Direttore Responsabile: Ruggero Ferro
Iscrizione al Registro Stampa del Tribunale di Padova n. 1235 del 26.9.1990

Pubblicato con il contributo di:

Cassa di Risparmio di Padova e Rovigo

Stampa Veronese (Padova)
Laser Fotocomposizioni (Padova)

RESOLUTION THEOREM PROVING

by

ALEXANDER LEITSCH

Technische Universität Wien

AUSTRIA

Contents

Chapter 1

Introduction

Logical calculi can serve for many purposes, such as reconstructing and analyzing mathematical proofs in a formal manner or automatizing the finding of mathematical proofs. As the paradoxes of set theory struck the mathematical community around 1900, the formal and consistent representation of theories became a central issue in foundational research. In the theory of proofs breath–taking results have been obtained in the thirties of this century. They culminated in the completeness and incompleteness (or better "incompletability") results of Gödel [Göd30],[Göd31]. Somewhat later Gentzen defined a natural notion of formal proofs [Gen34], which is closer to actual mathematical deduction than the so called Hilbert–type systems. Like Herbrand [Her30] Gentzen investigated the form and structure of mathematical proofs, while Gödel's work was more directed to provability.

In general it was not the purpose of proof theory to develop inference systems for actual deduction or proof–finding: taking into account the complexity of relevant mathematical proofs this is hardly surprising.

As computers and programming languages developed, the problem of mathematical proof finding was one of the first attacked in the field called Artificial Intelligence today. A direct application of Herbrand's proof theoretical result was tried by Gilmore [Gil60] for performing automated deduction in first order predicate logic.

The idea behind was the following: Herbrand's theorem gave a method to reduce a predicate logic proof to a propositional one by constructing a propositional formula F' (out of a closed predicate logic formula F) which is derivable iff F is. Thus the following method is suggested: Try to find F'; for each "candidate" F' decide the validity of F'. Because inference in propositional logic is simpler than in predicate logic this method seems to be quite natural from the point of view of logical complexity. But from the computational point of view this method shows two serious defects: 1) The finding of F', 2) The costs of inference on F'.

Point 2) above was successfully attacked by Davis and Putnam in the same year [DP60], while the search for F' and also the (possibly enormous) size of F' remained serious obstacles. The method to apply proof theory (which was developed for other purposes) to automated deduction directly has clearly failed. Obviously there was a need to model some natural techniques of proof finding employed by humans within a logical calculus. The invention of such a technique is the characteristic of Robinson's famous paper [Rob65], which was a real landmark in automated deduction (and in some sense its very beginning). Particularly the consequent use of the unification principle created a substantially new type of logical calculus. The resolution principle

(a combination of a propositional cut rule and the substitutional unification principle) yielded a spectacular improvement in performance versus Gilmore's method. While the propositional part of the resolution rule (the atomic cut) was changed in various ways and even abandoned (in Bibel's connection method [Bib82]), the unification principle is part of every relevant computational proof calculus. Though (as indicated above) resolution is no longer the only method in automated deduction today, it is still playing a central role and some features (such as the unification principle) are common to all computational calculi.

The purpose of this course is to present the resolution calculus, some of its newest forms and variants and its applications. We will try to show that resolution is not only a powerful technique in automated deduction, but also a tool to address problems in mathematical logic (such as the decision problem of classes in first order logic). Although not sufficiently honoured by the community of logicians, the resolution calculus can give interesting insights and contributions to proof theory and to the theory of proof complexity. It is a firm belief of the author that (in the long run) methods of automated deduction will not only play a role in computer science (logic programming, program verification, expert systems etc.) but also in logical proof theory. Because of its importance to computer science, resolution theory branched into many different topics and applications. For this reason we had to make a selection and to omit important applications such as logic programming (which deserves treatment in an own special course). Special emphasis is laid to foundational issues such as decision theory and proof complexity. We will illustrate how the development of more powerful deduction techniques can be influenced by problems in mathematical logic and complexity theory.

In chapter 3 we present the basics of resolution theory by following the usual path of presentation. Chapter 4 is devoted to resolution refinements; here we try to explain refinements from a more abstract point of view and we develop a formalism for the decision theory in chapter 6. Instead of giving an exhaustive survey of the numerous refinements we concentrate on a few typical ones, illustrate the key ideas and classify them structurally. In chapter 5 we discuss deletion methods, where (not completely standard) techniques as condensing and clause implication are discussed. Chapter 6 shows how resolution methods can be adapted to decide first order classes. Starting from Joyner's results [Joy76] we present some recent results and techniques (such as decision generators and model building methods). In chapter 7 we present a logical complexity theory of resolution. It is shown that resolution proofs, compared to other logical calculi, can be very long (the relation may be nonelementary). The method of function introduction is presented, which is based on resolution but has a strong power of lemmatization. We discuss some variants and applications of this method and present some recent speed–up results, such as the nonelementary reduction of proof length by function introduction (based on Statman's famous example from combinatory logic).

Chapter 2

Terminology

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 constant 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 $t_1, \ldots, t_n$ are terms and f is an n–place function symbol, then $f(t_1, \ldots, t_n)$ is also a term.

(iii) Nothing else is a term.

If a term t is of form $f(t_1, \ldots, t_n)$ we call it *functional*; the set of arguments of t — $args(t)$ — is $\{t_1, \ldots, t_n\}$; f is called the *leading (function) symbol* of t.

The set of all terms is called T.

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

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 $\Box$.

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 $C = \{L_1, \ldots, L_n\}$ we define $C^d = \{L_1^d, \ldots, L_n^d\}$.

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. $|C_+| \leq 1$.

2.2 Term Structure

The term depth of an expression or of 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 $t = f(t_1, \ldots, t_n)$, where F is an n–place function symbol, then $\tau(t) = 1 + \max\{\tau(t_i) | 1 \leq i \leq n\}$.

The term depth of a literal L is defined as $\tau(L) = \max\{\tau(t) | t \in args(L)\}$. The term depth of a clause C is defined as $\tau(C) = \max\{\tau(L) | L \in C\}$. For a set S of clauses we define $\tau(S) = \max\{\tau(C) | C \in S\}$.

Definition 2.10 If t is a term then $s(t)$ — the number of subterms of t — is defined inductively as follows:

(i) If t is variable or a constant, then $s(t) = 1$.

(ii) If $t = f(t_1, \ldots, t_n)$, then $s(t) = 1 + \sum_{i=1}^{n} s(t_i)$.

We define simultaneously the notions of a subexpression and the corresponding depth of occurrence of a subexpression:

Definition 2.11 The i^{th} *subexpression* — $SUB(i, E)$ — of an expression E and its respective *depth of occurrence* are defined inductively as follows:

(i) $SUB(0, E) = E$, $\tau_{SUB}(0, E) = 0$

(ii) if $SUB(k, E) = f(t_1, \ldots, t_n)$, where f is an n–place function symbol, then $SUB(1+k+\sum_{j=1}^{i-1} s(t_j), E) = t_i$ and $\tau_{SUB}(1+k+\sum_{j=1}^{i-1} s(t_j), E) = \tau_{SUB}(k, E)+1$.

(iii) if $SUB(k, E) = P(t_1, \ldots, t_n)$, where P is an or an n–place predicate symbol, possibly preceded by a negation sign, then $SUB(1 + k + \sum_{j=1}^{i-1} s(t_j), E) = t_i$ and $\tau_{SUB}(1 + k + \sum_{j=1}^{i-1} s(t_j), E) = \tau_{SUB}(k, E)$.

It can be verified easily that $SUB(i, E)$ and $\tau_{SUB}(i, E)$ are defined uniquely for all $0 \leq i < s(E)$.

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.

$$OCC(x, E) = |\{i | SUB(i, E) = x\}|.$$

$OCC(x, C)$ 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.

Example 2.1 Let $E = P(x, f(f(y)))$. Then we have

$$\begin{array}{lcllcl} SUB(0,E) & = & P(x, f(f(y))), & SUB(1,E) & = & x, \\ SUB(2,E) & = & f(f(y)), & SUB(3,E) & = & f(y), \\ SUB(4,E) & = & y, & V(E) & = & \{x, y\}, \\ OCC(x,E) & = & 1, & OCC(y,E) & = & 1. \end{array}$$

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.

$$\tau_{MIN}(t, E) = \min\{\tau_{SUB}(i, E) | \forall i \text{ s.t. } SUB(i, E) = t).$$

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.

Example 2.2 Let $P_1 = P(x, f(f(y)))$, $P_2 = Q(f(x))$ and $C = \{P_1, \neg P_2\}$. Then we have

$$\begin{array}{lcllcl} \tau(P_1) & = & 2, & \tau(P_2) & = & 1, \\ \tau(C) & = & 2, & & & \\ \tau_{SUB}(0, P_1) & = & 0, & \tau_{SUB}(0, P_2) & = & 0, \\ \tau_{SUB}(1, P_1) & = & 0, & \tau_{SUB}(3, P_1) & = & 1, \\ \tau_{MIN}(x, C) & = & 0, & \tau_{MAX}(x, C) & = & 1, \\ \tau_{MIN}(y, C) & = & 2, & \tau_{MAX}(y, C) & = & 2. \end{array}$$

Definition 2.15 The *maximal variable depth* of an expression E is defined as $\tau_v(E) = \max\{\tau_{MAX}(x, E)|x \in V(E)\}$. For clauses C we define $\tau_v(C) = \max\{\tau_v(L)|L \in C\}$; analogously for clause sets S $\tau_v(S) = \max\{\tau_v(C)|C \in S\}$.

