Chapter 9

Computational Logic

9.1 Introduction

The goal of this chapter is to provide a self-contained formal account of first-order term algebras and predicate calculus for the purpose of understanding symbolic processing techniques to implement logical reasoning. Although logical inference presented here relies essentially on *resolution refutation*, the formal material we introduce is also relevant to other methods not treated here (*e.g.*, tableaux, equations, natural deduction, *etc*.).

9.2 Algebraic Terms

We first present basic notions for first-order terms and substitutions. Then, we describe a non-deterministic unification procedure as a set of solution-preserving transformations on a set of equations due to the logician Jacques Herbrand [33] and later rediscovered by the computer scientists Alberto Martelli and Ugo Montanari [43]. Then, we present a recursive algorithmic version, and finally an efficient iterative unification algorithm.

9.2.1 Syntax and semantics

DEFINITION 9.2.1 (SIGNATURE) *An (operator)* signature *is a collection* $\Sigma = \biguplus_{n\geq 0} \Sigma_n$ *of (operator) symbols.*

Operator symbols are the elements of a signature Σ; they are also called *function* symbols or *functors*. The *arity* of a functor $f \in \Sigma_n$ is the number n. It denotes the number of arguments of f. Functors of arity 0 are also called *constant* symbols. It will always be assumed that $\Sigma_0 \neq \emptyset$.

DEFINITION 9.2.2 (Σ-INTERPRETATION) *A* Σ-interpretation $\mathfrak{I}$ *is a pair* $\langle \mathcal{D}_{\mathfrak{I}}, [\![\_]\!]_{\mathfrak{I}} \rangle$ *where:*

- $\mathcal{D}_{\mathfrak{I}}$ *is a set, called the* domain *of interpretation;*
- $[\![\_]\!]_{\mathfrak{I}} : \Sigma_n \;\rightarrow\; (\mathcal{D}_{\mathfrak{I}}^n \rightarrow \mathcal{D}_{\mathfrak{I}})$ *is a* denotation *function mapping functors of arity* n *to* n*-ary operations on the domain.*

A Σ-interpretation is also called a Σ-algebra.

Note that, since 0-ary operations on a set S are simply elements of S, this definition implies that $[\![\Sigma_0]\!]_{\mathfrak{I}} \subseteq \mathcal{D}_{\mathfrak{I}}$ (*i.e.*, constant symbols denote elements of the domain).

EXAMPLE 9.2.1 *Consider a signature* Σ *with* $\Sigma_0 = \{\emptyset\}$*,* $\Sigma_1 = \{\varsigma\}$*, and* $\Sigma_2 = \{\star\}$*.*
One possible Σ*-interpretation is given by* $\mathfrak{N} = \langle \mathbb{N}, [\![\_]\!]_{\mathfrak{N}} \rangle$*, where* $\mathbb{N}$ *is the set of all natural numbers, and* $[\![\_]\!]_{\mathfrak{N}}$ *is such that:*

- $[\![\emptyset]\!]_{\mathfrak{N}} = 0$*;*
- $[\![\varsigma]\!]_{\mathfrak{N}}(n) = n + 1$*;*
- $[\![\star]\!]_{\mathfrak{N}}(n, m) = n + m$*.*

Another possible Σ*-interpretation is given by* $\mathfrak{W} = \langle \mathbb{W}, [\![\_]\!]_{\mathfrak{W}} \rangle$*, where* $\mathbb{W}$ *is the set of all strings of* a*'s, and* $[\![\_]\!]_{\mathfrak{W}}$ *is such that:*

- $[\![\emptyset]\!]_{\mathfrak{W}} =$ ""*;*
- $[\![\varsigma]\!]_{\mathfrak{W}}(w) = aw$*;*
- $[\![\star]\!]_{\mathfrak{W}}(u, v) = uv$*.*

Let Σ be a signature. The set $\mathcal{T}_\Sigma$ of *ground terms* (or 0-order terms) is defined inductively as follows.

DEFINITION 9.2.3 (GROUND TERMS) *The set of* ground terms $\mathcal{T}_\Sigma$ *is the smallest set such that:*

- *if* $c \in \Sigma_0$ *then* $c \in \mathcal{T}_\Sigma$*;*
- *if* $f \in \Sigma_n$ *and* $t_i \in \mathcal{T}_\Sigma$ *for* $i = 1, \ldots, n$*, then* $f(t_1, \ldots, t_n) \in \mathcal{T}_\Sigma$*.*

Note that for $\mathcal{T}_\Sigma$ to be well-defined, it is necessary that $\Sigma_0 \neq \emptyset$.

EXAMPLE 9.2.2 *Considering again the signature* Σ *of Example 9.2.1, the following are examples of terms in* $\mathcal{T}_\Sigma$*:* $\emptyset$*,* $\varsigma(\emptyset)$*,* $\star(\emptyset, \varsigma(\emptyset))$*,* $\varsigma(\star(\emptyset, \varsigma(\emptyset)))$*, etc...*

For a term t in $\mathcal{T}_\Sigma$, the *denotation* of t in a Σ-interpretation $\mathfrak{I}$ is $[\![t]\!]_\mathfrak{I}$, given by the function $[\![\_]\!]_\mathfrak{I} : \mathcal{T}_\Sigma \to \mathcal{D}_\mathfrak{I}$ defined as follows:

- if $t = c \in \Sigma_0$, then $[\![t]\!]_\mathfrak{I} = [\![c]\!]_\mathfrak{I}$;
- if $t = f(t_1, \ldots, t_n)$, then $[\![t]\!]_\mathfrak{I} = [\![f]\!]_\mathfrak{I}\Big([\![t_1]\!]_\mathfrak{I}, \ldots, [\![t_n]\!]_\mathfrak{I}\Big)$.

EXAMPLE 9.2.3 *Considering again the signature Σ of Example 9.2.1 and the interpretations $\mathfrak{N}$ and $\mathfrak{W}$ defined there, the following are the denotations in either interpretations of some terms in $\mathcal{T}_\Sigma$:*

t	$[\![t]\!]_\mathfrak{N}$	$[\![t]\!]_\mathfrak{W}$
$\emptyset$	0	`""`
$\varsigma(\emptyset)$	1	`"a"`
$\varsigma(\varsigma(\emptyset))$	2	`"aa"`
$\star(\emptyset, \varsigma(\emptyset))$	1	`"a"`
$\varsigma(\star(\varsigma(\varsigma(\emptyset)), \varsigma(\emptyset)))$	4	`"aaaa"`

There is one particular Σ-interpretation that is worth noticing, whose domain is $\mathcal{T}_\Sigma$ itself! Namely, the interpretation $\mathfrak{T}_\Sigma$ where $\mathcal{D}_{\mathfrak{T}_\Sigma} = \mathcal{T}_\Sigma$ and $[\![\_]\!]_{\mathfrak{T}_\Sigma}$ is such that:

$$[\![f]\!]_{\mathfrak{T}_\Sigma}(t_1, \ldots, t_n) = f(t_1, \ldots, t_n) \tag{9.1}$$

for any $f \in \Sigma_n, (n \geq 0)$, and $t_i \in \mathcal{T}_\Sigma$ for $i = 1, \ldots, n$. Clearly, $[\![f]\!]_{\mathfrak{T}_\Sigma} : (\mathcal{D}_{\mathfrak{T}_\Sigma})^n \to \mathcal{D}_{\mathfrak{T}_\Sigma}$ is an n-ary operation on $\mathcal{D}_{\mathfrak{T}_\Sigma}$ as prescribed, and therefore $\mathfrak{T}_\Sigma$ is a *bona fide* Σ-interpretation. It is called the *free* interpretation (also the *syntactic* interpretation, or *initial* interpretation). Note that in the free interpretation the denotation of a term is simply itself.

Let $\mathcal{V}$ be a countably infinite set of symbols, called *variables*, assumed distinct from Σ (*i.e.*, $\mathcal{V} \cap \Sigma = \emptyset$).

The set of *first-order terms* over Σ and $\mathcal{V}$ is the set $\mathcal{T}_{\Sigma,\mathcal{V}}$ defined as follows:[1]

DEFINITION 9.2.4 (FIRST-ORDER TERMS) *The set of* (first-order) terms $\mathcal{T}_{\Sigma,\mathcal{V}}$ *is the smallest set such that:*

- *if $X \in \mathcal{V}$ then $X \in \mathcal{T}_{\Sigma,\mathcal{V}}$;*
- *if $c \in \Sigma_0$ then $c \in \mathcal{T}_{\Sigma,\mathcal{V}}$;*

[1]To distinguish between variable symbols functors, we will follow the convention of using symbols starting with uppercase letters for variables, while functors will be starting with lowercase letters.

- *if $f \in \Sigma_n$ and $t_i \in \mathcal{T}_{\Sigma,\mathcal{V}}$ for $i = 1, \ldots, n$, then $f(t_1, \ldots, t_n) \in \mathcal{T}_{\Sigma,\mathcal{V}}$.*

EXAMPLE 9.2.4 *Given the signature Σ such that $f \in \Sigma_3$, $h \in \Sigma_2$, $k \in \Sigma_1$, and $a \in \Sigma_0$, and given that W, X, Y, and Z are variables in $\mathcal{V}$, the terms $f(Z, h(Z, W), k(W))$ and $f(k(X), h(Y, k(a)), Y)$ are in $\mathcal{T}_{\Sigma,\mathcal{V}}$.*

Given a term $t \in \mathcal{T}_{\Sigma,\mathcal{V}}$, we define the set $\mathbf{V}(t)$ of variables of t as follows:

- if $t = X \in \mathcal{V}$, then $\mathbf{V}(t) = \{X\}$;
- if $t = c \in \Sigma_0$, then $\mathbf{V}(t) = \emptyset$;
- if $t = f(t_1, \ldots, t_n)$, then $\mathbf{V}(t) = \bigcup_{i=1}^{n} \mathbf{V}(t_i)$.

