent to

$$\forall\, \mathtt{D}\, \exists\, \mathtt{N}, \mathtt{L}\, \forall\, \mathtt{X}, \mathtt{Y}, \mathtt{Z}\, \Big(\mathtt{emp}(\mathtt{X}, \mathtt{Y}, \mathtt{Z}, \mathtt{D}) \rightarrow \mathtt{dept}(\mathtt{D}, \mathtt{N}, \mathtt{L})\Big).$$

  This is prenex normal form.

- What Skolem functions would be introduced for this formula?

Contents

Herbrand Interpretations (1)

Definition:

- A ground term is a variable-free term (i.e. a term built only from constants and function symbols).

Definition:

- A Σ-interpretation $\mathcal{I}$ is a Herbrand interpretation iff
  - for every sort s, the domain $\mathcal{I}[s]$ is the set of all ground terms of sort s, i.e. $\mathcal{I}[s] = TE_{s,\emptyset}(s)$.

    This assumes that for every sort there is at least one ground term.

    Otherwise one must extend the signature.
  - all function symbols are interpreted as the corresponding term constructors, i.e.

$$\mathcal{I}[f](t_1, \ldots, t_n) = f(t_1, \ldots, t_n).$$

Herbrand Interpretations (2)

- In general (non-Herbrand) interpretations, it is possible that
  - there are anonymous domain elements (objects not named by a constant or ground term).

    For universal formulas, such domain elements are not important. "For all" statements are even simpler to satisfy if the quantifiers range only over a subset.
  - different ground terms can denote the same object, e.g. $1+1$ and 2, or `murderer` and `buttler`.

    As long as the logic contains no real equality, this is no problem: One simply defines all predicates such that they treat e.g. $1+1$ and 2 in the same way. One can have a user-defined "$=$".
- It is quite difficult to consider arbitrary interpretations.

Herbrand Interpretations (3)

- Herbrand interpretations have a
  - fixed domain: Set of all ground terms.
  - fixed interpretation of constants as themselves.
  - fixed interpretation of function symbols as term constructors ("free interpretation").
- Thus, only the interpretation of the predicates can be chosen in an Herbrand interpretation.

  I.e. a Herbrand interpretation is given by the interpretation of the predicates.
- If the set of ground terms is finite, there are only finitely many Herbrand interpretations.

Herbrand Interpretations (4)

Definition:

- The Herbrand universe $\mathcal{U}_\Sigma$ for a signature Σ is the set of all ground terms that can be constructed with the constants and function symbols in Σ.

  If the signature should contain no constant, one adds one constant a (so that the Herbrand universe is not empty).

- For a logic program P, the Herbrand universe $\mathcal{U}_P$ is the set of ground terms that can be built with the constants and function symbols that appear in P.

  I.e. if a signature is not explicitly given, one assumes that the signature contains only the constants and function symbols that appear in P. Must must again add a constant if $\mathcal{U}_P$ would otherwise be empty.

Herbrand Interpretations (5)

Definition:

- The Herbrand base $\mathcal{B}_\Sigma$ is the set of all positive ground literals that can be built over Σ.

  Again, one must ensure that the set is not empty be adding a constant if Σ does not contain any constant.

- I.e. the Herbrand base is the set of all formulas of the form $p(t_1, \ldots, t_n)$, where p is a predicate of arity n in Σ, and $t_1, \ldots, t_n \in \mathcal{U}_\Sigma$.

- Again, if instead of a signature Σ, a logic program P is given, one constructs the signature of the symbols that appear in P.

Herbrand Interpretations (6)

- A Herbrand interpretation $\mathcal{I}$ can be identified with the set of all positive ground literals $p(t_1, \ldots, t_n)$ that are true in $\mathcal{I}$, i.e. with $H := \{A \in \mathcal{B}_\Sigma \mid \mathcal{I} \models A\}$.
- Conversely, $H \subseteq \mathcal{B}_\Sigma$ denotes the Herbrand interpretation with

$$\mathcal{I}[p] := \{(t_1, \ldots, t_n) \in \mathcal{U}_\Sigma^n \mid p(t_1, \ldots, t_n) \in H\}.$$

  Otherwise, $\mathcal{I}$ is fixed, because it is a Herbrand interpretation: For the single sort s, $\mathcal{I}[s] := \mathcal{U}_\Sigma$, for constants c, $\mathcal{I}[c] := c$, and for function symbols f of arity n: $\mathcal{I}[f](t_1, \ldots, t_n) := f(t_1, \ldots, t_n)$.