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 $\{x|\sigma(x) \neq x\}$ *domain* of σ and denote it by $dom(\sigma)$, $\{\sigma(x)|x \in dom(\sigma)\}$ is called *range* of σ ($rg(\sigma)$). By ε 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 $x_i \leftarrow t_i$ with the intended meaning $\sigma(x_i) = t_i$.

Definition 2.17 We say that a substitution σ 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 σ 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 σ a substitution.

(i) If E is a variable, then $E\sigma$ is $\sigma(E)$ (cf. definition *2.16*).

(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 n–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 σ to the atom of L. If C is a set of expressions or a clause, then $C\sigma = \{E\sigma|E \in C\}$.

Definition 2.19 An expression E_1 is an *instance* of another expression E_2 iff there exists a substitution σ 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 σ.

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

Definition 2.20 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 σ s.t. $E_1\sigma = E_2$. For substitutions ρ and θ we define analogously: $\rho \leq_s \theta$ iff there exists a substitution σ s.t. $\rho\sigma = \theta$. Similarly, if C and D are clauses, $C \leq_s D$ iff there exists a substitution σ s.t. $C\sigma \subseteq D$. In this case we also say, in accordance with the usual resolution terminology, that C *subsumes* D.

Chapter 3

The logical basis of resolution

3.1 The transformation to clause form

An essential feature of the resolution principle, such as of most other methods in automated deduction, is the inference on structurally very simple formulas called clauses. Clauses are quantifier-free constituents of conjunctive normal forms. While conjunctive normal forms are not required in all computational calculi, the elimination of quantifiers plays an essential role in all inference systems for automated deduction. Resolution is a refutational method and thus is based on the idea of indirect proof (reductio ad absurdum). Instead of proving A directly, we take $\neg A$, transform it to a clausal normal form $\mathcal{C}$ and refute $\mathcal{C}$.

Consider for example a mathematical theorem of the form G: $(A_1 \wedge \ldots \wedge A_n) \to C$, where the A_i's are some axioms and C is the conclusion (we assume that the A_i and C are all closed first order formulas). The first step is to consider F: $A_1 \wedge \ldots \wedge A_n \wedge \neg C$ (which is equivalent to $\neg G$), the second to transform F to a quantifier-free conjunctive normal form F'. In many practical cases the reduction to conjunctive normal form is relatively easy, as the form of F is already "conjunctive".

The crucial step in transforming F consists in the elimination of existential quantifiers. After removal of the existential quantifiers also the $\forall$-quantifiers can be omitted, because (by the specific structure of the formula) their position is of no significance anymore.

We now describe the main transformation steps in detail, where the underlying object is a formula F to be refuted).

Step 1: Eliminate $\to$, push all $\neg's$ in front of atomic formulas and remove multiple $\neg's$. More formally:

1a) Replace $(A \to B)$ by $(\neg A \vee B)$ everywhere in F.

1b) Replace $\neg(A \wedge B)$ by $(\neg A \vee \neg B)$
$\neg(A \vee B)$ by $(\neg A \wedge \neg B)$
$\neg\neg A$ by A
$\neg(\forall x)A$ by $(\exists x)\neg A$
$\neg(\exists x)A$ by $(\forall x)\neg A$

Step 2: Elimination of $\exists$-quantifiers.

After step-1 we have a formula F_1, containing only the logical symbols $\forall$, $\exists$, $\wedge, \vee, \neg$, where '$\neg$' only occurs in front of atomic formulas. Henceforth we call formulas of the form F_1 "normalized under step-1".

Let F be normalized under step 1 and $(\exists x)$ be an existential quantifier in F, which is in the scope of the universal quantifiers $(\forall y_1), \ldots, (\forall y_n)$. By adequate renaming of the quantifiers and by application of simple quantifier-shifting rules we get $F \sim (\forall y_1) \ldots (\forall y_n)(\exists x)F'$ (where F' is F without the quantifiers $(\forall y_1), \ldots, (\forall y_n), (\exists x)$).

Definition 3.1 Let (in some tree ordering) $(\exists x)$ be the first $\exists$-quantifier in F s.t.

$$F \sim (\forall y_1) \ldots (\forall y_n)(\exists x)F'.$$

We define

$$\delta(F) \equiv (F_{-(\exists x)})[^{f(y_1,\ldots,y_n)}_{x}]$$

where $(F_{-(\exists x)})$ is the formula F after omission of the quantifier $(\exists x)$ and f is a function symbol not occurring in F. If $(\exists x)$ is not in the scope of any $\forall$-quantifier, a new constant symbol is substituted for x. Note that $\delta(F)$ is logically equivalent to

$$(\forall y_1) \cdots (\forall y_n)F'[^{f(y_1,\ldots,y_n)}_{x}]$$

(but not syntactically identical).

The elimination of $\exists$-quantifiers effected by δ is frequently called skolemization (after the Norse logician Thoralf Skolem). The replacement of F by $\delta(F)$ is not without problems from a semantical point of view:

Example 3.1

$$F \equiv (\forall x)(\exists y)P(x, y), \delta(F) \equiv (\forall x)P(x, f(x)).$$

Obviously F is not logically equivalent to $\delta(F)$ (only $\delta(F) \rightarrow F$ holds) and thus the models of F and $\delta(F)$ are not the same.

But recall that our goal is to refute F; if instead of F we may refute $\delta(F)$ then no problems arise. In fact, F and $\delta(F)$ are sat(isfiability)-equivalent.

Definition 3.2 Two formulas F and G are sat-equivalent ($F \sim_{sat} G$) if it holds: F is satisfiable iff G is satisfiable.

The following theorem states the soundness of transformation δ under sat-equivalence.

Theorem 3.1 *Let F be a formula which is normalized under step–1 then $F \sim_{sat} \delta(F)$.*

Remark: A detailed proof can be found in the books [CL73], [Lov78].

Proof: (sketch)

(1) $\delta(F)$ satisfiable $\Rightarrow$ F satisfiable: This is trivial, as $\delta(F) \rightarrow F$ is valid.

(2) F satisfiable $\Rightarrow$ $\delta(F)$ satisfiable: Let $F \sim (\forall y_1) \ldots (\forall y_n)(\exists x)F'$. If $\mathcal{M}$ is a model of F then (intuitively) for all $y_1, \ldots, y_n$ there must be an x s.t. F' holds in $\mathcal{M}$. Thus (by the axiom of choice) we may select exactly one element x for every tuple $(y_1, \ldots, y_n)$ and get a n-ary function φ over the domain of $\mathcal{M}$. Let $\mathcal{M}'$ be an interpretation obtained from $\mathcal{M}$ by interpreting a new n-ary function symbol f by φ. Then $\mathcal{M}'$ is a model of

$$G \equiv (\forall y_1) \ldots (\forall y_n)F'[^{f(y_1,\ldots,y_n)}_{x}].$$

But G $\sim \delta(F)$.

Note that the argumentation above does not make use of the mathematical concept of a model–the term "for all $y_1, \ldots, y_n$" does not separate syntax and semantics cleanly–and only traces the intuitive lines of the proof.

In order to skolemize F it is not necessary to transform F into prenex form; such a transformation is even of negative influence as the following example shows.

Example 3.2

$$A \equiv (\forall x)(\exists y)P(x, y) \wedge (\forall u)(\exists v)Q(u, v).$$

The quantifier prefix of a prenex form for A is either $\forall\exists\forall\exists$ or $\forall\forall\exists\exists$. In both cases there is a $\exists-$quantifier which is in the scope of two $\forall-$quantifiers.
For

$$\begin{array}{lcl} B & \equiv & (\forall x)(\exists y)(\forall u)(\exists v)(P(x, y) \wedge Q(u, v)) \text{ we get} \\ \delta(B) & \equiv & (\forall x)(\forall u)(\exists v)(P(x, g(x)) \wedge Q(u, v)) \text{ and} \\ \delta(\delta(B)) & \equiv & (\forall x)(\forall u)(P(x, g(x)) \wedge Q(u, f(x, u))) \end{array}$$

On the other hand,

$$\delta(\delta(A)) \equiv (\forall x)P(x, g(x)) \wedge (\forall u)Q(u, h(u)).$$

That means skolemizing A directly requires introduction of one-place function symbols only, while two-place function symbols are required for the form B. Particularly note that $A \sim B$ does not imply $\delta(A) \sim \delta(B)$.

If F contains n existential quantifiers we can apply δ $n-$times and (as $\sim_{sat}$ is an equivalence relation) $F \sim_{sat} \delta^{(n)}(F)$ and $\delta^{(n)}(F)$ is $\exists-$free. Instead of $\delta^{(n)}(F)$ we write SK(F) and call SK(F) the skolemized form of F.

With the computation of SK(F) for a F normalized under step-1, step-2 (the elimination of $\exists$-quantifiers) is completed.