Clearly $\mathcal{T}_\Sigma$ is the subset of $\mathcal{T}_{\Sigma,\mathcal{V}}$ containing all terms without variables. That is,

$$\mathcal{T}_\Sigma = \{t \in \mathcal{T}_{\Sigma,\mathcal{V}} \mid \mathbf{V}(t) = \emptyset\}.$$

DEFINITION 9.2.5 (VALUATION) *Let $\mathfrak{I}$ be a Σ-interpretation. An $\mathfrak{I}$-valuation α is a function mapping variables to elements in the domain of interpretation; i.e., $\alpha : \mathcal{V} \to \mathcal{D}_\mathfrak{I}$.*

For a given $\mathfrak{I}$-valuation α and a term t in $\mathcal{T}_{\Sigma,\mathcal{V}}$ the denotation $[\![t]\!]^\alpha_\mathfrak{I}$ of t is given by the function $[\![\_]\!]^\alpha_\mathfrak{I} : \mathcal{T}_{\Sigma,\mathcal{V}} \to \mathcal{D}_\mathfrak{I}$ defined as follows:

- if $t = X \in \mathcal{V}$, then $[\![t]\!]^\alpha_\mathfrak{I} = \alpha(X)$;
- if $t = c \in \Sigma_0$, then $[\![t]\!]^\alpha_\mathfrak{I} = [\![c]\!]_\mathfrak{I}$;
- if $t = f(t_1, \ldots, t_n)$, then $[\![t]\!]^\alpha_\mathfrak{I} = [\![f]\!]_\mathfrak{I}\Big([\![t_1]\!]^\alpha_\mathfrak{I}, \ldots, [\![t_n]\!]^\alpha_\mathfrak{I}\Big)$.

To lighten notation, whenever the context makes clear which interpretation is being used, we may omit specifying $\mathfrak{I}$ and write only $[\![\_]\!]$ or $[\![\_]\!]^\alpha$ instead of $[\![\_]\!]_\mathfrak{I}$ or $[\![\_]\!]^\alpha_\mathfrak{I}$, and only "valuation" instead of "$\mathfrak{I}$-valuation."

9.2.2 Substitutions

Let t and t' be two terms in $\mathcal{T}_{\Sigma,\mathcal{V}}$, and let $X \in \mathcal{V}$. Then, the term $t^{X \leftarrow t'}$ is the term obtained from t by replacing all occurrences, if any, of the variable X in t by t'. Formally,

$$t^{X \leftarrow t'} = \begin{cases} t' & \text{if } t = X; \\ t & \text{if } t \in \mathcal{V} \text{ and } t \neq X; \\ c & \text{if } t = c \in \Sigma_0; \\ f(t_1^{X \leftarrow t'}, \ldots, t_n^{X \leftarrow t'}) & \text{if } t = f(t_1, \ldots, t_n), n \geq 1. \end{cases} \tag{9.2}$$

A *substitution* is a finitely non-identical assignment of terms to variables; *i.e.*, a function σ from $\mathcal{V}$ to $\mathcal{T}_{\Sigma,\mathcal{V}}$ such that the set $\{X \in \mathcal{V} \mid X \neq \sigma(X)\}$ is finite. This set is called the *domain* of σ and denoted by $\mathbf{dom}(\sigma)$. Such a substitution is also written as a set such as $\sigma = \{t_i/X_i\}_{i=1}^n$ where $\mathbf{dom}(\sigma) = \{X_i\}_{i=1}^n$ and $\sigma(X_i) = t_i$ for $i = 1$ to n. The set $\{t_i\}_{i=1}^n$ is called the *range* of σ and is denoted as $\mathbf{ran}(\sigma)$.

A substitution σ is uniquely extended to a function $\overline{\sigma}$ from $\mathcal{T}_{\Sigma,\mathcal{V}}$ to $\mathcal{T}_{\Sigma,\mathcal{V}}$ as follows:

- $\overline{\sigma}(X) = \sigma(X)$, if $X \in \mathcal{V}$;
- $\overline{\sigma}(a) = a$, if $a \in \Sigma_0$;
- $\overline{\sigma}(f(t_1, \ldots, t_n)) = f(\overline{\sigma}(t_1), \ldots, \overline{\sigma}(t_n))$, if $f \in \Sigma_n$, $t_i \in \mathcal{T}_{\Sigma,\mathcal{V}}$, $(1 \leq i \leq n)$.

Since they coincide on $\mathcal{V}$, and for notation convenience, we deliberately confuse a substitution σ and its extension $\overline{\sigma}$. Also, rather than writing $\sigma(t)$, we shall write $t\sigma$. Given two substitutions $\sigma = \{t_i/X_i\}_{i=1}^n$ and $\theta = \{s_j/Y_j\}_{j=1}^m$, their composition $\sigma\theta$ is the substitution which yields the same result on all terms as first applying σ then applying θ on the result. One computes such a composition as the set:

$$\sigma\theta = \Big(\{t\theta/X \mid t/X \in \sigma\} - \{X/X \mid X \in \mathbf{dom}(\sigma)\}\Big) \cup \Big(\theta - \{s/Y \mid Y \in \mathbf{dom}(\sigma)\}\Big).$$

Note that $\mathbf{1}_{\mathcal{V}}$, the identity function on $\mathcal{V}$, is a substitution. In set form, it is simply the empty set $\emptyset$; therefore, it is called the *empty substitution*. The set of substitutions is augmented with a special element $\perp$, called the *failing* substitution, such that:

$$\perp\sigma = \sigma\perp = \perp$$

for any substitution σ.

EXAMPLE 9.2.5 *If $\sigma = \{f(Y)/X, U/V\}$ and $\theta = \{b/X, f(a)/Y, V/U\}$, then composing σ and θ yields*

$$\sigma\theta = \{f(f(a))/X, f(a)/Y, V/U\}$$

while composing θ and σ gives

$$\theta\sigma = \{b/X, f(a)/Y, U/V\}.$$

Composition defines a preorder (*i.e.*, a reflexive and transitive relation) on substitutions. A substitution σ is *more general* than a substitution θ (written $\sigma \preceq \theta$) iff there exists a substitution ρ such that $\theta = \sigma\rho$.

EXAMPLE 9.2.6 $\{f(Y)/X\}$ *is more general than* $\{f(f(a))/X, f(a)/Y\}$.

DEFINITION 9.2.6 (VARIABLE RENAMING) *A* (variable) renaming ρ *is an injective substitution such that* $\mathbf{ran}(\sigma) \subseteq \mathcal{V}$.

The set form of a renaming is $\{Y_i/X_i\}_{i=1}^n$ such that $Y_i \neq Y_j$ for $i \neq j$. Clearly, if $\rho = \{Y_i/X_i\}_{i=1}^n$ is a renaming, then so is its inverse $\rho^{-1} = \{X_i/Y_i\}_{i=1}^n$, and $\rho\rho^{-1} = \rho\rho^{-1} = \emptyset$.

It is not difficult to see that if $\sigma \preceq \theta$ and $\theta \preceq \sigma$ then σ and θ are essentially the same substitutions *"up to renaming"* some variables and we write $\theta \sim \sigma$. Formally, $\sigma \preceq \theta$ and $\theta \preceq \sigma$, iff there is a renaming ρ such that $\sigma = \theta\rho$ and $\theta = \sigma\rho^{-1}$. It easy to verify that $\sim$ is an equivalence relation on substitutions.

DEFINITION 9.2.7 (GROUND SUBSTITUTION) *A* ground substitution σ *is a substitution such that* $\mathbf{ran}(\sigma) \subseteq \mathcal{T}_\Sigma$.

Let us denote the set of all ground substitutions as $\mathcal{S}_\Sigma$.

DEFINITION 9.2.8 (GROUND EXTENSION) *For any substitution σ, the* ground extension *of σ is the set* $\mathbf{ext}(\sigma)$ *of all ground substitutions that are less general than σ. That is,*

$$\mathbf{ext}(\sigma) = \{\theta \in \mathcal{S}_\Sigma \mid \sigma \preceq \theta\}.$$

9.2.3 Unification

Let t_1 and t_2 be two terms. If there exists a subsitution $\sigma \neq \bot$ such that $t_1\sigma = t_2\sigma$, then t_1 and t_2 are said to *unify,* and σ is called a *unifier* of t_1 and t_2. If t_1 and t_2 unify, then there exists a *most general unifier,* up to variable renaming, of t_1 and t_2 $\text{MGU}(t_1, t_2)$.

An *equation* is a pair of terms, written $s = t$. A substitution σ is a *solution* (or a *unifier*) of a set of equations $\{s_i = t_i\}_{i=1}^n$ iff $s_i\sigma = t_i\sigma$ for all $i = 1, \ldots, n$. Two sets of equations are *equivalent* iff they both admit *all* and *only* the same solutions. Following [43], we define two transformations on sets of equations—*term decomposition* and *variable elimination.* They both preserve solutions of sets of equations.

Let $\mathcal{E}$ be a set of equations.

- *Term Decomposition*

  If $\mathcal{E}$ contains an equation of the form $f(s_1, \ldots, s_n) = f(t_1, \ldots, t_n)$, where $f \in \Sigma_n$, $(n \geq 0)$, then the set $\mathcal{E}' = \mathcal{E} - \{f(s_1, \ldots, s_n) = f(t_1, \ldots, t_n)\} \cup \{s_i = t_i\}_{i=1}^n$ is equivalent to $\mathcal{E}$.[2]

[2]If $n = 0$, the equation is simply deleted.

- *Variable Elimination*

  If $\mathcal{E}$ contains an equation of the form $X = t$ where $t \neq X$, then the set $\mathcal{E}' = (\mathcal{E} - \{X = t\})\sigma \cup \{X = t\}$ where $\sigma = \{t/X\}$, is equivalent to $\mathcal{E}$.

A set of equations $\mathcal{E}$ is partitioned into two subsets: its *solved* part and its *unsolved* part. The solved part is its maximal subset of equations of the form $X = t$ such that X occurs nowhere in the full set of equations except as the left-hand side of this equation alone. The unsolved part is the complement of the solved part. A set of equations is said to be *fully solved* iff its unsolved part is empty.

Following is a unification algorithm. It is a non-deterministic normalization procedure for a given set $\mathcal{E}$ of equations which repeatedly chooses and performs one of the following transformations until none applies or failure is encountered.

