# Chapter 2

## TERMINOLOGY

In this chapter we provide definitions for the basic notions of clause logic and also introduce some more special terminology that we shall use throughout this monograph. Although we assume the reader to be familiar with the concept of resolution we review the fundamental definitions for the sake of clarity and completeness. Additional terminology will be introduced in later chapters whenever this helps the understanding of our definitions and proofs.

### 2.1 TERMS, LITERALS AND CLAUSES

Concerning the language of clause logic we assume that there is an infinite supply of *variable symbols V*, constant symbols CS, function symbols FS, and predicate symbols PS. As usual we assume each function and predicate symbol to be associated with some fixed arity which we denote by arity(F) for F in PS or FS. We call a predicate or function symbol unary iff it is of arity 1, binary iff the arity is 2, and in general n-place for arity n. The set of n- place function and predicate symbols is denoted by $FS_n$ and $PS_n$, respectively.

If S is some set of expressions, clauses or clause sets then CS(S), FS(S), and PS(S), refers to the set of constant, function and predicate symbols, respectively, that occur in S. (For a formal definition of occurrence see definition 2.12 below).

We define the notions term, atom, literal, expression and clause formally:

DEFINITION 2.1: A term is defined inductively as follows:

(i) Each variable and each constant is a term.

(ii) If $t_1, \dots, t_n$ are terms and f is an n-place function symbol, then $f(t_1, \dots, t_n)$ is also a term.

(iii) Nothing else is a term.

If a term t is of the form $f(t_1, \dots, t_n)$ we call it functional; the set of arguments of t - args(t) - is $\{t_1, \dots, t_n\}$; f is called the leading (function) symbol of t. The set of all terms is called T.

DEFINITION 2.2.: If $t_1, \dots, t_n$ are terms and P denotes an n-place predicate symbol then $A = P(t_1, \dots, t_n)$ is an atom; P is called the leading (predicate) symbol of A; args(A) is the set $\{t_1, \dots, t_n\}$.

DEFINITION 2.3.: A literal is either an atom or an atom preceded by a negation sign.

DEFINITION 2.4.: An expression is either a term or a literal.

DEFINITION 2.5.: A clause is a finite set of literals. The empty clause is denoted by $\square$.

Throughout this work we shall speak of classes of clause sets, by this we always mean sets of finite sets of clauses.

DEFINITION 2.6.: If a literal L is unsigned, i.e. if it is identical with an atom A, then the dual of L— $L^d$ — equals $\neg A$. Otherwise, if L is of the form $\neg A$ then $L^d = A$. For a set of literals $C = \{L_1, \dots, L_n\}$ we define $C^d = \{L_1^d, \dots, L_n^d\}$.

Additionally we introduce the following notation:

DEFINITION 2.7.: $C_+$ is the set of positive (unsigned) literals of a clause C, analogously $C_-$ denotes the set of negative literals (negated atoms) in C.

DEFINITION 2.8.: C is a Horn clause iff it contains at most one positive literal, i.e. $|C_+| \leq 1$.
A Horn clause C with $C_+ = C$ is called a fact, with $C_- = C$ a goal; if $C_+$ and $C_-$ are both nonempty then C is called a rule.

## 2. 2 TERM STRUCTURE

The term depth of an expression or a clause is defined as follows:

DEFINITION 2. 9.: The term depth of a term t - $\tau(t)$ - is defined by:

(i) If t is a variable or a constant, then $\tau(t) = 0$.

(ii) If $t = f(t_1, \dots, t_n)$, where F is an n-place function symbol, then $\tau(t) = 1 + \max\{\tau(t_i) \mid 1 \leq i \leq n\}$.

The term depth of a literal L is defined as $\tau(L) = \max\{\tau(t) \mid t \in args(L)\}$. The term depth of a clause C is defined as $\tau(C) = \max\{\tau(L) \mid L \in C\}$. For a set S of clauses we define $\tau(S) = \max\{\tau(C) \mid C \in S\}$.
It is convient to make use of some definitions concerning term structure that are well known form the term rewriting literature.