Example 3.3 We show all steps defined so far, starting from a theorem to be proved and ending with the skolemized form of its negation.

$$Let\ G \equiv [(\forall x)(\exists y)P(x,y) \wedge (\exists u)(\forall v)(P(u,v) \rightarrow Q(v))] \rightarrow (\exists z)Q(z).$$

$\neg G$ transforms under step-1 to:

$$\begin{aligned} F &\equiv (\forall x)(\exists y)P(x,y) \wedge (\exists u)(\forall v)(\neg P(u,v) \vee Q(u)) \wedge (\forall z)\neg Q(z). \\ \delta(F) &\equiv (\forall x)(P(x,f(x)) \wedge (\exists u)(\forall v)(\neg P(u,v) \vee Q(v)) \wedge (\forall z)\neg Q(z). \\ \mathrm{SK}(F) &\equiv \delta^{(2)}(F) \equiv (\forall x)P(x,f(x)) \wedge (\forall v)(\neg P(a,v) \vee Q(v)) \wedge (\forall z)\neg Q(z). \end{aligned}$$

In SK(F) the $\forall$-quantifiers are clearly redundant, because they can be shifted in front without changing the semantics.

$$\mathrm{SK}(F) \sim (\forall x)(\forall v)(\forall z)(P(x,f(x)) \wedge (\neg P(a,v) \vee Q(v)) \wedge \neg Q(z)).$$

The redundancy of quantifiers is not merely a feature of example 3.3, but holds for all Skolemized forms. We thus get:

Step-3: Eliminate all $\forall$–quantifiers from SK(F). The form obtained after step–3 is frequently called negation–normal form (NNF).

The NNF of F in example 3.3 is:

$$P(x,f(x)) \wedge (\neg P(a,v) \vee Q(v)) \wedge \neg Q(z).$$

In this case the NNF is already identical to the conjunctive normal form (CNF) of F. Generally one more step is required.

Step–4: Transform the NNF of F to a CNF.

Besides the usual way to transform into a logical equivalent CNF there is the possibility to transform into a sat–equivalent CNF (by introducing new predicate symbols) in polynomial time. But we don't go into details here [Ede92].

It is obvious that in a CNF also the Connectives $\wedge$, $\vee$ are redundant as they only hold specific fixed places. Moreover (as it is generally defined) a CNF has to be considered as normalized under associativity, commutativity and idempotency of $\wedge$ and $\vee$. So we get a natural representation of a CNF, as a set of clauses, a clause being a set of literals (see definition 2.5). Thus this last transformation step gives us the set of clauses

$$\{\{P(x,f(x))\},\{\neg P(a,v),Q(v)\},\{\neg Q(z)\}\}$$

for the formula F in example 3.3.

The clause form is quasi logic–free, as we need not to take into account properties of quantifiers and connectives anymore. Instead the computational inference methods concentrate on the term structure (which can even become more complicated by transformation to clause form).

3.2 Herbrand's Theorem for Clause Forms

In [Her30] Herbrand showed that the provability of a predicate logic formula F is equivalent to the provability of a purely existential form F' (obtained from F by eliminating $\forall$-quantifiers and introducing function symbols like in chapter 3.1). Moreover he proved that F' is derivable iff there is a disjunctive quantifier- and variable-free formula F" (defined by instances of the matrix of F') which is derivable. Here we have the concept of provability–equivalence (as opposed to logical equivalence) analogous to the concept of sat–equivalence defined before. It is easy to dualize Herbrand's result: A universal prenex formula is unsatisfiable iff there is a conjunctive quantifier- and variable-free ground formula (defined by ground instances of the matrix of F) F' which is unsatisfiable. Note that the second formulation is model theoretic, while the first is proof–theoretic; but by the completeness of (all standard) first order inference systems there is no actual difference. By adapting Herbrand's theorem to clause logic we get

Theorem 3.2 *Let $\mathcal{C}$ be a set of clauses. Then $\mathcal{C}$ is unsatisfiable iff there is a finite set of ground instances $\mathcal{C}$' of clauses in $\mathcal{C}$ which is unsatisfiable.*

The proof of theorem 3.2 is too complex to be presented here; again we refer to [CL73] and [Lov78]. But we will sketch the main ideas of the proof for Herbrand's theorem in its model theoretic version above. Note that a finite set of ground instances can be interpreted as a conjunctive normal form containing no variables.

First of all, satisfiability can be characterized by satisfiability via a specific kind of interpretation,the so called Herbrand interpretation. Before giving the formal definitions we present a motivating example.

Example 3.4 Let

$$\mathcal{C} = \{\ \{P(x,f(x))\},\ \{\neg P(a,v),Q(v)\},\ \{\neg Q(b)\}\ \}.$$

We define a domain for a model inductively:

$$\begin{aligned} H_0 &= \{a,b\}, \\ H_{n+1} &= H_n \cup \{f(t) \mid t \in H_n\}, \\ H_* &= \textstyle\bigcup_{i=0}^{\infty} H_i\ . \end{aligned}$$

H_* is the set of all ground terms over $\{a,b,f\}$ and is called the Herbrand universe of $\mathcal{C}$. Moreover we define an interpretation φ for the function symbol f as:

$$\varphi = \{(t,f(t)) \mid t \in H_*\}.$$

a,b are interpreted by "themselves" (note that the symbols a,b belong to the domain of interpretation). It remains to define truth values for P and Q, or better predicates π and ρ (π for P, ρ for Q):

$$\rho(t) = \text{TRUE} \ \text{ for } \ t \neq b,\ \rho(b) = \text{FALSE}.$$

$\pi(a,b)$ = FALSE, $\pi(s,t)$ = TRUE for $s \neq a$ or $t \neq b$. Because the domain of interpretation consists of terms we may write $P(a,b)$ = FALSE instead of $\pi(a,b)$ = FALSE. In fact the model can be described by assigning truth values to the ground atoms $P(s,t)$, $Q(s)$ for $s,t \in H_*$.

In general, a Herbrand interpretation is characterized by fixed domains (Herbrand universes) and fixed interpretations for constant – and function symbols.

Definition 3.3 Let $\mathcal{C}$ be a set of clauses. Then the Herbrand universe $H(\mathcal{C})$ is defined as:

$$H(\mathcal{C}) = \bigcup_{i=0}^{\infty} H_i(\mathcal{C}),$$

where the H_i are defined as:

$$H_0(\mathcal{C}) = \begin{cases} \mathrm{CS}(\mathcal{C}) & \text{if } \mathrm{CS}(\mathcal{C}) \neq \emptyset \\ \{a\} & \text{if } \mathrm{CS}(\mathcal{C}) = \emptyset,\ a \in \mathrm{CS} \text{ arbitrarily chosen,} \end{cases}$$

$$H_{n+1}(\mathcal{C}) = \{f(t_1,\ldots,t_{a(f)}) \mid f \in FS(\mathcal{C}), t_i \in H_n(\mathcal{C})\} \cup H_n(\mathcal{C}) \ .$$

Definition 3.4 Let $\mathcal{C}$ be a set of clauses. A Herbrand interpretation of $\mathcal{C}$ is an interpretation $(H(\mathcal{C}), \Phi, I)$ where for every $a \in \mathrm{CS}(\mathcal{C})$ $\phi(a) = a$ and for every $f \in FS_n(\mathcal{C})$, $n \in N$:

$$\phi(f) = \{(t_1,\ldots,t_n, f(t_1,\ldots,t_n)) \mid t_i \in H(\mathcal{C})\}.$$

As (semantically) a clause is the generalization of a disjunctive formula $\{L_1,\ldots,L_k\}$ corresponds to $(\forall x_1)\ldots(\forall x_n)(L_1 \vee \ldots \vee L_k)$. $\{L_1,\ldots,L_k\}$ is true in a Herbrand interpretation $\Delta = (H,\Phi,I)$ iff $\nu_{\Delta_J}(L_1 \vee \cdots \vee L_k)$ = TRUE for all interpretations $J \sim I \bmod x_1,\ldots,x_n$. But by the definition of a Herbrand interpretation

$$\nu_{\Delta_J}(L_1 \vee \cdots \vee L_k) = \text{TRUE for all } J \sim I \bmod x_1,\ldots,x_n$$
$$\text{iff}$$
$$\nu_{\Delta}(\sigma(L_1 \vee \cdots \vee L_K)) = \text{TRUE for all ground substitutions } \sigma \text{ over } H.$$

Thus C is true in Δ if for every $\sigma \in \mathrm{GS}(\mathcal{C})$ there is a literal L_i s.t. $\nu_\Delta(\sigma(L_i))$ = TRUE. On the other hand C is false in Δ if there is a ground instance σ over $\mathcal{C}$ s.t.

$$\nu_\Delta(\sigma(L_1 \vee \ldots \vee L_k)) = \text{FALSE}.$$

Herbrand interpretations are characterized by the truth values of the ground atoms and thus can be represented as:

$$\{P(s_1,\ldots,s_n) \mid (s_1,\ldots,s_n) \in A(P), P \in \mathrm{PS}_n(\mathcal{C}), n \in N\} \cup$$
$$\{\neg P(s_1,\ldots,s_n) \mid (s_1,\ldots,s_n) \in H_* - A(P), P \in \mathrm{PS}_n(\mathcal{C}),\ n \in N\}$$

for $A(P) = \{(s_1,\ldots,s_n) \mid (s_1,\ldots,s_n) \in H_*,\ \Phi(P)(s_1,\ldots,s_n) = \text{TRUE}\}$ ($P \in \mathrm{PS}_n(\mathcal{C})$). For $\mathcal{C} = \{\ \{P(x,f(x))\},\ \{\neg P(a,v), Q(v)\},\ \{\neg Q(b)\}\}$ the model defined before can thus be written as:

$$\{\neg Q(b), \neg P(a,b)\} \cup \bigcup_{s \in H_* - \{b\}} \{Q(s)\} \cup \bigcup_{(s,t) \in H_*^2 - \{(a,b)\}} \{P(s,t)\}.$$

The importance of Herbrand interpretations is grounded on the following result:

Theorem 3.3 *$\mathcal{C}$ is satisfiable iff there is a Herbrand model of $\mathcal{C}$.*

Proof: [CL73], [Lov78].