$(u.1)$ Select any equation of the form $t = X$ where t is not a variable, and rewrite it as $X = t$.

$(u.2)$ Select any equation of the form $X = X$ and erase it.

$(u.3)$ Select any equation of the form $f(s_1, \ldots, s_n) = g(t_1, \ldots, t_m)$ where $f \in \Sigma_n$ and $g \in \Sigma_m, (n, m \geq 0)$; if $f \neq g$ or $n \neq m$, stop with failure; otherwise, if $n = 0$ erase the equation, else $(n \geq 1)$ replace it with n equations $s_i = t_i, (i = 1, \ldots, n)$.

$(u.4)$ Select any equation of the form $X = t$ where X is a variable which occurs somewhere else in the set of equations and such that $t \neq X$. If t is of the form $f(t_1, \ldots, t_n)$, where $f \in \Sigma_n$, and if X occurs in t, then stop with failure; otherwise, let $\sigma = \{t/X\}$ and replace every other equation $l = r$ by $l\sigma = r\sigma$.

If this procedure terminates with success, the set of equations which emerges as the outcome is fully solved. Its solved part defines a *most general unifier*, up to varaible renaming, of all the terms participating as sides of equations in $\mathcal{E}$. If it terminates with failure, the set of equations $\mathcal{E}$ is unsatisfiable and no unifier for it exists.

EXAMPLE 9.2.7 *The set* $\mathcal{E} = \{p(Z, h(Z, W), f(W)) = p(f(X), h(Y, f(a)), Y)\}$ *is normalized as follows:*

$\{Z = f(X)\ ,\ h(Z, W) = h(Y, f(a))\ ,\ f(W) = Y\}$ [*by* $(u.3)$]

$\{Z = f(X)\ ,\ h(f(X), W) = h(Y, f(a))\ ,\ f(W) = Y\}$ [*by* $(u.4)$]

$\{Z = f(X)\ ,\ f(X) = Y\ ,\ W = f(a)\ ,\ f(W) = Y\}$ [*by* $(u.3)$]

$\{Z = f(X)\ ,\ Y = f(X)\ ,\ W = f(a)\ ,\ f(W) = Y\}$ [*by* $(u.1)$]

$\{Z = f(X)\ ,\ Y = f(X)\ ,\ W = f(a)\ ,\ f(W) = f(X)\}$ [*by* $(u.4)$]

$$\{Z = f(X)\ ,\ Y = f(X)\ ,\ W = f(a)\ ,\ f(f(a)) = f(X)\} \qquad [\textit{by}\ (u.4)]$$

$$\{Z = f(X)\ ,\ Y = f(X)\ ,\ W = f(a)\ ,\ f(a) = X\} \qquad [\textit{by}\ (u.3)]$$

$$\{Z = f(X)\ ,\ Y = f(X)\ ,\ W = f(a)\ ,\ X = f(a)\} \qquad [\textit{by}\ (u.1)]$$

$$\{Z = f(f(a))\ ,\ Y = f(f(a))\ ,\ W = f(a)\ ,\ X = f(a)\} \qquad [\textit{by}\ (u.4)]$$

producing the substitution

$$\sigma = \{f(f(a))/Z, f(a)/W, f(f(a))/Y, f(a)/X\}$$

which is the MGU *of* $p(Z, h(Z, W), f(W))$ *and* $p(f(X), h(Y, f(a)), Y)$ *and both yield the same term*

$$p(f(f(a)), h(f(f(a)), f(a)), f(f(a)))$$

when applied the substitution σ.

Recursive unification

A recursive unification algorithm is given in Figure 9.1.

This algorithm assumes that a term is represented as a type TERM which is a union type (or supertype) of two types: VARIABLE and NON-VARIABLE. The type NON_VARIABLE is a record structure with fields for the functor (a string), the arity (an integer), and the subterms (an array of terms). That is,

```
type NON_VARIABLE = structure
                    {  STRING functor;
                       INT arity;                         (9.3)
                       TERM subterms[ ];
                    }
```

We assume that a boolean function *is_variable*(TERM t) is defined that returns **true** if t is of type VARIABLE and **false** if t is of type NON-VARIABLE.

The algorithm of Figure 9.1 also assumes that a substitution is represented by an abstract data type SUBSTITUTION that implements all the necessary operations (*i.e.*, composition and application).

Efficient unification

A more efficient unification algorithm is given in Figure 9.3. It is based on the UNION/FIND method [2].

```
SUBSTITUTION function $mgu$ (TERM $t_1$, TERM $t_2$)
  begin
    return $unify(t_1, t_2, \emptyset)$;
  end
```

```
SUBSTITUTION function $unify$ (TERM $t_1$, TERM $t_2$, SUBSTITUTION $\sigma$)
  begin
    TERM $u_1 \leftarrow t_1\sigma$;
    TERM $u_2 \leftarrow t_2\sigma$;
    if $is\_variable(u_1)$ then
      if $occurs\_in(u_1, u_2)$ then return $\bot$
      else return $\sigma\{u_2/u_1\}$;

    if $is\_variable(u_2)$ then
      if $occurs\_in(u_2, u_1)$ then return $\bot$
      else return $\sigma\{u_1/u_2\}$;

    if $u_1.functor = u_2.functor$ and $u_1.arity = u_2.arity$ then
      begin
        SUBSTITUTION $\theta \leftarrow \sigma$;
        INT $i \leftarrow 1$;
        while $i \leq u_1.arity$ and $\theta \neq \bot$ do
          begin
            $\theta \leftarrow unify(u_1.subterms[i], u_2.subterms[i], \theta)$;
            $i \leftarrow i + 1$;
          end
        return $\theta$;
      end
    else return $\bot$;
  end
```

Figure 9.1: A recursive unification algorithm

```
BOOLEAN function occurs_in (VARIABLE $v$, TERM $t$)
  begin
    BOOLEAN $found \leftarrow$ false;
    if $is\_variable(t)$ then $found \leftarrow (t = v)$
    else
      begin
        INT $i \leftarrow 1$;
        while $i \leq t.arity$ and $\neg found$ do
          begin
            $found \leftarrow occurs\_in(v, t.subterms[i])$
            $i \leftarrow i + 1$;
          end
      end
    return $found$;
  end
```

Figure 9.2: Variable occurrence test

Besides the fact that the algorithm of Figure 9.1 is recursive, its source of inefficiency is the repeated applications and compositions of substitutions. For these operations, the representation of a substitution as a set of term/variable pairs will necessitate association table lookups and various set operations. All this can be avoided by eliminating the explicit representation of substitution. Instead, a variable will store the term to which it is bound in its own structure. The type VARIABLE is thus modified as follows:

```
type VARIABLE = structure
                  { STRING name;
                    TERM binding;                    (9.4)
                  }
```

where *binding* is initialized to the variable itself to indicate that the variable is unbound.

With this representation, accessing a variable always necessitates following a possible chain of bindings until an unbound variable or a non-variable term is found. This operation, called *variable dereference* is given in Figure 9.4. Using variable dereference, there is no need to compose or apply substitutions.

The unification algorithm of Figure 9.3 uses this together with an iterative control to achieve better performance than the algorithm in Figure 9.1

```
BOOLEAN function $unify$ (TERM $t_1$, TERM $t_2$)
  begin
    STACK $stack \leftarrow \emptyset$;
    $push(t_1, stack)$;
    $push(t_2, stack)$;
    BOOLEAN $success \leftarrow$ true;
    while $stack \neq \emptyset$ and $success$ do
      begin
        TERM $u_1 \leftarrow deref(pop(stack))$;
        TERM $u_2 \leftarrow deref(pop(stack))$;
        if $u_1 \neq u_2$ then
          if $is\_variable(u_1)$ then
            if $occurs\_in(u_1, u_2)$ then $success \leftarrow$ false
            else $u_1.binding \leftarrow u_2$
          else
          if $is\_variable(u_2)$ then
            if $occurs\_in(u_2, u_1)$ then $success \leftarrow$ false
            else $u_2.binding \leftarrow u_1$
          else
          if $u_1.functor = u_2.functor$ and $u_1.arity = u_2.arity$ then
            for INT $i \leftarrow 1$ until $u_1.arity$ do
              begin
                $push(u_1.subterms[i], stack)$;
                $push(u_2.subterms[i], stack)$;
              end
          else $success \leftarrow$ false;
      end
    return $success$;
  end
```

Figure 9.3: A more efficient unification algorithm

```
TERM function $deref$ (TERM $t$)
  begin
    TERM $u \leftarrow t$;
    while $is\_variable(u)$ and $u.binding \neq u$ do
      $u \leftarrow u.binding$;
    return $u$;
  end
```

Figure 9.4: Variable dereference

```
BOOLEAN function $occurs\_in$ (VARIABLE $v$, TERM $t$)
  begin
    STACK $stack \leftarrow \emptyset$;
    $push(t, stack)$;
    BOOLEAN $found \leftarrow$ false;
    while $stack \neq \emptyset$ and $\neg found$ do
      begin
        TERM $u \leftarrow deref(pop(stack))$;
        if $is\_variable(u)$ then $found \leftarrow (u = v)$
        else
          for INT $i \leftarrow 1$ until $u.arity$ do
          $push(u.subterms[i], stack)$;
      end
    return $found$;
  end
```

Figure 9.5: Variable occurrence test with variable dereference

9.3 Predicate Calculus

9.3.1 Syntax

Let $\Sigma = \biguplus_{n \geq 0} \Sigma_n$ and $\Pi = \biguplus_{n \geq 0} \Pi_n$ be two signatures, and $\mathcal{V}$ a set of variables. The symbols in Π will be referred to as “predicate” symbols.

We define the set Φ of *well-formed formulae* over $\Sigma, \Pi, \mathcal{V}$.