DEFINITION 2. 10.: Let A be an expression. Then the set of positions in A P(A) is a set of sequences of integers, defined as follows:

(i) $\varepsilon$ (i.e. the empty sequence) $\in P(A)$;

(ii) if $p \in P(t_i)$ then $i.p \in P(A)$ for $A = F(t_1, \dots, t_n)$, where F is an n-ary predicate symbol (possibly preceeded by a negation sign) or a function symbol and $1 \leq i \leq n$;

(iii) nothing else is in P(A).

The length of a position $|p|$ is the number of integers in the sequence. The number of subterms of A - s(A) - is defined to be $|P(A)|$.

DEFINITION 2. 11.: For any position p of an expression A the subterm of A at position p SUB(p, A) is defined as follows:

(i) $SUB(\varepsilon, A) = A$;

(ii) $SUB(i.p, A) = SUB(p, t_i)$, if $A = F(t_1, \dots, t_n)$ for some n-ary predicate symbol (possibly preceeded by a negation sign) or some function symbol F.

The depth of occurence of the subterm of position defined as $\tau_{sub}(p, A) = |p|$.

DEFINITION 2.12.: We say that an expression t occurs in an expression E, iff there is an i, s.t. t = SUB(i, E). Occasionally, we shall write E[t] to indicate that t is a proper subterm of E, i.e. that t occurs in E but $t \neq E$. We also say that a function or predicate symbol F occurs in E iff F is the leading symbol of some expression t that occurs in E.

The set of all variables occurring in E is called V(E); if C is a clause, then V(C) is the union over all $V(P_i)$ for all atoms $P_i$ in C.

We define $E_1$ and $E_2$ to be variable disjoint iff $V(E_1) \cap V(E_2) = \emptyset$.

By OCC(x, E) we denote the number of occurrences of a variable x in E, i.e. $OCC(x, E) = |\{i \mid SUB(i, E) = x\}|$. OCC(x, C) is defined analogously for clauses C.

DEFINITION 2.13.: An expression or a clause is called ground if no variables occur in it. We call it constant free if no constants occur in it, and function free if it does not contain function symbols.

EXAMPLES: If E = P(x, f(f(y))), then SUB($\varepsilon$, E) = P(x, f(f(y))), SUB(1.$\varepsilon$, E) = x, SUB(2.$\varepsilon$, E) = f(f(y)), SUB(2.1.$\varepsilon$, E) = f(y), and SUB(2.1.1.$\varepsilon$, E) = y; V(E) = {x, y}. OCC(x, E) = OCC(y, E) = 1. E is not ground, but constant free.

DEFINITION 2.14.: $\tau_{min}(t,E)$ is defined as the minimal depth of occurrence of a term t within an expression E, i.e.

$$\tau_{min}(t, E) = \min\{\tau_{SUB}(i, E)\} \mid SUB(i, E) = t\}.$$

If C is a clause, then $\tau_{min}(t, C)$ denotes the minimum of $\tau_{min}(t, P_i)$ for all atoms $P_i$ of C. $\tau_{max}(t, E)$ respectively $\tau_{max}(t, C)$ are defined in the same way.

EXAMPLES: If $P_1 = P(x, f(f(y)))$, $P_2 = Q(f(x))$ and $C = \{P_1, \neg P_2\}$, then $\tau(P_1) = 2$, $\tau(P_2 = 1$, $\tau(C) = 2$, $\tau_{SUB}(0, P_1) = \tau_{SUB}(0, P_2) = 0$, $\tau_{SUB}(1, P_1) = 0$, $\tau_{SUB}(3, P_1) = 1$, $\tau_{min}(x, C) = 0$, $\tau_{max}(x, C) = 1$, $\tau_{min}(y, C) = \tau_{max}(y, C) = 2$.

DEFINITION 2.15.: The maximal variable depth of an expression E is defined as $\tau_V(E) = \max\{\tau_{max}(x, E) \mid x \in V(E)\}$. For clauses C we define

$\tau_V(C) = \max\{\tau_V(L) \mid L \in C\}$;

analogously for clause sets S $\tau_V(S) = \max\{\tau_V(C) \mid C \in S\}$.