By theorem 3.3 we may reduce the question of unsatisfiability to that of unsatisfiability via Herbrand interpretations. Coming back to the proof of Herbrand's theorem, the last step is to show that, in case no Herbrand interpretation satisfies $\mathcal{C}$, this fact must be definable in finite terms. By systematically excluding H–interpretations from being models we end with finitely many alternatives which yield the set of ground instances $\mathcal{C}'$ for $\mathcal{C}$ (the key concept is that of a semantic tree). Again let

$$\mathcal{C} = \{\{P(x,f(x))\},\ \{\neg P(a,v), Q(v)\},\ \{\neg Q(z)\}\}$$

like in example 3.3. For the finite unsatisfiable set of ground clauses we may take

$$\{\{P(a,f(a))\},\ \{\neg P(a,f(a)), Q(b)\},\ \{\neg Q(b)\}\}.$$

While Gilmore [Gil60] used Herbrand's theorem directly (by successively generating sets of ground instances and testing them for unsatisfiability) modern inference methods use Herbrand's theorem for theoretical analysis only (not for algorithms).

3.3 The Unification Principle

Unification is essential to deductions in every logical calculus containing a substitution rule (or substitution axioms). Suppose for example that we have derived the formulas $A(a)$ and $(\forall x)(A(x) \rightarrow B(x))$ in PL. To infer $B(a)$ we apply the substitution rule to the second formula and get $A(a) \rightarrow B(a)$; afterwards we apply modus ponens and derive $B(a)$. Thus we have "unified" $A(a)$ and $A(x)$ to $A(a)$ in order to apply modus ponens. In the resolution calculus for clause logic unification of literals plays a central role.

Again, consider the example

$$\mathcal{C} = \{\{P(x,f(x))\},\ \{\neg P(a,v), Q(v)\},\ \{\neg Q(z)\}\}.$$

$\mathcal{C}$ corresponds to the PL–formula

$$(\forall x)P(x,f(x)) \wedge (\forall v)(\neg P(a,v) \vee Q(v)) \wedge (\forall z)\neg Q(z).$$

We derive a contradiction by using the substitution rule and the propositional cut rule

$$\frac{A \vee L \vee B \quad C \vee L^d \vee D}{A \vee B \vee C \vee D}.$$

We start with substitutions first:

$$S_1: \quad \frac{(\forall x)P(x,f(x))}{P(a,f(a))}$$

$$S_2: \quad \frac{(\forall v)(\neg P(a,v) \vee Q(v))}{\neg P(a,f(a)) \vee Q(f(a))}$$

The substitution used in S_1 is $\sigma_1 = \{x \leftarrow a\}$, the one used in S_2 is $\sigma_2 = \{v \leftarrow f(a)\}$. Defining

$$\sigma = \sigma_1 \cup \sigma_2 = \{x \leftarrow a,\ v \leftarrow f(a)\}$$

we get

$$\sigma(P(x,f(x))) = \sigma(P(a,v)) = P(a,f(a)).$$

σ is called a unifier of $P(x,f(x)), P(a,v)$. After application of S_1, S_2 we derive:

$$\frac{P(a,f(a)) \quad \neg P(a,f(a)) \vee Q(f(a))}{Q(f(a))}$$

By S_3:

$$\frac{(\forall z)\neg Q(z)}{\neg Q(f(a))}$$

we get the unifier $\eta = \{z \leftarrow f(a)\}$ of $Q(f(a))$ and $Q(z)$. From $Q(f(a))$ and $\neg Q(f(a))$ we derive a contradiction. This example shows the basic idea of resolution, where unification is performed to make a propositional cut rule applicable.

The general problem is to find a substitution σ for a set of expressions

$$\{E_1, \ldots, E_n\}$$

s.t. $\sigma(E_1) = \cdots = \sigma(E_n)$, or $\mid \{E_1, \ldots, E_n\}\sigma \mid = 1$.

Definition 3.5 Let W be a nonempty set of expressions. A substitution σ is called unifier of W if $\mid W\sigma \mid = 1$.

In general there are infinitely many unifiers even if the domain of the substitution is restricted to $V(W)$.

Example 3.5

$$W = \{P(x,f(x)),\ P(f(y),z)\}.$$

The following substitutions with domain $\subseteq \{x,y,z\}$ are unifiers:

$$\sigma = \{x \leftarrow f(y), z \leftarrow f(f(y))\}$$

and for every $t \in T - \{y\}$:

$$\sigma_t = \{x \leftarrow f(t), z \leftarrow f(f(t)), y \leftarrow t\}.$$

We see that there is something superfluous in the definition of the unifiers σ_t and that σ is more "economical". Indeed σ is a so called most general unifier (m.g.u.)

as every other unifier of W can be derived from σ by an additional substitution. In fact it holds:

$$\sigma_t = \sigma\{y \leftarrow t\} \text{ for all } t \in T - \{y\} \text{ and thus}$$
$$\sigma \leq_s \sigma_t \text{ for all such } t.$$

In computational logic only m.g.u.'s are computed (this really suffices). Note that modulo variable permutations there is only one m.g.u. for a set of expressions W, while the set of all unifiers can be infinite; thus the restriction to m.g.u.'s results in an obvious gain in efficiency.

Definition 3.6 A unifier σ of W is called most general unifier (m.g.u.) if for every unifier η of W it holds $\sigma \leq_s \eta$.

In the sense of the subsumption ordering $\leq_s$, m.g.u.'s are minimal elements in the set of all unifiers.

The properties of substitutions and unifications are investigated exhaustively in [Ede85]. One of the results is that the order of computing unifiers is irrelevant:

$$mgu(W_1 \cup W_2) = mgu(W_1\sigma_1 \cup W_2\sigma_2)$$

where

$$\sigma_i = mgu(W_i).$$

Thus

$$mgu(\{E_1, \ldots, E_n\}) = mgu(\{E_1, F\})$$

where

$$F = E_2\sigma = \cdots = E_n\sigma$$

and

$$\sigma = mgu(\{E_2, \ldots, E_n\}).$$

By the property above it clearly suffices to compute m.g.u.'s for two–element sets W and to iterate the computation. As for two m.g.u.'s σ, γ we have $\sigma =_s \gamma$ but not (in general) $\sigma = \gamma$, mgu () is not a function in the strict sense. But we will never get problems by speaking of "the" most general unifier.

To find m.g.u.'s does not require imagination or creativity, but can be achieved by algorithms (so called unification algorithms). There are linear time algorithms to decide unifiability and to represent the unifier in compositional form [MM82] (to apply the unifier to the set of expressions may be exponential). As we do not focus on complexity issues here, we present a simple (exponential worst-case) algorithm which decides unifiability of two expressions and, in case of unifiability, computes a m.g.u.

Let

$$E = \{P(t_1, \ldots, t_n),\ P(s_1, \ldots, s_n)\}.$$

By elementary properties of substitutions we have

$$\begin{aligned} P(t_1,\ldots,t_n)\sigma &= P(t_1\sigma,\ldots,t_n\sigma),\\ P(s_1,\ldots,s_n)\sigma &= P(s_1\sigma,\ldots,s_n\sigma). \end{aligned}$$

Thus E is unifiable iff there is a substitution σ s.t.

$$s_1\sigma = t_1\sigma, s_2\sigma = t_2\sigma, \ldots, s_n\sigma = t_n\sigma.$$

We may also say that all pairs $(s_1,t_1),\ldots,(s_n,t_n)$ have to be unified simultaneously. The (s_i,t_i) may be again of the form $(f(u_1,\ldots,u_m), f(v_1,\ldots,v_m))$ and so the property holds recursively. This leads to the following definition:

Definition 3.7 Let E_1, E_2 be two expressions. The set of corresponding pairs $CORR(E_1,E_2)$ is defined as follows:

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

(2) If $(F_1,F_2) \in CORR(E_1,E_2)$ s.t. $F_1 = F(s_1,\ldots,s_n)$ and $F_2 = F(t_1,\ldots,t_n)$, where $F \in \mathrm{FS} \cup \mathrm{PS}$ or $F \equiv \neg P$ for $P \in \mathrm{PS}$, then $(s_i,t_i) \in CORR(E_1,E_2)$ for $i = 1,\ldots n$.