- Thus, in the following, Herbrand interpretations are subsets of $\mathcal{B}_\Sigma$.

Herbrand Interpretations (7)

Definition:

- A Herbrand model of a logic program P is a Herbrand interpretation $\mathcal{I}$ that is a model of P.

Exercise:

- Name two different Herbrand models of P:

$$p(a).$$
$$p(b).$$
$$q(a,b).$$
$$r(X) \leftarrow p(X) \wedge q(X,Y).$$

- Please name also a Herbrand interpretation that is not a Herbrand model of P.

Herbrand Interpretations (8)

Theorem:

- A set Φ of universal formulas (formulas in prenex normal form with only universal quantifiers) without "$=$" is consistent iff it has a Herbrand model.

  Assuming that only interpretations are considered with non-empty domains.

Remark:

- The consistency of a set of formulas does not depend on the signature. Thus, it suffices to consider Herbrand interpretations with respect to the signature that contains only the symbols appearing in Φ.

Herbrand Interpretations (9)

Example/Exercise:

- Suppose that we want to prove that $\Phi := \{p(a)\}$ implies $G := \exists X\ p(X)$.
- $\Phi \models G$ iff $\Phi \cup \{\neg\forall(G)\}$ is inconsistent. Thus, we must check $\Phi' := \{p(a),\ \forall X(\neg p(X))\}$ for consistency.
- It is consistent iff it has a Herbrand model.
- The only ground term is a. Thus, there are only two Herbrand interpretations, $\mathcal{I}_1 := \{p(a)\}$ and $\mathcal{I}_2 := \emptyset$. None of the two satisfies both formulas.
- Exercise: What happens if $G := \forall X\ p(X)$?

Contents

Substitutions (1)

Definition:

- A (Σ, ν)-substitution θ is a mapping from variables to terms, i.e. $\theta\colon VARS \to TE_{\Sigma,\nu} \cup VARS$, such that
  - the set $\{V \in VARS \mid \theta(V) \neq V\}$ is finite,

    This ensures that a substitution can be finitely represented.
  - it respects the sorts, i.e. if $\nu(X) = s$, then $\theta(X) \in TE_{\Sigma,\nu}(s)$.

    If one uses a logic without sorts (as in Prolog), this part is obviously not needed. One also does not need an explicit variable declaration ν.
- A substitution is usually written as set of variable-term-pairs, e.g. $\theta = \{X_1/t_1, \ldots, X_n/t_n\}$.

  It is the identity mapping for all not explicitly mentioned variables.

Substitutions (2)

Definition:

- The result of applying a substitution θ to a term t, written $\theta(t)$ or $t\theta$, is defined as follows:
  - If t is a variable X, then
    $$\theta(t) := \theta(X).$$
  - If t has the form $f(t_1, \ldots, t_n)$, then
    $$\theta(t) := f(t'_1, \ldots, t'_n),$$
    where $t'_i := \theta(t_i)$ for $i := 1, \ldots, n$.

Remark:

- This definition extends the domain of a substitution from variables to terms.

  Terms are tree structures. The substitution recursively descends in the tree and replaces every variable that it finds.

Substitutions (3)

Definition:

- The result of applying a substitution θ to a formula F, written $\theta(F)$ or $F\theta$, is defined as follows:
  - If F is an atomic formula $p(t_1, \ldots, t_n)$, then $\theta(F) := p(t'_1, \ldots, t'_n)$, where $t'_i := \theta(t_i)$.
  - If F is $(\neg G)$, then $\theta(F) := (\neg G')$ with $G' := \theta(G)$.
  - If F has the form $(G_1 \wedge G_2)$, then $\theta(F) := (G'_1 \wedge G'_2)$ where $G'_i := \theta(G_i)$. The same for $\vee$, $\leftarrow$, $\rightarrow$, $\leftrightarrow$.
  - If F has the form $\forall X : s\ G$, then $\theta(F) := \forall X : s\ G'$, where $G' := \theta'(G)$ and θ' agrees with θ except that $\theta'(X) = X$. The same for $\exists$.