## 2.3 SUBSTITUTIONS

Another basic notion is the concept of substitution.

DEFINITION 2.16.: Let V be the set of variables and T be the set of terms. A substitution is a mapping $\sigma$: V to T s.t. $\sigma(x) = x$ almost everywhere. We call the set $\{x \mid \sigma(x) \neq x\}$ domain of $\sigma$ and denote it by dom($\sigma$), $\{\sigma(x) \mid x \in dom(\sigma)\}$ is called range of $\sigma$ (rġ($\sigma$)). By $\varepsilon$ we denote the empty substitution, i.e. $\varepsilon(x) = x$ for all variables x.

We shall occasionally specify a substitution as a (finite) set of expressions of the form $t_i / x_i$ with the intended meaning $\sigma(x_i) = t_i$.

DEFINITION 2.17.: We say that a substitution $\sigma$ is based on a clause set S iff no other constant and functions symbols besides that in CS(S) and FS(S), respectively, occur in the terms of rg($\sigma$).

A ground substitution is a substitution $\sigma$ s.t. there are only ground terms in rg($\sigma$).

The application of substitutions to expressions is defined as follows:

DEFINITION 2.18.: Let E be an expression and $\sigma$ a substitution.

(i) If E is a variable, then $E\sigma$ is $\sigma(E)$ (cf. definition 2.17).

(ii) If E is a constant, then $E\sigma = E$.

(iii) Otherwise E is of the form $F(t_1, \ldots, t_n)$, where F is either an p-place function or predicate symbol (possibly negated).

In this case $E\sigma = F(t_1\sigma, \ldots, t_n\sigma)$. If L is a literal, then $L\sigma$ is defined to be the application of $\sigma$ to the atom of L. If C is a set of expressions or a clause, then $C\sigma = \{E\sigma \mid E \in C\}$.

DEFINITION 2.19.: An expression $E_1$ is an instance of another expression $E_2$ iff there exists a substitution $\sigma$ s.t. $E_1 = E_2\sigma$. Likewise a clause $C_1$ is an instance of clause $C_2$ iff $C_1 = C_2\sigma$ for some substitution $\sigma$.

We may compare expressions, substitutions and clauses using the following ordering relation.

DEFINITION 2.21.: Let $E_1$ and $E_2$ be expressions, then $E_1 \leq_s E_2$ - read: $E_1$ is more general than $E_2$ - iff there exists a substitution $\sigma$ s.t. $E_1\sigma = E_2$. For substitutions $\rho$ and $\vartheta$ we define analogously: $\rho \leq_s \vartheta$ iff there exists a substitution $\sigma$ s.t. $\rho\sigma = \vartheta$. Similarly, if C and D are clauses, $C \leq_s D$ iff there exists a substitution $\sigma$ s.t. $C\sigma \subseteq D$. In this case we also say, in accordance with the usual resolution terminology, that C subsumes D.

DEFINITION 2.22.: A clause is called condensed if it does not subsume a proper subclause of itself.

EXAMPLE:
$\{P(x,a), P(a,x)\}$ is condensed; $C = \{P(x), P(a)\}$ is not condensed, because it subsumes $\{P(a)\}$ which is a subclause of C. For every clause C there is (up to renaming) a unique subclause D s.t. $C \leq_s D$ and D is condensed. We call D the condensation of C and denote it by $C_{cond}$.
Condensation is an important technique in Joyner's resolution decision procedures [Joy 76].

DEFINITION 2.23.: A set of expressions M is unifiable by a substitution $\sigma$ iff $E_i\sigma = E_j\sigma$ for all $E_i, E_j \in M$. $\sigma$ is called most general unifier (m.g.u.) of M iff for every other unifier $\rho$ of M: $\sigma \leq_s \rho$.

We shall also say that $E_1$ is unifiable with $E_2$ iff $\{E_1, E_2\}$ is unifiable.

Remember that any two different m.g.u.s of a set of expressions only differ in the names of the variables.

## 2.4 FACTORS AND RESOLVENTS

DEFINITION 2.24.: A factor of a clause C is a clause $C\vartheta$, where $\vartheta$ is a m.g.u. of some $C' \subseteq C$. In case $|C\vartheta| < |C|$ we call the factor non-trivial.