DEFINITION 9.3.1 (WELL-FORMED FORMULA) *A* well-formed formula *(*“wff” *for short) is an element of* Φ*, the smallest set such that:*

1. $\mathfrak{true} \in \Phi$*;*
2. $\mathfrak{false} \in \Phi$*;*
3. *if* $p \in \Pi_n$ *and* $t_i \in \mathcal{T}_{\Sigma,\mathcal{V}}$ *for* $i = 1, \ldots, n$*, then* $p(t_1, \ldots, t_n) \in \Phi$*;*
4. *if* $\phi \in \Phi$*, then* $\neg\phi \in \Phi$*;*
5. *if* $\phi \in \Phi$ *and* $\phi' \in \Phi$*, then* $\phi \wedge \phi' \in \Phi$*;*
6. *if* $\phi \in \Phi$ *and* $\phi' \in \Phi$*, then* $\phi \vee \phi' \in \Phi$*;*
7. *if* $X \in \mathcal{V}$ *and* $\phi \in \Phi$*, then* $\exists X.\phi \in \Phi$*;*
8. *if* $X \in \mathcal{V}$ *and* $\phi \in \Phi$*, then* $\forall X.\phi \in \Phi$.

Wffs of the form 1–4 are called *atomic* formulae. A wff of the form 4 is called a *(positive) literal*. A wff of the form $\neg p(t_1, \ldots, t_n)$ is called a *negative literal*,

The symbols $\neg$, $\wedge$, and $\vee$ are called *(logical) connectives* and are pronounced, respectively, *“not,” “and,”* and *“or.”* They are also called, respectively, *negation*, *conjunction* and *disjunction*. The symbols $\exists$ and $\forall$ are called *(logical) quantifiers* and are pronounced, respectively, *“there exists”* and *“for all”*. For $\exists X.\phi$ and $\forall X.\phi$, the wff ϕ is called the *(quantification) scope* of X. For $\neg\phi$, the wff ϕ is called the *scope of negation*.

We could define syntactically $\wedge$ in terms of $\vee$ and $\neg$, or alternatively define $\vee$ in terms of $\wedge$ and $\neg$, by using one of:

- $\phi \wedge \phi'$ iff $\neg(\neg\phi \vee \neg\phi')$;
- $\phi \vee \phi'$ iff $\neg(\neg\phi \wedge \neg\phi')$.

Similarly, $\exists$ could be defined syntactically in terms of $\forall$ and $\neg$, or alternatively $\forall$ in terms of $\exists$ and $\neg$, by using one of:

- $\exists X.\phi$ iff $\neg(\forall X.(\neg\phi))$;
- $\forall X.\phi$ iff $\neg(\exists X.(\neg\phi))$.

These four syntactic identities above are known as *DeMorgan's laws*.

Alternatively, rather than taking the DeMorgan's syntactic identities as primitives, we will see that defining a wff as we did in Definition (9.3.1) we can provide a direct semantics to all the given connectives and quantifiers (as opposed to defining some syntactically from others) thereby *proving* DeMorgan's laws and other syntactic identities as *theorems*.[3]

We will use two other logical connectives by defining them in terms of those introduced in the syntax of Definition (9.3.1):

- implication: $\phi \rightarrow \phi'$ (ϕ *implies* ϕ') is defined as $\neg\phi \vee \phi'$;
- logical equivalence: $\phi \leftrightarrow \phi'$ (ϕ *is logically equivalent to* ϕ') is defined as $(\phi \rightarrow \phi') \wedge (\phi' \rightarrow \phi)$.

Given a wff ϕ, we define the set $\mathbf{FV}(\phi)$ of *free variables* of ϕ as follows:

- $\mathbf{FV}(\mathfrak{true}) = \emptyset$;
- $\mathbf{FV}(\mathfrak{false}) = \emptyset$;
- $\mathbf{FV}(p(t_1, \ldots, t_n)) = \bigcup_{i=1}^{n} \mathbf{V}(t_i)$;
- $\mathbf{FV}(\neg\phi) = \mathbf{FV}(\phi)$;
- $\mathbf{FV}(\phi \wedge \phi') = \mathbf{FV}(\phi) \cup \mathbf{FV}(\phi')$;
- $\mathbf{FV}(\phi \vee \phi') = \mathbf{FV}(\phi) \cup \mathbf{FV}(\phi')$;
- $\mathbf{FV}(\exists X.\phi) = \mathbf{FV}(\phi) - \{X\}$;
- $\mathbf{FV}(\forall X.\phi) = \mathbf{FV}(\phi) - \{X\}$.

DEFINITION 9.3.2 (FREE VARIABLE) *A variable X is said to* occur free *in a wff ϕ iff $X \in \mathbf{FV}(\phi)$.*

DEFINITION 9.3.3 (LOGICAL SENTENCE) *A* (logical) sentence *ϕ is a wff such that $\mathbf{FV}(\phi) = \emptyset$.*

[3] See Definition (9.3.5), Theorem (9.3.1) and Theorem (9.3.2).

Logical sentences are also called *closed formulae*.

Let ϕ be a wff, let t be a term in $\mathcal{T}_{\Sigma,\mathcal{V}}$, and let $X \in \mathcal{V}$. Then, the wff $\phi^{X\leftarrow t}$ is the wff obtained from ϕ by replacing all *free* occurrences, if any, of the variable X in ϕ by t. Formally,

$$\phi^{X\leftarrow t} = \begin{cases} \phi & \text{if } \phi = \mathfrak{true} \text{ or } \phi = \mathfrak{false}; \\ p(t_1^{X\leftarrow t}, \ldots, t_n^{X\leftarrow t}) & \text{if } \phi = p(t_1, \ldots, t_n), n \geq 1; \\ \neg\psi^{X\leftarrow t} & \text{if } \phi = \neg\psi; \\ \phi_1^{X\leftarrow t} \wedge \phi_2^{X\leftarrow t} & \text{if } \phi = \phi_1 \wedge \phi_2; \\ \phi_1^{X\leftarrow t} \vee \phi_2^{X\leftarrow t} & \text{if } \phi = \phi_1 \vee \phi_2; \\ \psi & \text{if } \phi = \exists X.\psi \text{ or } \phi = \forall X.\psi. \end{cases} \tag{9.5}$$

9.3.2 Semantics

It is possible to give a formal meaning to wffs relying on natural language and the intuitive understanding that we (humans) have of the words "true" and "false." Many texts in Formal Logic do so, and then use the meaning attributed to syntactic logical formulae by such a linguistic semantics in formal metalogical proofs (*e.g.*, soundness, completeness, compactness, *etc*.). However, such proofs are typically inductions on the syntactic structure of formulae and therefore often long-winded and rather mechanical—indeed, tedious. Thus, this style of metalogical proofs is known as *syntactic* or *proof-theoretic*.

Another way of defining the formal meaning of wffs does not rely directly on the informal linguistic notion of truth and falsity, but defines such notions by means of formal mathematics—typically Set Theory. This way of attributing a semantics to logical formulae specifies the meaning of a wff as a well-defined mathematical object through a denotational mapping and therefore allows all formal metalogical reasoning to be carried out in the semantic universe. This style of metalogical proofs is known as *semantic* or *model-theoretic*, and the proofs are typically more immediate and direct (as opposed to inductive) since they do no involve syntactic objects, but use instead their abstract mathematical meanings. Such a style of semantics was proposed by the mathematician and logician Alfred Tarski, and is the approach we will follow.

DEFINITION 9.3.4 (Σ, Π-INTERPRETATION) *A* Σ, Π-interpretation $\mathfrak{I} = \langle \mathcal{D}_{\mathfrak{I}}, [\![\_]\!]_{\mathfrak{I}} \rangle$ *is a* Σ*-interpretation such that the denotation function* $[\![\_]\!]_{\mathfrak{I}}$ *is extended to* Π *by mapping predicates of arity* n *to* n*-ary relations on the domain. Namely,*

$$[\![p]\!]_{\mathfrak{I}} \subseteq \mathcal{D}_{\mathfrak{I}}^n, \quad \textit{for any } p \in \Pi_n.$$

In order to build a formal notion of truth not defined in terms of our informal intuition, *we will characterize the meaning of a formula as a set*. Intuitively, because a formula may have free variables, its meaning will be *the set of all variable valuations* that allow it to make “consistent” sense in some formal way.

For example, let us assume that we have a unary predicate symbol: *human*. Also, let us assume that the domain of interpretation is the set of all living things on Earth. Then, *human*(X) will make sense only for those variable valuations that map X to a human being. We can thus define the *meaning* or *denotation* $[\![human(X)]\!]$ to be the set of all valuations that map X to a human being. Assume now that we also have a binary predicate *older*. Now, the wff $human(X) \wedge older(X, Y)$ will make sense only for valuations that map X to a human being (*e.g.*, *John Doe*) and Y to another (*e.g.*, *Lisa McIntosh*) such that John Doe is older than Lisa McIntosh. In other words, the meaning $[\![human(X) \wedge older(X, Y)]\!]$ of the wff *human*$(X)\wedge$*older*(X, Y) is simply obtained as the *set intersection* of the meanings $[\![human(X)]\!]$ and $[\![older(X, Y)]\!]$ of the two parts of the formula. This is the essence of the Tarskian-style of semantics that we present below.

Let $\mathfrak{I}$ be a Σ, Π-interpretation.

DEFINITION 9.3.5 (SEMANTICS OF WFFS) *The denotation* $[\![\phi]\!]_{\mathfrak{I}}$ *of a wff* ϕ *is defined by a mapping* $[\![\_]\!]_{\mathfrak{I}} : \Phi \rightarrow 2^{\mathcal{V} \rightarrow \mathcal{D}_{\mathfrak{I}}}$ *as follows:*[4]

- $[\![\mathfrak{true}]\!]_{\mathfrak{I}} \ = \ \mathcal{V} \rightarrow \mathcal{D}_{\mathfrak{I}}$;
- $[\![\mathfrak{false}]\!]_{\mathfrak{I}} \ = \ \emptyset$;
- $[\![p(t_1, \ldots , t_n)]\!]_{\mathfrak{I}} \ = \ \{\alpha \mid \langle [\![t_1]\!]^{\alpha}_{\mathfrak{I}}, \ldots , [\![t_n]\!]^{\alpha}_{\mathfrak{I}} \rangle \in [\![p]\!]_{\mathfrak{I}}\}$;
- $[\![\neg\phi]\!]_{\mathfrak{I}} \ = \ \overline{[\![\phi]\!]_{\mathfrak{I}}}$;
- $[\![\phi \wedge \phi']\!]_{\mathfrak{I}} \ = \ [\![\phi]\!]_{\mathfrak{I}} \cap [\![\phi']\!]_{\mathfrak{I}}$;
- $[\![\phi \vee \phi']\!]_{\mathfrak{I}} \ = \ [\![\phi]\!]_{\mathfrak{I}} \cup [\![\phi']\!]_{\mathfrak{I}}$;
- $[\![\exists X.\phi]\!]_{\mathfrak{I}} \ = \ \bigcup_{d \in \mathcal{D}_{\mathfrak{I}}} \{\alpha \mid \alpha^{X \rightarrow d} \in [\![\phi]\!]_{\mathfrak{I}}\}$;
- $[\![\forall X.\phi]\!]_{\mathfrak{I}} \ = \ \bigcap_{d \in \mathcal{D}_{\mathfrak{I}}} \{\alpha \mid \alpha^{X \rightarrow d} \in [\![\phi]\!]_{\mathfrak{I}}\}$.