Substitutions (4)

- I.e. when a substitution is applied to a formula, one replaces the variables as specified by the substitution and leaves the rest of the formula as it is.
- In this way, the domain of the substitution is again extended to arbitrary formulas. Of course, this also includes literals, clauses, and logic programming rules.
- Only free variables are replaced by a substitution.

  For clausal logic, this is not important, because all variables are free (not bound by a quantifier).
- When a variable X is replaced by a term t, any variables inside t are not touched.

  I.e. t is the final result. Therefore, $\{X/Y,\ Y/X\}$ does not result in an infinite loop. It simply exchanges the two variables.

Substitutions (5)

Example/Remark:

- Consider the substitution $\theta := \{X/a, Y/Z\}$.

  As explained above, substitions are usually written down as a set of variable/term pairs. The example specifies the substitution θ with $\theta(X) = a$, $\theta(Y) = Z$, and $\theta(V) = V$ for all other variables V.

- This substitution applied to the literal $p(X, Y, V, b)$ gives the literal $p(a, Z, V, b)$.

- The postfix notation is often used for applying a substitution, e.g. $A\theta$ means $\theta(A)$.

- Note that a substitution is applied only once, not iteratively. E.g. $\theta = \{X/Y,\ Y/Z\}$ maps $p(X)$ to $p(Y)$, and not to $p(Z)$.

Substitutions (6)

Definition:

- A substitution θ is a ground substitution for a quantifier-free formula F iff it replaces all variables that occur in F by ground terms.

  Since substitutions do not touch quantified variables, and the purpose of ground substitutions is to eliminate all variables, it is important that there are no quantifiers. Of course, clauses and rules are quantifier-free.

Remark:

- Thus, the result of applying a ground substitution to a clause F is a ground clause (variable-free clause).

  I.e. a ground substitution replaces all variables by concrete values.

  For Herbrand interpretations, ground substitutions and variable assignments are basically the same.

Ground Instances (1)

Definition:

- A clause F_1 is an instance of a clause F_2 iff there is a substitution θ with $F_1 = \theta(F_2)$.
- A ground instance is an instance that is variable-free (the result of applying a ground substitution).
- We write $ground(P)$ for the set of all ground instances of rules in P.

  If the signature contains no function symbols and only finitely many constants, the set of ground instances of a logic program is finite. Otherwise, it will be usually be infinite (in the one-sorted case). Many "Answer Set" logic programming systems have an "intelligent grounder" that computes only the interesting ground instances (ground instances that can possibly be applied). In contrast, Prolog does not compute ground instances first: It replaces variables by terms during the proof only as far as needed.

Ground Instances (2)

Example:

- E.g. $\texttt{parent}(\texttt{arno}, \texttt{birgit}) \leftarrow \texttt{mother}(\texttt{arno}, \texttt{birgit})$ is a ground instance of $\texttt{parent}(\texttt{X}, \texttt{Y}) \leftarrow \texttt{mother}(\texttt{X}, \texttt{Y})$.
- The ground substitution is $\theta = \{\texttt{X}/\texttt{arno},\ \texttt{Y}/\texttt{birgit}\}$.
- E.g. $\texttt{parent}(\texttt{arno}, \texttt{chris}) \leftarrow \texttt{mother}(\texttt{arno}, \texttt{chris})$ is another ground instance of the same rule.
- E.g. $\texttt{parent}(\texttt{chris}, \texttt{birgit}) \leftarrow \texttt{mother}(\texttt{birgit}, \texttt{doris})$ is not a ground instance of the above rule.

  One must of course replace all occurrences of the same variable in a rule by the same value (when computing a single ground instance of a single rule).

Ground Instances (3)

Exercise:

- Let the following rule be given:

$$p(a, X) \leftarrow q(X, Y) \wedge r(Y, a).$$

- Which of the following rules are ground instances of the given rule?
  - ☐ $p(a, a) \leftarrow q(a, a) \wedge r(a, a).$
  - ☐ $p(a, b) \leftarrow q(a, b) \wedge r(b, a).$
  - ☐ $p(a, b) \leftarrow q(b, c) \wedge r(c, a).$
  - ☐ $p(b, a) \leftarrow q(a, a) \wedge r(a, a).$
  - ☐ $p(a, b) \leftarrow q(b, Y) \wedge r(Y, a).$