For the resolvent we retain the original definition of Robinson [Rob 65], which combines factorization and (binary) resolution. But be aware that, in some of the chapters to come, we shall locally define clauses and resolvents differently (namely as lists of literals). It will always be clear from the context which concept we are using.

DEFINITION 2.25.: If C and D are variable disjoint clauses and M and N are subsets of C and D respectively, s.t. $N^d \cup M$ is unifiable by the m.g.u. $\vartheta$, then $E = (C - M)\vartheta \cup (D - N)\vartheta$ is a (Robinson)-resolvent of C and D.

If M and N are singleton sets then E is called binary resolvent of C and D.

The atom A of $(N^d \cup M)\vartheta$ is called the resolved atom. We also say that E is generated via A. The elements of N and M are called the literals resolved upon.

DEFINITION 2.26.: For a clause set S we define Res(S) as the set of Robinson-resolvents of S. Additionally we define:

$R^0(S) = S$,

$R^{i+1}(S) = R^i(S) \cup Res(R^i(S))$, and

$R^*(S) = \bigcup_i R^i(S)$.

We say that a clause C is derivable from a clause set S iff $C \in R^*(S)$.

In the following chapters we shall introduce various refinements of Robinson's resolution procedure. By a refinement of resolution we mean an operator Res' s.t. $Res'(S) \subseteq Res(S)$ for all clause sets S. $R'^i$ and $R'^*$ are defined in the obvious way.

In contrast to resolution refinements we shall also define variants of resolution: For resolution variants we allow ordinary resolvents to be replaced by certain instances of it. This technique is also called saturation. (See especially chapter 5 for examples of this method).

## 2.5 A UNIFICATION ALGORITHM

Some critical parts of proofs in the following chapters demand for a careful tracing of the unification procedure. For this purpose we state a simple version of an unification algorithm. We first have to introduce some additional terminology, which will also prove useful in later sections.

DEFINITION 2.27.: Let $E_1$, $E_2$ be two expressions. The set of corresponding pairs $CORR(E_1, E_2)$ - is defined as follows:

(i) $(E_1, E_2) \in CORR(E_1, E_2)$.

(ii) If $(F_1, F_2) \in CORR(E_1, E_2)$ s.t. $F_1 = F(s_1, \ldots, s_n)$ and $F_2 = F(t_1, \ldots, t_n)$, for some function or predicate symbol F ( possibly preceded by a negation sign), then $(s_i, t_i) \in CORR(E_1, E_2)$.

(iii) Nothing else is in $CORR(E_1, E_2)$.

A pair $(F_1, F_2) \in CORR(E_1, E_2)$ is called irreducible iff the leading symbols of $F_1$ and $F_2$ are different. $(F_1, F_2)$ is called strongly irreducible iff it is irreducible and both, $F_1$ and $F_2$, are not variables.

We are now able to present a unification algorithm which finds a m.g.u. $\vartheta$ of two variable disjoint expressions $E_1$ and $E_2$, if there is one.

```
begin
  ϑ := ε;  i := 0;
  while E1ϑ ≠ E2ϑ do
    if CORR(E1ϑ, E2ϑ) contains a strongly irreducible pair then failure;
    if there is a pair (s, t) ∈ CORR(E1ϑ, E2ϑ) s.t.
       s ≠ t and either s or t is a variable then
       i := i + 1;
       if s is a variable then xi := s; ti := t;
       else {t is a variable}  xi := t; ti := s;
       endif
    else failure
    endif;
    if xi occurs in ti then failure
    else ϑ := ϑ{ti / xi}
    endif
  endwhile
  {ϑ is the m.g.u. of E1 and E2 }
end.
```

Remarks: Termination of this algorithm with failure means that there doesn't exist a m.g.u. of $E_1$ and $E_2$.

The substitution component $t_i/x_i$ is called the $i^{th}$ mesh substituent of $\vartheta$.

The presented algorithm is by no means well suited for actual computation but it fits nicely purposes of argumentation in some of our proofs.