Note that the meanings of the syntactic symbols $\mathfrak{true}$ and $\mathfrak{false}$ are not defined in terms of natural language. Rather, $\mathfrak{true}$ denotes the set of all possible valuations, and $\mathfrak{false}$ the empty set.

EXAMPLE 9.3.1 *Consider a set of two elements* a *and* b*, and two predicates such that* $p(a)$ *holds but not* $p(b)$*, and both* $q(a)$ *and* $q(b)$ *hold. This is expressed formally in our setting as having* $\Pi_1 = \{p, q\}$ *and an interpretation* $\mathfrak{A}$ *with* $\mathcal{D}_{\mathfrak{A}} = \{a, b\}$ *and* $[\![\_]\!]_{\mathfrak{A}}$ *such that:*

- $[\![p]\!]_{\mathfrak{A}} = \{a\}$;

[4]The notation $f^{a \rightarrow b}$ in defined in Equation (A.1) on Page 182.

- $[\![q]\!]_{\mathfrak{A}} = \{a, b\}$.

The meaning of $\phi = \exists X.p(X)$ *in* $\mathfrak{A}$ *is given by Definition (9.3.5) as follows:*

$$\begin{aligned}
[\![\phi]\!]_{\mathfrak{A}} &= \textstyle\bigcup_{d\in\mathcal{D}_{\mathfrak{A}}}\{\alpha \mid \alpha^{X\to d} \in [\![p(X)]\!]_{\mathfrak{A}}\}\\
&= \textstyle\bigcup_{d\in\{a,b\}}\{\alpha \mid \alpha^{X\to d} \in [\![p(X)]\!]_{\mathfrak{A}}\}\\
&= \{\alpha \mid \alpha^{X\to a} \in [\![p(X)]\!]_{\mathfrak{A}}\} \,\cup\, \{\alpha \mid \alpha^{X\to b} \in [\![p(X)]\!]_{\mathfrak{A}}\}\\
&= \{\alpha \mid \alpha^{X\to a} \in \{\beta \mid \beta(X) = a\}\} \,\cup\, \{\alpha \mid \alpha^{X\to b} \in \{\beta \mid \beta(X) = a\}\}\\
&= \{\alpha \mid \alpha^{X\to a}(X) = a\} \,\cup\, \{\alpha \mid \alpha^{X\to b}(X) = a\}\\
&= (\mathcal{V} \to \mathcal{D}_{\mathfrak{A}}) \,\cup\, \emptyset\\
&= [\![\mathfrak{true}]\!]_{\mathfrak{A}}.
\end{aligned}$$

The meaning of $\phi = \forall X.p(X)$ *in* $\mathfrak{A}$ *is given by:*

$$\begin{aligned}
[\![\phi]\!]_{\mathfrak{A}} &= \textstyle\bigcap_{d\in\mathcal{D}_{\mathfrak{A}}}\{\alpha \mid \alpha^{X\to d} \in [\![p(X)]\!]_{\mathfrak{A}}\}\\
&= \textstyle\bigcap_{d\in\{a,b\}}\{\alpha \mid \alpha^{X\to d} \in [\![p(X)]\!]_{\mathfrak{A}}\}\\
&= \{\alpha \mid \alpha^{X\to a} \in [\![p(X)]\!]_{\mathfrak{A}}\} \,\cap\, \{\alpha \mid \alpha^{X\to b} \in [\![p(X)]\!]_{\mathfrak{A}}\}\\
&= \{\alpha \mid \alpha^{X\to a} \in \{\beta \mid \beta(X) = a\}\} \,\cap\, \{\alpha \mid \alpha^{X\to b} \in \{\beta \mid \beta(X) = a\}\}\\
&= \{\alpha \mid \alpha^{X\to a}(X) = a\} \,\cap\, \{\alpha \mid \alpha^{X\to b}(X) = a\}\\
&= (\mathcal{V} \to \mathcal{D}_{\mathfrak{A}}) \,\cap\, \emptyset\\
&= [\![\mathfrak{false}]\!]_{\mathfrak{A}}.
\end{aligned}$$

The meaning of $\phi = \forall X.q(X)$ *in* $\mathfrak{A}$ *is given by:*

$$\begin{aligned}
[\![\phi]\!]_{\mathfrak{A}} &= \textstyle\bigcap_{d\in\mathcal{D}_{\mathfrak{A}}}\{\alpha \mid \alpha^{X\to d} \in [\![q(X)]\!]_{\mathfrak{A}}\}\\
&= \textstyle\bigcap_{d\in\{a,b\}}\{\alpha \mid \alpha^{X\to d} \in [\![q(X)]\!]_{\mathfrak{A}}\}\\
&= \{\alpha \mid \alpha^{X\to a} \in [\![q(X)]\!]_{\mathfrak{A}}\} \,\cap\, \{\alpha \mid \alpha^{X\to b} \in [\![q(X)]\!]_{\mathfrak{A}}\}\\
&= \{\alpha \mid \alpha^{X\to a} \in \{\beta \mid \beta(X) \in \mathcal{D}_{\mathfrak{A}}\}\} \,\cap\, \{\alpha \mid \alpha^{X\to b} \in \{\beta \mid \beta(X) \in \mathcal{D}_{\mathfrak{A}}\}\}\\
&= \{\alpha \mid \alpha^{X\to a} \in \mathcal{V} \to \mathcal{D}_{\mathfrak{A}}\} \,\cap\, \{\alpha \mid \alpha^{X\to b} \in \mathcal{V} \to \mathcal{D}_{\mathfrak{A}}\}\\
&= (\mathcal{V} \to \mathcal{D}_{\mathfrak{A}}) \,\cap\, (\mathcal{V} \to \mathcal{D}_{\mathfrak{A}})\\
&= [\![\mathfrak{true}]\!]_{\mathfrak{A}}.
\end{aligned}$$

Note that all that has been presented for the first-order predicate logic is also valid for *propositional* logic. Indeed, if we limit Π only to Π_0 (*i.e.*, $\Pi_n = \emptyset$ for $n \geq 1$), then we obtain a propositional calculus by eliminating quantified wffs. There is also no need for a term algebra $\mathcal{T}_{\Sigma,\mathcal{V}}$ since the only possible literals are $p \in \Pi_0$. Recall that a nullary relation is either $\emptyset$ or $\mathbf{1}$. Therefore, the semantics of a propositional predicate p is either that of $\mathfrak{false}$ if p denotes $\emptyset$, or that of $\mathfrak{true}$ if p denotes $\mathbf{1}$. Hence, the "truth value" of a propositional formula is obtained as either $[\![\mathfrak{true}]\!]$ or $[\![\mathfrak{false}]\!]$. The propositional calculus is thus a simple degenerate case of the first-order predicate calculus.

Clearly, because a logical sentence does not have free variables, it follows from Definition (9.3.5) that the meaning of a sentence is always equal either to the meaning of $\mathfrak{true}$ or to the meaning of $\mathfrak{false}$. This is an unambiguous and rigorous definition of *logical truth* which allows us to refer to a logical sentence as being true or false irrespective of what "true" or "false" mean in natural language.

Note that this notion of logical truth depends on the particular interpretation structure $\mathfrak{I}$ where denotations take their meanings. Namely,

DEFINITION 9.3.6 (SATISFACTION) *An interpretation $\mathfrak{I}$ is said to* satisfy *a wff ϕ (written as $\models_{\mathfrak{I}} \phi$) iff $[\![\phi]\!]_{\mathfrak{I}} = [\![\mathfrak{true}]\!]_{\mathfrak{I}}$.*

An interpretation $\mathfrak{I}$ is said to satisfy a set of wffs Γ (written as $\models_{\mathfrak{I}} \Gamma$) if it satisfies *every* wff in Γ.

DEFINITION 9.3.7 (TAUTOLOGICAL EQUIVALENCE) *Two wffs ϕ and ϕ' are said to be* tautologically equivalent *(written as $\phi \vdash\!\dashv \phi'$) iff $[\![\phi]\!]_{\mathfrak{I}} = [\![\phi']\!]_{\mathfrak{I}}$ for all interpretations $\mathfrak{I}$.*

DEFINITION 9.3.8 (VALID WFF) *A wff ϕ is said to be* valid *(written as $\models \phi$) iff it is tautologically equivalent to $\mathfrak{true}$.*

Note that the well-known DeMorgan's identities are obtained as consequences of the semantics of wffs.

THEOREM 9.3.1 (DEMORGAN'S EQUIVALENCES) *The following are tautological equivalences:*

- $\neg(\phi \wedge \phi') \vdash\!\dashv \neg\phi \vee \neg\phi'$;
- $\neg(\phi \vee \phi') \vdash\!\dashv \neg\phi \wedge \neg\phi'$;
- $\neg\exists X.\phi \vdash\!\dashv \forall X.(\neg\phi)$;
- $\neg\forall X.\phi \vdash\!\dashv \exists X.(\neg\phi)$.

PROOF This is left as exercise.