(3) Nothing else is in $\mathrm{CORR}(E_1,E_2)$.

A pair (F_1,F_2) is called irreducible if the leading symbols of F_1,F_2 are different and strongly irreducible if it is irreducible and both F_1,F_2 are not variables.

Example 3.6

$$\begin{aligned} E_1 &= P(f(x,y),f(x,a)),\\ E_2 &= P(z,f(x,b))\\ \mathrm{CORR}(E_1,E_2) &= \{(E_1,E_2),\ (f(x,y),z),\ (f(x,a),f(x,b)),\ (x,x),\ (a,b)\} \end{aligned}$$

$(f(x,y),z)$ is irreducible and (a,b) is strongly irreducible.

It is obvious that $\{E_1,E_2\}$ is not unifiable if $\mathrm{CORR}(E_1,E_2)$ contains a strongly irreducible pair. (Note that strong irreducibility is stable under substitutions).

The key technique of the following unification algorithm is to eliminate pairs which are irreducible, but not strongly irreducible.

Example 3.7

$$\begin{aligned} E_1 &= P(x,f(x,y)),\\ E_2 &= P(f(u,u),v),\\ \mathrm{CORR}(E_1,E_2) &= \{(E_1,E_2),(x,f(u,u)),(f(x,y),v)\} \end{aligned}$$

We eliminate the irreducible pair $(x,f(u,u))$ by applying the substitution

$$\sigma_1 = \{x \leftarrow f(u,u)\}.$$

We get

$(E_1\sigma_1, E_2\sigma_1)$ and
$\mathrm{CORR}(E_1\sigma_1,E_2\sigma_1) = \{(E_1\sigma_1,E_2\sigma_1),(f(u,u),f(u,u)),(f(f(u,u),y),v),(u,u)\}$.

There is only one irreducible pair left which we eliminate by

$$\sigma_2 = \{v \leftarrow f(f(u,u),y)\}.$$

It is easy to see that

$$E_1\sigma_1\sigma_2 = E_2\sigma_1\sigma_2$$

and that $\mathrm{CORR}(E_1\sigma_1\sigma_2, E_2\sigma_1\sigma_2)$ consists only of expressions of the form (s,s).

Unification can fail because of two reasons:

1) A strongly irreducible pair occurs.

2) There exists an irreducible pair of the form (x,t) or (t,x) s.t. $x \in V(t)$.

Note that if $x \in V(t)$ (x occurs properly in t) then $\lambda(x)$ occurs properly in $\lambda(t)$ for every substitution λ; thus, in this case, x and t are not unifiable.

We have now the means to define UAL, a nondeterministic unification algorithm for two expressions (Fig. 3.1). The while-loop of UAL is always terminating because the number of variables in $E_1\vartheta \cup E_2\vartheta$ is less than in $E_1 \cup E_2$.

Theorem 3.4 UAL *is a decision algorithm for unification. If* $\{E_1,E_2\}$ *is unifiable then* UAL *yields a m.g.u.* ϑ.

Proof: Let η be an arbitrary unifier of $\{E_1,E_2\}$.
One shows by induction on k:
For every ϑ_k (ϑ_k being ϑ after k executions of the while–loop) there is a μ_k s.t. $\vartheta_k\mu_k = \eta$. For $k = 0$ $\vartheta_k = \varepsilon$ and μ_k can be set to η.
Because UAL must terminate with a k_0 we have $\vartheta_{k_0}\mu_{k_0} = \eta$, where μ_{k_0} depends on η, but not k_0 and ϑ_{k_0}; thus ϑ_{k_0} is a m.g.u..
If $\{E_1,E_2\}$ is not unifiable then UAL must clearly terminate with failure because $E_1\vartheta \neq E_2\vartheta$ holds for all ϑ and the while loop terminates.

UAL, or rather any deterministic version of it, is not very efficient because the $E_i\vartheta$ are actually computed during the algorithm. On the other hand UAL clearly reflects the properties of unification and is well suited for theoretical purposes.

```
algorithm UAL;
begin
   ϑ := ε;
   while E1ϑ ≠ E2ϑ do
      if there is a strongly irreducible pair in CORR(E1ϑ, E2ϑ) then
         failure
      else
         Select an irreducible pair (s, t) ∈ CORR(E1ϑ, E2ϑ);
         if s ∈ V then
            α := s; β := t;
         else
            α := t; β := s;
         end if;
         if α occurs in β then
            failure
         else
            ϑ := ϑ{α ← β};
         end if
      end if
   end while
   (* ϑ is a m.g.u. *)
end.
```

Figure 3.1: Unification Algorithm for E_1 and E_2

3.4 The Resolution Principle

The characteristic of the resolution principle is the combination of the unification method and of a propositional cut rule. Let

$$\mathcal{C}_1 = \{L_1, \ldots, L_n\},$$

$$\mathcal{C}_2 = \{M_1, \ldots, M_m\}$$

be two clauses s.t. $L_i^d = M_j$ for some i, j, $1 \leq i \leq n, 1 \leq j \leq m$. The formulas F_1, F_2 corresponding to C_1, C_2 are:

$$\begin{array}{lcl} F_1 & \equiv & (\forall x_1) \cdots (\forall x_r)(L_1 \vee \cdots \vee L_i \vee \cdots \vee L_n) \text{ and} \\ F_2 & \equiv & (\forall y_1) \cdots (\forall y_s)(M_1 \vee \cdots \vee M_j \vee \cdots \vee M_m). \end{array}$$

By substitution and simple propositional rules we derive

$$L_i^d \rightarrow L_1 \vee \cdots \vee L_{i-1} \vee L_{i+1} \vee \cdots \vee L_n$$

from F_1 and

$$M_j^d \rightarrow M_1 \vee \cdots \vee M_{j-1} \vee M_{j+1} \vee \cdots \vee M_m$$

from F_2.
But by $M_j^d = L_i$ and the valid propositional rule

$$\frac{L^d \rightarrow A,\ L \rightarrow B}{A \vee B}$$

we get

$$L_1 \vee \cdots \vee L_{i-1} \vee L_{i+1} \vee \cdots \vee L_n \vee M_1 \vee \cdots \vee M_{j-1} \vee M_{j+1} \cdots \vee M_m.$$

But the last formula corresponds to the clause $(C_1 - \{L_i\}) \cup (C_2 - \{M_j\})$. In case of ground clauses, the derivation of the last clause already characterizes the resolution principle, but for general clauses a combination with the unification principle is required.

So far we are at the point to define the propositional resolution principle (which on ground clauses coincides with the general one to be defined later).

Definition 3.8 Let C_1, C_2 be two clauses and $L \in C_1, M \in C_2$ s.t. $L^d = M$. Then $(C_1 - \{L\} \cup (C_2 - \{M\})$ is called propositional resolvent of C_1, C_2.

It is obvious that propositional resolution does not suffice for a complete refutational calculus in clause logic, as clauses are universal forms representing infinitely many quantifier–free disjunctions.

Example 3.8 Let $\mathcal{C}$ be the set $\{C_1, C_2, C_3\}$ of clauses with

$$\begin{aligned} C_1 &= \{P(x, f(x))\}, \\ C_2 &= \{\neg P(a, v), Q(v)\}, \\ C_3 &= \{\neg Q(z)\}. \end{aligned}$$

We see that propositional resolution is not applicable to $\mathcal{C}$ as there are no complementary literals. But by unifying $\{Q(v), Q(z)\}$ with m.g.u. $\sigma = \{z \leftarrow v\}$ we obtain

$$C_2\sigma = \{\neg P(a, v), Q(v)\}$$

and

$$C_3\sigma = \{\neg Q(v)\}.$$

Now propositional resolution yields

$$C_4 = \{\neg P(a, v)\}.$$

In the next step we unify $\{P(x, f(x)), P(a, v)\}$ with $\sigma' = \{x \leftarrow a, v \leftarrow f(a)\}$ and get

$$C_4\sigma' = \{\neg P(a, f(a))\},$$

$$C_1\sigma' = \{P(a, f(a))\}.$$

Propositional resolution of $C_4\sigma'$ and $C_1\sigma'$ yields $\{\}$ (which we henceforth denote by $\square$) and we obtain a contradiction.

We are not through yet, as unifying only two complementary literals and applying propositional resolution is not sufficient to refute all unsatisfiable set of clauses. Moreover we will see that, without renaming variables in the two clauses used for inference, inference may be either impossible or yield an inadequate result.

Example 3.9

$$\mathcal{C} = \{\{P(x)\}, \{\neg P(f(x)), Q(y)\}, \{\neg Q(z), R(y), S(z)\}\}$$

In trying to resolve C_1, C_2 we test $W = \{P(x), P(f(x))\}$ for unification; but W is not unifiable as x properly occurs in $f(x)$. This is an unwanted effect, as names of variables in clauses are meaningless (because they must be considered as generalized). This problem can be solved by renaming C_1 to $C_1' = \{P(z)\}$. Another side effect takes place if we resolve C_2 and C_3: By cutting out the Q–literals via m.g.u. $\sigma = \{z \leftarrow y\}$ we get $C_4 = \{\neg P(f(x)), R(y), S(y)\}$. On the other hand we could write C_3 as $\{\neg Q(z), R(u), S(z)\}$ without changing its semantics. But then we obtain $C_4' = \{\neg P(f(x)), R(u), S(y)\}$; note that C_4' is (strictly) more general than C_4. Thus we see that renaming of variables is important to unification and to the semantical status of the resulting clause.