## 2.6 SPLITTING

In later chapters, we sometimes refer to the well known splitting rule. To make the argumentation more precise we present a formal definition.

DEFINITION 2.28.: A clause C is called decomposed iff for all subsets C', C'' of C s.t. $C' \cup C'' = C$ and both, C' and C'', are nonempty:

$$V(C') \cap V(C'') \neq \emptyset.$$

Otherwise, if $V(C') \cap V(C'') = \emptyset$, we say that C can be decomposed into (split *components)* C' and C''.

DEFINITION 2.29.: For any clause set S let SPLIT(S) denote the set of clause sets obtained by splitting all members of S as far as possible. More accurately we define recursively:

(i) $\Sigma_0 = \{S\}$.

(ii) If for all $S' \in \Sigma_i$ and all $C \in S'$ C is decomposed then $\Sigma_{i+1} = \Sigma_i$.

(iii) If there is some $S' \in \Sigma_i$ s.t. for some $C \in S'$ C is can be decomposed into C' and C'' then

$$\Sigma_{i+1} = (\Sigma_i - \{S'\}) \cup ((S' - \{C\}) \cup \{C'\}) \cup ((S' - \{C\}) \cup \{C''\}).$$

(A proper C is chosen nondeterministically)

Now we may define $\text{SPLIT}(S) = \Sigma_k$ where $k = \min\{m \mid \Sigma_m = \Sigma_{m+1}\}$.

The splitting rule says that we have to apply resolution to all members of SPLIT(S) separately to test for the unsatisfiablity of S. (Recall that S is unsatisfiable iff S' is unsatisfiable for all S' ∈ SPLIT(S).)

## 2.7 HERBRAND SEMANTICS

For the semantics of clause logic we refer, as usual, to the terminological machinery developed by J. Herbrand. We review the basic definitions:

DEFINITION 2.30.: The Herbrand universe $H_S$ of a clause set S is the set of all ground terms s.t. only constant and function symbols in CS(S) and FS(S) occur in them. (If CS(S) is empty we introduce a special constant symbol to prevent $H_S$ from being empty).

DEFINITION 2.31.: The Herbrand tree $HT_S$ of a clause set S is a directed graph of the following structure: The vertices of $HT_S$ are identified with the elements of $H_S$; there is an edge from the vertex s to the vertex t ($s,t \in H_S$) iff $t \in args(s)$.
Observe that $HT_S$ does not contain directed cycles.

DEFINITION 2.32.: An Herbrand instance of an atom or a clause C in S is a ground instance $C\vartheta$ of C s.t. $\vartheta$ is based on S.

DEFINITION 2.33.: The Herbrand base $\hat{H}_S$ is the set of all Herbrand instances of atoms appearing in clauses of S.

DEFINITION 2.34.: An Herbrand interpretation $HI_S$ for a clause set S is a subset of $\hat{H}_S$ with the intended meaning that the truth value true is assigned to all elements of $HI_S$ and the truth value false is assigned to all atoms in $\hat{H}_S - HI_S$.

Remark: We shall also denote $HI_S$ as

$$\{A \mid A \in HI_S\} \cup \{\neg A \mid A \in \hat{H}_S - HI_S\}.$$

## 2.8 REMARKS

Of course, clause logic may be viewed just as a special syntactic frame for more "classic" formulations of first order predicate logic. We assume familiarity with such formulations and refer to the standard textbooks for such notions as first order formula, quantificational prefix, conjunctive normal form, prenex normal form etc..

Recall that the correspondence between first order formulas and clause sets is established via transformation of formulas to prenex normal form and Skolemization (i.e. eliminating existential quantifiers by substituting certain functional terms for the corresponding variables).

Algorithms for the transformation of a formula to some clause set which is equivalent w.r.t. satisfiability can be found in textbooks on automated theorem proving [CL 73].

Throughout this work we make use of the following naming conventions: For variable symbols we use letters from the end of the alphabet (u, v, w, x, y, z); for constant symbols, letters a, b, c are used; function symbols are denoted by f, g or h; as metavariables for terms we use t or s; capital letters will denote atoms, literals, clauses or certain sets of expressions. Whenever needed this letters are augmented by indices.