Other familiar syntactic identities are obtained as consequences of the semantics of wffs. In the following, $\mathfrak{Q}$ stands for either $\exists$ or $\forall$, and $\diamond$ stands for either $\wedge$ or $\vee$.

THEOREM 9.3.2 (SYNTACTIC IDENTITIES) *The following are tautological equivalences:*

- $\neg\mathfrak{true} \vdash\!\dashv \mathfrak{false}$;
- $\neg\mathfrak{false} \vdash\!\dashv \mathfrak{true}$;
- $\neg\neg\phi \vdash\!\dashv \phi$;
- $\phi \wedge \mathfrak{true} \vdash\!\dashv \phi$;
- $\phi \wedge \mathfrak{false} \vdash\!\dashv \mathfrak{false}$;
- $\phi \vee \mathfrak{true} \vdash\!\dashv \mathfrak{true}$;
- $\phi \vee \mathfrak{false} \vdash\!\dashv \phi$;
- $\phi \diamond \phi' \vdash\!\dashv \phi' \diamond \phi$;
- $(\phi \diamond \phi') \diamond \phi'' \vdash\!\dashv \phi \diamond (\phi' \diamond \phi'')$;
- $\phi \wedge (\phi' \vee \phi'') \vdash\!\dashv (\phi \wedge \phi') \vee (\phi \wedge \phi'')$;
- $\phi \vee (\phi' \wedge \phi'') \vdash\!\dashv (\phi \vee \phi') \wedge (\phi \vee \phi'')$;
- $\exists X.(\phi \vee \phi') \vdash\!\dashv (\exists X.\phi) \vee (\exists X.\phi')$;
- $\forall X.(\phi \wedge \phi') \vdash\!\dashv (\forall X.\phi) \wedge (\forall X.\phi')$;
- $\mathfrak{Q}X.\mathfrak{Q}Y.\phi \vdash\!\dashv \mathfrak{Q}Y.\mathfrak{Q}X.\phi$;
- $\mathfrak{Q}X.\phi \vdash\!\dashv \phi$ *if* $X \notin \mathbf{FV}(\phi)$;
- $\mathfrak{Q}X.\phi \vdash\!\dashv \mathfrak{Q}Y.\phi^{X\leftarrow Y}$ *if* $Y \notin \mathbf{FV}(\phi)$;
- $\mathfrak{Q}X.(\phi \diamond \phi') \vdash\!\dashv (\mathfrak{Q}X.\phi) \diamond \phi'$ *if* $X \notin \mathbf{FV}(\phi')$.

PROOF This is left as exercise.

DEFINITION 9.3.9 (PRENEX FORMULA) *A wff is a* prenex *formula (or in prenex form) iff it does not contain any quantifiers, or if it is of the form* $\mathfrak{Q}X.\phi$ *where* ϕ *is a prenex formula.*

In other words, a prenex formula has all its quantifiers, if any, occur at the outset. Theorem (9.3.2) can be used to show that any wff is equivalent to a prenex formula.

9.3.3 Logical inference

DEFINITION 9.3.10 (LOGICAL CONSEQUENCE) *A wff ϕ is a* logical consequence *of a set of wffs Γ (written $\Gamma \vdash \phi$) iff, for every interpretation $\mathfrak{I}$, $\models_{\mathfrak{I}} \phi$ whenever $\models_{\mathfrak{I}} \Gamma$.*

In other words, $\Gamma \vdash \phi$ iff every interpretation satisfying Γ also satisfies ϕ.

DEFINITION 9.3.11 (SATISFIABILITY) *A wff is said to be* satisfiable *iff there is at least one interpretation $\mathfrak{I}$ such that $\models_{\mathfrak{I}} \phi$.*

If ϕ is satisfied in $\mathfrak{I}$ then there must be some $\mathfrak{I}$-valuation α in $[\![\phi]\!]_{\mathfrak{I}}$, and we write $\alpha \models_{\mathfrak{I}} \phi$.

Logical inference is the process of transforming a wff into another one in such a way as to preserve its satisfiability.

DEFINITION 9.3.12 (CANONICAL INTERPRETATION) *An interpretation $\mathfrak{I}$ is* canonical *iff whenever $\models_{\mathfrak{I}} \phi$ then $\models_{\mathfrak{J}} \phi$ for all interpretations $\mathfrak{J}$.*

THEOREM 9.3.3 (CANONICITY OF THE TERM ALGEBRA) *The Σ-algebra $\mathcal{T}_\Sigma$ is canonical.*

In other words, in order to establish whether a wff ϕ is satisfiable in *any* interpretation at all, it is sufficient to establish whether ϕ is satisfiable for a valuation mapping variables to ground terms.

Since any substitition σ defines a mapping from $\mathcal{V}$ to $\mathcal{T}_{\Sigma,\mathcal{V}}$, it also defines a set of $\mathcal{T}_\Sigma$-valuations $\sigma\theta$ in its ground extension which are obtained by "completing" σ by composing it with some ground substitution θ. We will exploit this later for resolution-refutation where it is sufficient to establish that a wff ϕ is satisfiable in $\mathcal{T}_\Sigma$ (and therefore in all other interpretations) by showing that $\neg\phi\sigma \vdash\!\dashv \mathfrak{false}$ for some (not necessarily ground) substitution.

Note that, although two similar quantifiers that follow one another may be swapped in a wff without changing the meaning of the wff (*e.g.*, $\forall X.\forall Y.\phi$ and $\forall Y.\forall X.\phi$ mean the same), such is *not* the case if the two quantifiers are different (*e.g.*, $\forall X.\exists Y.\phi$ and $\exists Y.\forall X.\phi$ do not have the same meaning!).

Consider a wff of the form $\forall X.\exists Y.\phi$: intuitively, the occurrence of the existential variable Y in the scope of the universal variable X establishes a *functional dependence* between X and Y. Indeed, given an interpretation $\mathfrak{I}$ such that $\models_{\mathfrak{I}} \forall X.\exists Y.\phi$, for any fixed element $d \in \mathcal{D}_{\mathfrak{I}}$, there will be a valuation $\alpha \in [\![\forall X.\exists Y.\phi]\!]_{\mathfrak{I}}$ such that $\alpha(X) = d$ (by definition of the meaning of $\forall$); and for any such $\alpha(X)$, there is a *uniquely* determined element $\alpha(Y)$ (by definition of the meaning of $\exists$). Therefore, this characterizes $\alpha(Y)$ as a *function* of $\alpha(X)$. Let $s_\alpha : \mathcal{D}_{\mathfrak{I}} \rightarrow \mathcal{D}_{\mathfrak{I}}$ be this function, and let us consider the new interpretation $\mathfrak{J}$ obtained from $\mathfrak{I}$ by adding a new element s in Σ_1 with denotation $[\![s]\!]_{\mathfrak{J}}$=s_α. Then, by construction, $\mathfrak{J}$ satisfies the wff $\forall X.\phi^{Y \leftarrow s(X)}$. Conversely, if $\mathfrak{J}$ satisfies the wff $\forall X.\phi^{Y \leftarrow s(X)}$, then, by Definition (9.3.5), $\mathfrak{I}$ must satisfy $\forall X.\exists Y.\phi$ (by using $\alpha(Y) = s_\alpha(\alpha(X))$ for any valuation α. In fact, this is true regardless of the specific function s_α: it can be any function in $\mathcal{D}_{\mathfrak{I}} \rightarrow \mathcal{D}_{\mathfrak{I}}$—what matters is that it provides a witness as the valuation of

Y as a function of an element valuating the universal variable X in whose scope Y appears. Such a witness function is called a *Skolem function*, after the logician Thoralf Skolem who originally introduced the concept.

The idea of using Skolem witness functions to eliminate existential variables can be expressed in general term as the following theorem.

THEOREM 9.3.4 (SKOLEMIZATION) *Let* $\phi = \forall X_1.\cdots\forall X_n.\exists X.\psi,\ (n \geq 0)$ *be a wff; then, for any interpretation* $\Im$*:* $\models_\Im \phi$ *iff* $\models_\Im \forall X_1.\cdots\forall X_n.\psi^{X\leftarrow s(X_1,\cdots,X_n)}$ *where* s *is a new* n*-ary function symbol.*

Note that for $n = 0$, Theorem (9.3.4) indicates that any valuation that satisfies the wff $\phi^{X\leftarrow c}$, where c is a new constant symbol, will also satisfy the wff $\exists X.\phi$.

Clausal form

Resolution is a logical inference rule that we will introduce later. It applies to wffs in a specific form called *clausal* form.

DEFINITION 9.3.13 (CLAUSE) *A* clause *is a wff consisting of a disjunction of (positive or negative) literals.*

Note that a clause does not contain any quantifiers.

DEFINITION 9.3.14 (CLAUSAL FORMULA) *A wff is a* clausal formula *(or in* clausal form*) iff it is a conjunction of clauses where no variable occurs in two different clauses.*

For convenience, a clausal formula is simply represented as a set of clauses.

Any wff can be put in clausal form by the repeated combined applications of DeMorgan's laws, the syntactic identities of Theorem (9.3.2), reduction to prenex form and skolemization. Figure 9.6 contains the complete set of rewrite rules that are needed to normalize any wff into an equivalent clausal formula. Figure 9.7 specifies a procedure using these rules which performs the following sequence of transformations, at each step repeatedly applying any rule mentioned between the square brackets until none applies.[5] The correctness of this procedure follows directly from the foregoing theorems. Termination is also easy to establish.