The next example shows that unification within (!) clauses is also necessary for completeness.

Example 3.10

$$\mathcal{C} = \{\{P(x), P(y)\}, \{\neg P(u), \neg P(v)\}\}$$

Obviously $\mathcal{C}$ is unsatisfiable. Just take

$$C_1' = \{P(a)\}, \; C_2' = \{\neg P(a)\}$$

as ground instances of C_1, C_2; C_1' and C_2' resolve to $\square$. Now suppose we may unify complementary literals and apply the propositional resolution rule. Furthermore we enforce renaming of clauses before resolution . Under these conditions we derive $C_3 = \{P(x), \neg P(v)\}$ from C_1, C_2 and some variants of C_3. But by renaming, selecting potentially complementary literals, unifying and applying propositional resolution with C_3 we only get variants of C_1 and C_2 and the derivation cycles.This problem can be solved by unification within C_1. Take $W = \{P(x), P(y)\}$ and compute the m.g.u. $\sigma = \{x \leftarrow y\}$; then the clause $\{P(y)\}$ results by which (in two steps) a contradiction can be obtained.

Unification within a clause is characterized by the following concept:

Definition 3.9 Let C be a clause, $A \subseteq C$ and $A \neq \emptyset$. If A is unifiable by m.g.u. σ, $C\sigma$ is called factor of C. If $\mid A \mid = 1$ then $C\sigma (= C)$ is called a trivial, if $\mid A \mid > 1$ a nontrivial factor of C (in the latter case we have $\mid C\sigma \mid < \mid C \mid$).

Resolution with factoring and renaming is then defined in

Definition 3.10 Let C_1, C_2 be clauses with $V(C_1) \cap V(C_2) = \emptyset$.
Let C_1' be a factor of C_1 and C_2' be a factor of C_2.
Suppose that there are literals $L \in C_1$' and $M \in C_2$' s.t. $\{L^d, M\}$ is unifiable by m.g.u. σ. Then $(C_1' - \{L\})\sigma \cup (C_2' - \{M\})\sigma$ is called resolvent of C_1 and C_2.

It should be noted at this point that there are different definitions of resolution. In some of them a resolvent is defined as

$$(C_1'\sigma - \{L\}\sigma) \cup (C_2'\sigma - \{M\}\sigma)$$

instead of

$$(C_1' - \{L\})\sigma \cup (C_2' - \{M\})\sigma$$

as in definition 3.10.
Indeed

$$C\sigma - \{L\}\sigma \subseteq (C - \{L\})\sigma,$$

but the converse inclusion does not hold in general as the example
$C = \{P(x), P(a)\}, L = \{P(x)\}$ and $\sigma = \{x \leftarrow a\}$ shows.
The form

$$(C_1'\sigma - \{L\}\sigma) \cup (C_2'\sigma - \{M\}\sigma)$$

is more natural as it is propositional resolution after application of unification, but the form

$$(C_1' - \{L\})\sigma \cup (C_2' - \{M\}\sigma)$$

is better for theoretical purposes.
At least we have always

$$(C_1'\sigma - \{L\}\sigma) \cup (C_2'\sigma - \{M\}\sigma) \subseteq (C_1' - \{L\})\sigma \cup (C_2' - \{M\})\sigma.$$

The concept on the left-hand side can be found in the books of Chang& Lee [CL73] and of Loveland [Lov78], while the right one (although defined in a different way) is that of Robinson [Rob65]. The concepts in [CL73], [Lov78] can cause some unwanted side effects in lifting which are analyzed in [Lei89].

The resolution calculus is based on resolution as single rule operating on sets of clauses. Deductions are defined in the usual way.

Definition 3.11 A resolution deduction (R-deduction) of a clause C from a set of clauses $\mathcal{C}$ is a sequence $C_1, \ldots, C_n$ with the following properties:

1) $C_n = C$

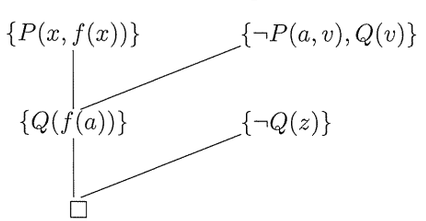
2) For every i s.t. $1 \leq i \leq n$ we either have that C_i is a variant of a clause in $\mathcal{C}$ or C_i is a resolvent of some variants of C_j, C_k for $j, k < i$.

An R-deduction of $\square$ from $\mathcal{C}$ is called R-refutation of $\mathcal{C}$.

Example 3.11 (compare example 3.8)

$$\begin{aligned} \mathcal{C} &= \{\{P(x, f(x))\}, \{\neg P(a, v), Q(v)\}, \{\neg Q(z)\}\}. \\ \Gamma &= \{P(x, f(x))\}, \{\neg P(a, v), Q(v)\}, \{Q(f(a))\}, \{\neg Q(z)\}, \square; \end{aligned}$$

Γ is a R-refutation of $\mathcal{C}$ which can be represented more conveniently in the tree:



Note that there may be different tree representations for one sequence. Deductions could also be defined as trees (a priori), but we don't need the tree concept of deduction in this course.

Recall that $\mathcal{C}$, as used above, is the clausal form of the negation of the theorem

$$G \equiv [(\forall x)(\exists y)P(x, y) \wedge (\exists u)(\forall v)(P(u, v) \rightarrow Q(v))] \rightarrow (\exists z)Q(z).$$

Thus Γ can be considered as a proof (by contradiction) of G.
R-deductions are always correct, as $F(\{C_1, C_2\}) \rightarrow F(\{C_3\})$ is valid if C_3 is resolvent of C_1, C_2 (the "Robinson"–resolvent is always implied by a Chang&Lee–resolvent, where the latter is defined by substitution and propositional resolution). But we also need (refutational) completeness, that is the existence of an R-refutation if $\mathcal{C}$ is unsatisfiable.

Theorem 3.5 *If $\mathcal{C}$ is an unsatisfiable set of clauses then there exists an R-refutation of $\mathcal{C}$.*

Proof:(sketch) One first shows that (propositional) resolution is complete on sets of ground clauses. Then let $\mathcal{C}$ be an unsatisfiable set of (general) clauses. By Herbrand's theorem there exists a finite set of ground instances $\mathcal{C}$' of clauses in $\mathcal{C}$ s.t. $\mathcal{C}$' is unsatisfiable. By the completeness of propositional resolution there exists an R-refutation Γ' of $\mathcal{C}$'.
By a technique (called lifting) it is possible to find a (general) R-refutation Γ of $\mathcal{C}$ s.t. there exists a ground substitution η with $\Gamma\eta = \Gamma'$.
The most essential (and hardest) step in the proof is the lifting of the ground refutation. The lifting property is remarkable, because in R-deductions only m.g.u.'s are allowed as substitutions; on the other hand arbitrary ground substitutions are allowed in the definition of $\mathcal{C}$'. Lifting is possible not only for ground refutations but for arbitrary resolution deductions.

Theorem 3.6 *(Lifting lemma)*
Let $\mathcal{C}$' be a set of instances of clauses in $\mathcal{C}$ and let Γ' be a R-deduction from Γ'. Then there exists a R-deduction Γ from $\mathcal{C}$ and a substitution η s.t. $\Gamma\eta = \Gamma'$.

Remark: Theorem 3.6 is formulated more generally as it is needed to prove theorem 3.5 (ground instances suffice for this purpose). But theorem 3.6 is important also in completeness proofs for subsumption type refinements, where the general formulation is required.

Of course theorem 3.6 has to be proved before theorem 3.5. The nature of theorem 3.6 is purely proof theoretical, while semantical concepts are important in theorem 3.5.

Chapter 4

Resolution Refinements

4.1 The Concept of Refinement

From the logical point of view we might be satisfied at this point, as we have shown correctness and completeness of the resolution principle. But the large number of possible resolvents derivable from a set of clauses is a serious obstacle to practical applications. Thus it is significant that Robinson presented a paper on hyper-resolution [Rob65a]—a refinement to be described later—in the same year as his foundational paper on resolution [Rob65] was published.

The principal idea of a refinement is to restrict the R-deduction concept without loosing completeness; under such restrictions there are less possible derivations and the node degree of search trees (in implementations) is decreased. So one motivation for restricting the R-deduction concept is efficiency. Another important application of refinements is the construction of resolution decision procedures for (decidable) clause classes; here the key idea consists in finding a refinement which is complete and produces only finitely many resolvents (on the class under consideration).

Let us consider the concept of refinement generally and from a formal point of view. First we define some mathematical notions describing the set of derivable resolvents.

Definition 4.1 Let $\mathcal{C}$ be a set of clauses. Then $R(\mathcal{C})$ is the set of all resolvents which can be defined by (variants of) clauses in $\mathcal{C}$, where $R(\mathcal{C})$ is factorized under renaming (there are no different clauses $C_1, C_2 \in R(\mathcal{C})$ which are variants of each other).

Note that, by factorizing under renaming, we ensure that $R(\mathcal{C})$ is always finite for finite $\mathcal{C}$.
Generally we write $\mathcal{C}/_{\sim_v}$ for a set of clauses $\mathcal{C}$ after factorization under renaming. $R(\mathcal{C})$ is the set of clauses derivable within one step; the set of all derivable clauses $R^*(\mathcal{C})$ can be specified as follows:

$$\begin{aligned} R^0(\mathcal{C}) &= \mathcal{C}/_{\sim_v} \\ R^{i+1}(\mathcal{C}) &= (R^i(\mathcal{C}) \cup R(R^i(\mathcal{C})))/_{\sim_v} \\ R^*(\mathcal{C}) &= \textstyle\bigcup_{i=0}^{\infty} R^i(\mathcal{C}) \end{aligned}$$

Attention: $R^1(\mathcal{C})$ and $R(\mathcal{C})$ are to be considered as different!
By the completeness of resolution we know that $\square \in R^*(\mathcal{C})$ if $\mathcal{C}$ is unsatisfiable. If

there is a refutation tree of depth n for $\mathcal{C}$ then $\square \in R^n(\mathcal{C})$.
As already mentioned, $R^n(\mathcal{C})$ can be very large even for small n. Thus the idea is to define an operator S s.t. $S^*(\mathcal{C}) \subset R^*(\mathcal{C})$ and $\square \in S^*(\mathcal{C})$ if $\mathcal{C}$ is unsatisfiable.
The restriction $S^i(\mathcal{C}) \subseteq R^i(\mathcal{C})$ for all $i \geq 0$ is too general to make sense: Think about the following method for the definition of the $S^i(\mathcal{C})$:

1. Search for an R-refutation Γ of $\mathcal{C}$. If there is no refutation then define $S^i(\mathcal{C}) = \mathcal{C}/_{\sim_v}$ for all $i \geq 0$.

2. If Γ is an R-refutation then define:

$$\begin{array}{lcl} S^0(\mathcal{C}) & = & \mathcal{C}/_{\sim_v} \\ S^{i+1}(\mathcal{C}) & = & (\{C \mid C \text{ in } \Gamma,\, C \in R(R^i(\mathcal{C}))\} \cup S^i(\mathcal{C}))/_{\sim_v}. \end{array}$$

By this definition, clearly $S^i(\mathcal{C}) \subseteq R^i(\mathcal{C})$ for all i and $S^*(\mathcal{C}) = \mathcal{C}/_{\sim_v}$ for $\mathcal{C}$ satisfiable. Although S defines a refinement in an abstract sense, it is unreasonable from a practical point of view since unrefined resolution has been applied to define the refinement. Moreover S^i clearly is not recursive, because

$$(\mathcal{C} \text{ is unsatisfiable and } \square \notin \mathcal{C})$$

iff

$$(S^*(\mathcal{C}) \neq \mathcal{C}/_{\sim_v} \text{ and } \square \notin \mathcal{C})$$

(thus a recursive computation of $S^*(\mathcal{C})$—which is always finite—would give a decision procedure for whole clause logic). Still very general but more reasonable is the following definition.

Definition 4.2 A resolution refinement operator S is a mapping from sets of clauses to sets of clauses s.t. it holds:

1. $S(\mathcal{C})$ is a finite subset of $R^*(\mathcal{C})$.

2. S is recursive.

Again we define

$$\begin{array}{lcl} S^0(\mathcal{C}) & = & \mathcal{C}/_{\sim_v}, \\ S^{i+1}(\mathcal{C}) & = & (S^i(\mathcal{C}) \cup S(S^i(\mathcal{C})))/_{\sim_v}, \\ S^*(\mathcal{C}) & = & \bigcup_{i=0}^{\infty} S^i(\mathcal{C}). \end{array}$$

It is immediately verified that $S^*(\mathcal{C}) \subseteq R^*(\mathcal{C})$ for all set of clauses $\mathcal{C}$. We did not postulate $S(\mathcal{C}) \subset R(\mathcal{C})$, because there are practically relevant "refinements" which do not fulfill this property. Definition 4.2 captures all relevant refinements which don't use deletion rules.

4.2 Restrictions on the Resolvents of two Clauses

The restrictions analyzed in this chapter are local in the sense, that only the set of resolvents defined by two clauses is made smaller, but there is no specific condition on the form of deduction trees. Although there is a great variety of such local refinements, we focus on two typical basic types:

1. A-ordering refinements, and

2. Locking refinements.

In A-ordering refinements the central feature is the ordering of atoms, which ensures that only the largest atoms are resolved.

Definition 4.3 An A-ordering $<_A$ is a binary relation on atoms with the following properties:

(A1) $<_A$ is irreflexive.

(A2) $<_A$ is transitive.

(A3) For all atoms A, B and for all substitutions ϑ, $A <_A B$ implies $A\vartheta <_A B\vartheta$.

The property (A3) is important to ground lifting, as will be explained later.

Example 4.1 Let A, B be arbitrary atoms. We define $A <_d B$ iff

1. $\tau(A) < \tau(B)$ and

2. For all $x \in V(A)$: $\tau_{\max}(x, A) < \tau_{\max}(x, B)$ (and $V(A) \subseteq V(B)$).

Irreflexivity and transitivity of $<_d$ easily follow from 1. If $\tau_{\max}(x, A) < \tau_{\max}(x, B)$ for all $x \in V(A)$ and $\tau(A) < \tau(B)$ then for all $\vartheta \in SUBST$:

$$\tau_{\max}(y, A\vartheta) < \tau_{\max}(y, B\vartheta) \text{ for } y \in V(A\vartheta) \text{ and } \tau(A\vartheta) < \tau(B\vartheta).$$

Thus $A <_d B$ implies $A\vartheta <_d B\vartheta$ and property (A3) holds too.

For $<_d$ we have $P(x, x) <_d Q(f(x), y)$ and
$P(x, y) <_d R(g(x, y))$,
but not $P(x, f(a)) <_d Q(x, f(x))$ (1 is violated)
$P(x, a) <_d P(f(a), x)$. (2 is violated)

Note that by (A1), (A2) and (A3) it is guaranteed that two atoms A, B with $A <_A B$ are not unifiable (otherwise we get $A\vartheta <_A B\vartheta$ by (A3) and $A\vartheta = B\vartheta$ for a unifier ϑ; but then (A1) is violated).

The ordering $<_A$ is an ordering for atoms, not for literals. Thus in extending the ordering to literals, the sign of the literal is insignificant:

$$L <_A M \quad \text{iff} \quad \text{at}(L) <_A \text{at}(M) \text{ for literals } L, M.$$

According to our definition of refinement we have to specify an operator $R_{<_A}$ defining a set of resolvents obtained under the $<_A$ restriction. For this purpose we introduce the concept of resolved literal.

Let C_1, C_2 be two variable disjoint clauses, $C_1\eta_1, C_2\eta_2$ factors of C_1, C_2 and

$$(C_1\eta_1 - \{L_1\})\sigma \cup (C_2\eta_2 - \{L_2\})\sigma$$

be a resolvent of C_1, C_2 then $L_1\sigma, L_2\sigma$ are called the resolved literals, $\text{at}(L_1)\sigma$ is called the resolved atom. Note that the resolved literal is not the original one (occurring in C), but the literal subjected to the m.g.u. of the resolution.

Definition 4.4 Let $\mathcal{C}$ be a set of clauses and $<_A$ be an A-ordering. We define $C \in R_{<_A}(\mathcal{C})$ iff C is a resolvent of two clauses $C_1, C_2 \in \mathcal{C}$ and for no literal $L \in C$: $A <_A L$ where A is the resolved atom.

Example 4.2 Let $\mathcal{C}$ be the set $\{C_1, C_2\}$ of clauses with

$$\begin{aligned} C_1 &= \{\neg R(f(x)), Q(f(x), x)\}, \\ C_2 &= \{\neg Q(y, z), R(f(y))\}. \end{aligned}$$

Then $C_3 = \{Q(f(x), x), \neg Q(x, z)\}$ is a $<_d$-resolvent, i.e., $C_3 \in R_{<_d}(\mathcal{C})$, because $R(f(x))$ is the resolved atom and there is no literal L in C_3 s.t. $R(f(x)) <_d L$.
$C_4 = \{\neg R(f(x)), R(f(f(x)))\}$ is in $R(\mathcal{C})$ but not in $R_{<_d}(\mathcal{C})$, because $Q(f(x), x)$ is the resolved literal and $Q(f(x), x) <_d R(f(f(x)))$.