EXAMPLE 9.3.2 *Let us follow the procedure on the following wff:*

$$\forall X.\Big(p(X) \rightarrow \Big(\forall Y.(p(Y) \rightarrow p(f(X,Y))) \wedge \neg\forall Y.(q(X,Y) \rightarrow p(Y))\Big)\Big) \tag{9.22}$$

[5] All the rules in Figure 9.6 are to be applied *up to commutativity and associativity* of $\vee$ and $\wedge$.

$$\phi \leftrightarrow \phi' \implies (\phi \rightarrow \phi') \wedge (\phi' \rightarrow \phi) \tag{9.6}$$

$$\phi \rightarrow \phi' \implies \neg\phi \vee \phi' \tag{9.7}$$

$$\neg\neg\phi \implies \phi \tag{9.8}$$

$$\neg(\phi \vee \phi') \implies (\neg\phi) \wedge (\neg\phi') \tag{9.9}$$

$$\neg(\phi \wedge \phi') \implies (\neg\phi) \vee (\neg\phi') \tag{9.10}$$

$$\neg(\exists X.\phi) \implies \forall X.(\neg\phi) \tag{9.11}$$

$$\neg(\forall X.\phi) \implies \exists X.(\neg\phi) \tag{9.12}$$

$$\underline{\text{if } X \text{ occurs outside } \phi} : \quad \exists X.\phi \implies \exists Y.\phi^{X \leftarrow Y} \qquad (Y \text{ is new}) \tag{9.13}$$

$$\underline{\text{if } X \text{ occurs outside } \phi} : \quad \forall X.\phi \implies \forall Y.\phi^{X \leftarrow Y} \qquad (Y \text{ is new}) \tag{9.14}$$

$$\forall X_1.\cdots\forall X_n.\exists X.\phi \implies \forall X_1.\cdots\forall X_n.\phi^{X \leftarrow s(X_1,\cdots,X_n)} \qquad (s \text{ is new}) \tag{9.15}$$

$$(\forall X.\phi) \vee \phi' \implies \forall X.(\phi \vee \phi') \tag{9.16}$$

$$(\forall X.\phi) \wedge \phi' \implies \forall X.(\phi \wedge \phi') \tag{9.17}$$

$$\forall X.\phi \implies \phi \tag{9.18}$$

$$\phi \vee (\phi_1 \wedge \phi_2) \implies (\phi \vee \phi_1) \wedge (\phi \vee \phi_2) \tag{9.19}$$

$$\underline{\text{if } X \in \mathbf{V}(\phi')} : \quad \phi \wedge \phi' \implies \phi^{X \leftarrow Y} \wedge \phi' \qquad (Y \text{ is new}) \tag{9.20}$$

$$\phi_1 \wedge \cdots \wedge \phi_n \implies \{\phi_1, \ldots, \phi_n\} \tag{9.21}$$

Figure 9.6: Rules for Reduction to Clausal Normal Form

1. ***Eliminate non-primitive connectives*** [Rules (9.6) and (9.7)]
2. ***Reduce the scope of negations*** [Rules (9.8)–(9.12)]
3. ***Standardize quantified variables*** [Rules (9.13) and (9.14)]
4. ***Eliminate existential quantifiers*** [Rule (9.15)]
5. ***Convert to prenex form*** [Rules (9.16) and (9.17)]
6. ***Eliminate universal quantifiers*** [Rule (9.18)]
7. ***Reduce to conjunctive normal form*** [Rule (9.19)]
8. ***Rename variables*** [Rule (9.20)]
9. ***Eliminate conjunctions*** [Rule (9.21)]

Figure 9.7: Procedure for Reduction to Clausal Normal Form

1. *Eliminate non-primitive connectives:*

$$\forall X.\Big(\neg p(X) \vee \big(\forall Y.(\neg p(Y) \vee p(f(X,Y))) \wedge \ \neg\forall Y.(\neg q(X,Y) \vee p(Y))\big)\Big)$$

2. *Reduce the scope of negations:*

$$\forall X.\Big(\neg p(X) \vee \big(\forall Y.(\neg p(Y) \vee p(f(X,Y))) \wedge \ \exists Y.(q(X,Y) \wedge \neg p(Y))\big)\Big)$$

3. *Standardize quantified variables:*

$$\forall X.\Big(\neg p(X) \vee \big(\forall Y.(\neg p(Y) \vee p(f(X,Y))) \wedge \ \exists Z.(q(X,Z) \wedge \neg p(Z))\big)\Big)$$

4. *Eliminate existential quantifiers:*

$$\forall X.\Big(\neg p(X) \vee \big(\forall Y.(\neg p(Y) \vee p(f(X,Y))) \wedge \ (q(X,g(X)) \wedge \neg p(g(X)))\big)\Big)$$

5. *Convert to prenex form:*

$$\forall X.\forall Y.\Big(\neg p(X) \vee \big(\neg p(Y) \vee p(f(X,Y)) \wedge\ (q(X,g(X)) \wedge \neg p(g(X)))\big)\Big)$$

6. *Eliminate universal quantifiers:*

$$\neg p(X) \vee \big(\neg p(Y) \vee p(f(X,Y)) \wedge\ (q(X,g(X)) \wedge \neg p(g(X)))\big)$$

7. *Reduce to conjunctive normal form:*

$$\big(\neg p(X) \vee \neg p(Y) \vee p(f(X,Y))\big) \wedge \big(\neg p(X) \vee q(X,g(X))\big) \wedge \big(\neg p(X) \vee \neg p(g(X))\big)$$

8. *Rename variables:*

$$\big(\neg p(X_1) \vee \neg p(Y) \vee p(f(X_1,Y))\big) \wedge \big(\neg p(X_2) \vee q(X_2,g(X_2))\big) \wedge \big(\neg p(X_3) \vee \neg p(g(X_3))\big)$$

9. *Eliminate conjunctions:*

$$\{\ \neg p(X_1) \vee \neg p(Y) \vee p(f(X_1,Y)),\ \ \neg p(X_2) \vee q(X_2,g(X_2)),\ \ \neg p(X_3) \vee \neg p(g(X_3))\ \}$$

The procedure terminates with a clausal formula which is equivalent to the original wff (9.22).

Resolution

Let p be a nullary predicate, and let us consider the clausal formula $\{p, \neg p\}$. Clearly, this wff is the same as $p \wedge \neg p$, which can be easily shown to be tautologically equivalent to $\mathfrak{false}$. For convenience, we shall write $\mathfrak{false}$ as $\emptyset$ and call it the *empty clause*.

Let us now consider the clause $\Gamma = \{p \vee \phi, \neg p \vee \phi'\}$. Written with full connectives, it is the same $(p \vee \phi) \wedge (\neg p \vee \phi')$. Using implications, it becomes $(\neg p \rightarrow \phi) \wedge (p \rightarrow \phi')$. Because p is either true or false, one of the two implications is necessarily true. Therefore, Γ is false whenever $\phi \vee \phi'$ is false. Thus, to check whether Γ is false it is sufficient to check that $\phi \vee \phi'$ is false.

With the same reasoning, we can see that $\Gamma = \{p \vee \phi, \neg p \vee \phi'\} \cup \Gamma'$ is false whenever the clausal formula $\{\phi \vee \phi'\} \cup \Gamma'$ is false.

Let q be an n-ary predicate and $\Gamma = \{q(\vec{t}\,) \vee \phi, \neg q(\,\vec{s}\,) \vee \phi'\} \cup \Gamma'$.[6] Because all variables in this clause are universally quantified, it must be satisfied in the term interpretation for all valuations of

[6]We use the notation $\vec{t}$ as a shorthand for a sequence $t_1, \ldots, t_n$ of 0 or more terms.

the variables. If $q(\vec{t}\,)$ and $q(\vec{s}\,)$ unify with MGU σ, they are called *complementary literals*. Then, any ground substitution in $\mathtt{ext}(\sigma)$ is a particular variable valuation which must also satisfy Γ. In other words, Γ is satisfiable if the clause $\Gamma\sigma$ is satisfiable. But, in $\Gamma\sigma$ the *same* literal $q(\vec{t}\,)\sigma$ occurs (like p above) both positively and negatively. Therefore, following the same reasoning, to establish that Γ is false, it is sufficient to establish that this is also the case for the clause obtained when both complementary literals are eliminated from $\Gamma\sigma$; namely, $(\{\phi \vee \phi'\} \cup \Gamma'\})\sigma$.

This justifies the correctness of the following general inference rule.

DEFINITION 9.3.15 (RESOLUTION) Resolution *is an inference rule transforming a clausal formula* Γ *(i.e., a set of clauses) into a clausal formula* Γ' *using the following transformation:*

- *let* ϕ *and* ϕ' *be two clauses in* Γ*;*
- *let* ℓ *a* positive *literal in* ϕ *and* $\neg\ell'$ *a* negative *literal in* ϕ' *such that* $\mathrm{MGU}(\ell, \ell') = \sigma$ *and* $\sigma \neq \bot$*;*
- $\Gamma' = \left(\left(\Gamma - \{\phi, \phi'\}\right) \cup \left\{(\phi - \{\ell\}) \vee (\phi' - \{\neg\ell'\})\right\}\right)\sigma.$

The set of clauses Γ being transformed by resolution is called the *resolvent*. The pair of complementary literals used in performing a resolution transformation is called the *resolving pair*. Performing a resolution transformation is also called *resolving* the clause.

Resolution is in fact many familiar inference rules in disguise.

$$\begin{array}{rcll}
\phi \wedge (\phi \rightarrow \psi) & \vdash & \psi & (\textit{Modus Ponens}) \\
(\phi \rightarrow \psi) \wedge \neg\psi & \vdash & \neg\phi & (\textit{Modus Tollens}) \\
(\phi \rightarrow \psi) \wedge (\psi \rightarrow \xi) & \vdash & \phi \rightarrow \xi & (\text{Cut}) \\
(\phi \rightarrow \psi) \wedge (\neg\phi \rightarrow \xi) & \vdash & \psi \vee \xi & (\text{Join}) \\
\phi \wedge \neg\phi & \vdash & \mathfrak{false} & (\text{Contradiction})
\end{array}$$

Note that the resolution rule does not transform a clausal formula into an equivalent formula. Rather, it transforms it into a formula whose falsity implies that of the original one. This is because unifying a pair of complementary literals selects particular valuations, which only allows to conclude that if such a valuation falsifies the transformed wff, then it will also falsify the original formula. In other words, if ϕ is a clausal formula and ϕ' is derived from ϕ by resolution then, for any wff Γ, in order to estabish whether $\Gamma \vdash \phi$, it is sufficient to establish $\Gamma \wedge \neg\phi \vdash \mathfrak{false}$. Thus, to establish that a wff ϕ is a logical consequence of a set of wffs Γ it is sufficient to establish that the empty clause $\emptyset$ can be derived by repeated application of the resolution rule to the clausal form of

$\Gamma \wedge \neg\phi$. This, also known as *proof by contradiction*, is the essence of the proof procedure known as *resolution-refutation*.

The general provability of wffs in the first-order predicate calculus is summed up in the two following results.

THEOREM 9.3.5 (UNDECIDABILITY OF FIRST-ORDER PREDICATE CALCULUS)
There is no procedure that permits, for any sentence ϕ, to establish effectively whether $\models \phi$ in finitely many steps.

THEOREM 9.3.6 (SEMI-DECIDABILITY OF FIRST-ORDER PREDICATE CALCULUS)
There is a procedure that permits, for any sentence ϕ, to establish effectively whether $\not\models \phi$ in finitely many steps.

PROOF The non-deterministic procedure that consists of applying all possible resolution transformations to the clausal form of ϕ will produce $\emptyset$ if and only if $\not\models \phi\sigma$, for some substitution σ .

In other words, Theorems 9.3.5 and 9.3.6 say that it is not possible in general for a computer to prove a theorem, but it is possible for it to disprove one. However, the complexity of the task is in general quite great. The resolution procedure being non-deterministic, it turns out that the difficulty will lie in devising a correct *proof strategy* to choose among many potential resolving pairs.

EXAMPLE 9.3.3 *Let us use resolution refutation to prove that if all men are human, and all humans are mortal, and if Socrates is a man, then Socrates is mortal. We will denote Socrates by socrates* $\in \Sigma_0$*, the "man" relation by man* $\in \Pi_1$*, the "human" relation by human* $\in \Pi_1$*, and the "mortal" relation by mortal* $\in \Pi_1$*. Then, we define the wffs:*

$$\begin{aligned}
\phi_1 &= \forall X.(\mathit{man}(X) \to \mathit{human}(X))\\
\phi_2 &= \forall X.(\mathit{human}(X) \to \mathit{mortal}(X))\\
\phi_3 &= \mathit{man}(\mathit{socrates})\\
\Gamma &= \phi_1 \wedge \phi_2 \wedge \phi_3
\end{aligned}$$

from which to derive:

$$\phi = \mathit{mortal}(\mathit{socrates}).$$

In order to establish $\Gamma \vdash \phi$*, we will resolve the clausal form of the wff* $\Gamma \wedge \neg\phi$*. This clausal formula* *is:*

$$\begin{aligned}
&\{\ \neg man(X_1) \vee human(X_1)\ , && (9.23)\\
&\ \ \neg human(X_2) \vee mortal(X_2)\ , && (9.24)\\
&\ \ man(socrates)\ , && (9.25)\\
&\ \ \neg mortal(socrates)\ \}. && (9.26)
\end{aligned}$$

The resolving pair $\langle human(X_1), \neg human(X_2)\rangle$ between Clauses (9.23) and (9.24), with the substitution $\{X_1/X_2\}$, yields the resolvent:

$$\begin{aligned}
&\{\ \neg man(X_1) \vee mortal(X_1)\ , && (9.27)\\
&\ \ man(socrates)\ , && (9.28)\\
&\ \ \neg mortal(socrates)\ \}. && (9.29)
\end{aligned}$$

Next, we can resolve Clause (9.27) and Clause (9.29) on $\langle mortal(X_1), \neg mortal(socrates)\rangle$ with the substitution $\{socrates/X_1\}$, yielding the resolvent:

$$\begin{aligned}
&\{\ \neg man(socrates)\ , && (9.30)\\
&\ \ man(socrates)\ \}. && (9.31)
\end{aligned}$$

This contains only one complementary pair with the empty substitution. Resolving this yields the empty clause. Therefore, the proof is completed.

Prolog

Logic programming languages, of which Prolog [16, 40, 59, 46] is *the* most popular representative, express programs as relational rules of the form:

$$r_0(\vec{t}_0)\ \texttt{:-}\ r_1(\vec{t}_1), \ldots, r_n(\vec{t}_n). \qquad (9.32)$$

where the r_i's are relationals symbols and the $\vec{t}_i$'s are tuples of first-order terms. This syntax is in fact a variation of the implication:

$$r_1(\vec{t}_1) \wedge \ldots \wedge r_n(\vec{t}_n) \rightarrow r_0(\vec{t}_0). \qquad (9.33)$$

This, in turn, is just the clause:

$$r_1(\vec{t}_1) \vee \ldots \vee r_n(\vec{t}_n) \vee \neg r_0(\vec{t}_0). \qquad (9.34)$$

One reads such a rule as: "*For all bindings of their variables, the terms $\vec{t}_0$ are in relation r_0 if the terms $\vec{t}_1$ are in relation r_1 and ... the terms $\vec{t}_n$ are in relation r_n.*" In the case where $n = 0$, the rule reduces to the simple unconditional assertion, or *fact*, $r_0(\vec{t}_0)$ that the terms $\vec{t}_0$ are in relation r_0. A fact will be written omitting the `:-` symbol. These rules are called *definite clauses*;[7] expressions such as $r_i(\vec{t}_i)$ are called *atoms*; the *head* of a definite clause is the atom on the left of the `:-` symbol, and its *body* is the conjunction of atoms on its right.

For example, the following are two definite clauses, the first one being a fact:

$$\begin{array}{l} \textit{conc}([], L, L). \\ \textit{conc}(H.T, L, H.R) \texttt{ :- } \textit{conc}(T, L, R). \end{array} \tag{9.35}$$

where '[]' $\in \Sigma_0$ is a constant and the function symbol '.' $\in \Sigma_2$ is written in infix notation. This may be used as a program to concatenate two lists where [] is used as a list terminator.[8]

A *query* is a clause of the form:

$$\texttt{:-}\ q_1(\vec{s}_1), \ldots, q_m(\vec{s}_m). \tag{9.36}$$

A query as shown above may be read: *"Does there exist some binding of variables such that the terms $\vec{s}_1$ are in relation q_1 and ... $\vec{s}_m$ are in relation q_m?"* To emphasize that this is interpreted as a question, the symbol `:-` is then written `?-` as in:[9]

$$\texttt{?-}\ q_1(\vec{s}_1), \ldots, q_m(\vec{s}_m). \tag{9.37}$$

SLD resolution is a non-deterministic deduction rule by which queries are transformed. It takes its origins in Automatic Theorem Proving based on the Resolution Principle discovered by J. Alan Robinson [51] and was proposed by Robert A. Kowalski [36] as a computation rule. Technically, it is characterized as linear resolution over definite clauses, using a selection function. Linear resolution is a particular strategy of the general resolution rule whereby that one single fixed clause keeps being transformed by resolving it against other clauses in a given set. SLD resolution is a further restriction of linear resolution where (1) the fixed clause is a query, (2) clauses in the set are definite, and (3) an oracular function selects which atom in the query to resolve on and which definite clause in the set to resolve against. Thus, the letters "SLD" stand respectively for "*Selection*," "*Linear*," and "*Definite*."

More specifically, using the above Prolog notation for queries and rules, SLD resolution consists in choosing an atom $q_i(\vec{s}_i)$ in the query's body and a definite clause in the given set whose head $r_0(\vec{t}_0)$ ***unifies*** with $q_i(\vec{s}_i)$ thanks to a variable substitution σ (*i.e.*, $q_i(\vec{s}_i)\sigma = r_0(\vec{t}_0)\sigma$), then replacing it by the body of that clause in the query, applying substitution σ to all the new query. That is,

$$\texttt{?-}\ q_1(\vec{s}_1)\sigma, \ldots, q_{i-1}(\vec{s}_{i-1})\sigma, r_1(\vec{t}_1)\sigma, \ldots, r_n(\vec{t}_n)\sigma, q_{i+1}(\vec{s}_{i+1})\sigma, \ldots, q_m(\vec{s}_m)\sigma. \tag{9.38}$$

[7] A *definite clause* is a clause that contains *at most one* negative literal.

[8] For example, $1.2.3.[]$ is a list. Edinburgh Prolog syntax uses $[X|Y]$ instead of $X.Y$; it also uses a simplified variant to express a list *in extenso*, allowing writing [1,2,3] rather than [1|[2|[3|[]]]].

[9] Prolog rules and queries are the only possible forms of definite clauses.

The process is repeated and stops when and if the query's body is empty (success) or no rule head unifies with the selected atom (failure). There are two non-deterministic choices made in the process: one of an atom to rewrite in the query and one among the potentially many rules whose head unifies with this atom. In any case, SLD resolution is *sound* (*i.e.*, it does not derive wrong solutions) and, provided these choices are made by a fair non-deterministic selection function, it is also *complete* (*i.e.*, it derives all solutions).

Prolog's computation rule is a particular deterministic approximation of SLD resolution. Specifically, it is a flattening of SLD resolution emulating a depth-first search. It sees a program as an *ordered* set of definite clauses, and a query or definite clause body as an *ordered* set of atoms. These orders are meant to provide a rigid guide for the two choices made by the selection function of SLD resolution. Thus, Prolog's particular computation strategy transforms a query by rewriting the query attempting to unify its leftmost atom with the head of the first rule according to the order in which they are specified. If failure is encountered, a backtracking step to the latest rule choice point is made, and computation resumed there with the next alternative given by the following rule. For example, if the two clauses for predicate *conc* are given as above, then the Prolog query '`?-` *conc*$(1.2.T, 3.4.[], L)$.' succeeds with the substitution $T = [], L = 1.2.3.4.[]$, while the query '`?-` *conc*$(1.2.[], X, 3.Y)$.' fails.

Strategies for choice of where to apply linear resolution are all logically consistent in the sense that if computation terminates, the variable binding exhibited is a legitimate solution to the original query. In particular, like non-deterministic SLD resolution, Prolog resolution is *sound*. However, unlike non-deterministic SLD resolution, it is *incomplete*. Indeed, Prolog's particular strategy of doing linear resolution may diverge although finitely derivable solutions to a query may exist. For example, if the definite clauses for *conc* are given in a different order (*i.e.*, first the rule, then the fact), then the query '`?-` *conc*(X, Y, Z).' never terminates although it has (infinitely many) finitely derivable solutions!