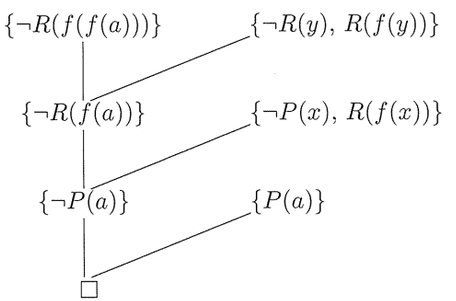
Note that A-ordering (as defined in [Joy76] and by us here) is an "a posteriori"-ordering. Whether a resolvent is allowed or not is determined by comparing the literals in the resolvent and the resolved literal. Thus it is not absurd to use (unrestricted) resolution in the definition of a refinement. The a posteriori ordering is stronger than an a priori ordering (defined on literals before application of the m.g.u): Using $<_d$ as an a priori ordering the resolvent C_4 in example 4.2 would be allowed, as there is no order relation w.r.t. $<_d$ among the literals in C_1 and C_2.

Definition 4.5 An A-ordering deduction is a resolution deduction $C_1, \ldots, C_n$ s.t. it holds: If C_i is a resolvent of C_j, C_k then $C_i \in R_{<_A}(\{C_j, C_k\})$.

Example 4.3

$$\mathcal{C} = \{\{P(a)\}, \quad \{\neg P(x), R(x))\}, \quad \{\neg R(y), R(f(y))\}, \{\neg R(f(f(a)))\}\}$$

The following R-deduction (represented in tree form) is a $<_d$-refutation:



Note that by starting from $\{P(a)\}$ and resolving from "left to right" we do not get an $<_d$-deduction.

Theorem 4.1 *A-ordering refinements are complete, i.e. if $\mathcal{C}$ is unsatisfiable then*

$$\square \in R^*_{<_A}(\mathcal{C}) \text{ for every A-ordering } <_A .$$

Proof: [KH69].

Some remarks on the proof of theorem 4.1: As usual completeness is proved by showing that 1) $<_A$-ground resolution is complete and 2) every ground $<_A$-deduction can be lifted to a $<_A$-deduction on the general clause level. It is for 2) that the property (A3) in the definition of an A-ordering is really needed. Note that (A3) cannot be replaced by the condition: $A <_A B$ implies $not(B\vartheta <_A A\vartheta)$ (a rather inexact but intuitively attractive form would be $A <_A B \Rightarrow A\vartheta \leq_A B\vartheta$). To illustrate this problem we define the following ordering:

$$A <_v B \quad \text{iff} \quad V(A) \subset V(B).$$

Clearly $<_v$ is irreflexive and transitive and $V(A) \subset V(B)$ implies $V(A\vartheta) \subseteq V(B\vartheta)$ (and therefore not $(B\vartheta <_v A\vartheta)$.

W.r.t. $<_v$ lifting is impossible; consider the following ground $<_v$-refutation (note that $A <_v B$ is impossible for ground atoms A, B).

$$\Gamma' = \{P(a, a), Q(a)\}, \{\neg Q(a)\}, \{P(a, a)\}.$$

Suppose that the general clauses are $\{P(x, y), Q(x)\}, \{\neg Q(z)\}$. $\{P(x, y)\}$ is not a $<_v$-resolvent as $P(x, y) > Q(x)$ ($Q(x)$ being the resolved atom). Therefore lifting Γ' to a $<_v$-deduction Γ is impossible.

While A-orderings are atom orderings defined by syntactical properties of atom formulas, locking (or indexing) is a quite different order type: Every literal gets a number and inherits this number during deduction, where two identical literals may get different numbers; resolution is only allowed on literals having the minimal index inside a clause.

Example 4.4 Let $\mathcal{C}$ be the set $\{C_1, C_2, C_3\}$ of clauses with

$$\begin{aligned} C_1 &= \{\neg P(x,y), P(y,x)\}, \\ C_2 &= \{P(u,a)\}, \\ C_3 &= \{\neg P(v,b)\}. \end{aligned}$$

We have

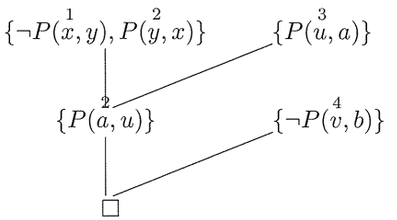
$$R(\mathcal{C}) = \{\{P(a,u)\},\ \{\neg P(b,v)\},\ \{\neg P(x,y), P(x,y)\}\}.$$

Note that it is allowed to resolve C_1 with a variant of itself, giving the tautological clause $\{\neg P(x,y), P(x,y)\}$.

None of the resolvents in $R(\mathcal{C})$ can be excluded by an A-ordering (however the third resolvent could be deleted, because it is a tautology—a deletion method compatible with A-orderings). Particularly $P(u,a) < P(a,u)$ is impossible for every A-ordering because $P(u,a)$ and $P(a,u)$ are unifiable.
But by indexing literals we can change the situation:

$$\{\neg P(\overset{1}{x},y), P(\overset{2}{y},x)\},\ \{P(\overset{3}{u},a)\},\ \{\neg P(\overset{4}{v},b)\}.$$

Under the restriction that only literals with lowest index may be resolved—the other literals are "locked"—we only get the resolvent $\{P(\overset{2}{a},u)\}$ (the index is inherited from the second literal of C_1). So we get



as lock refutation of $\mathcal{C}$.

As numbers are inherited, the order type of a literal within a clause depends on the deduction of the clause. That means this kind of ordering is deduction dependent, while A-ordering is not. In lock resolution a clause cannot simply be defined as a set of literals; instead we need the new concept of indexed literal.

Definition 4.6 A pair (L,i) where L is a literal and i is a number is called an indexed literal. A set of indexed literals is called an indexed clause.

It is not convenient that—within the same indexed clause—one literal appears with two different indices; to overcome such a problem we apply the technique of "merging low".

Definition 4.7 An indexed clause is called reduced if $(L,i) \in C$, $(L,j) \in C$ implies $i = j$.

The method to reduce a clause C is quite simple: Delete every (L,i) from C for which a (L,j) exists s.t. $j < i$. If C is an indexed clause we denote the (unique) reduced indexed clause as $r(C)$. For example we have

$$r(\{(P(x),1),(P(x),4),(Q(x),3),(Q(x),2)\}) \quad = \quad \{(P(x),1),(Q(x),2)\}.$$

Substitutions on indexed literals can be defined in a straight forward manner:

$$\begin{aligned} &(L,i)\vartheta = (L\vartheta, i), \\ &\quad \text{and for clauses} \\ &\{(L_1,i_1),\ldots,(L_n,i_n)\}\vartheta = \{(L_1,i_1)\vartheta,\ldots,(L_n,i_n)\vartheta\}. \end{aligned}$$

Definition 4.8 Let (L,i) be a minimal indexed literal in the indexed clause C and let σ be a m.g.u. of $\{L, L_1, \ldots, L_m\}$ where $(L_1,i_1),\ldots,(L_m,i_m)$ are indexed literals in C. Then $r(C\sigma)$ is called lock factor of C.

For

$$C = \{(P(x),1),(P(y),2),(Q(x),2),(Q(a),3)\}$$

the clause

$$\{(P(x),1),(Q(x),2),(Q(a),3)\}$$

is a lock factor of C, but

$$\{(P(a),1),(P(y),2),(Q(a),2)\}$$

is not.

Definition 4.9 Let C_1, C_2 be variable disjoint indexed clauses; Let C_1', C_2' be lock factors of C_1, C_2 and (L,i) a minimal literal in C_1', (M,j) a minimal literal in C_2'. Suppose that $\{L, M^d\}$ is unifiable by m.g.u. σ; then

$$r((C_1' - \{(L,i)\})\sigma \ \cup \ (C_2' - \{(M,j)\})\sigma)$$

is called lock resolvent of C_1 and C_2.

Similarly to R and $R_{<_A}$ we can define a resolution operator R_{lock}, but we have to take into account that R_{lock} works on indexed clauses.

Theorem 4.2 *Lock resolution is complete, i.e. if $\mathcal{C}$ is unsatisfiable then*

$$\square \in R^*_{\text{lock}}(\mathcal{C})$$

(for every locking of C).

Proof: [Boy71]

It is essential for the completeness of lock resolution that clauses with the same literals but different indices are not identified.

Example 4.5 To improve the readability of indexed clauses we write L^i instead of (L, i).

$$\mathcal{C} = \{\{P^1(a), R^2(a)\};\ \{\neg R^3(a), P^4(a)\},\ \{R^5(a), \neg P^6(a)\},\ \{\neg P^7(a), \neg R^8(a)\}\},$$

$$R^1_{\text{lock}}(\mathcal{C}) = \mathcal{C} \cup \{\{R^2(a), \neg R^8(a)\},\ \{P^4(a), \neg P^6(a)\}\},$$

$$R^2_{\text{lock}}(\mathcal{C}) = R^1_{\text{lock}}(\mathcal{C}) \cup \{\{P^4(a), \neg R^8(a)\},\ \{\neg P^6(a), \neg R^8(a)\}\}.$$

If we identify

$\{P^4(a), \neg R^8(a)\}$, and

$\{\neg R^3(a), P^4(a)\}$, and also the clauses

$\{\neg P^6(a), \neg R^8(a)\}$ and

$\{\neg P^7(a), \neg R^8(a)\}$ then we get

$R^2_{\text{lock}} = R^1_{\text{lock}}$ and thus

$R^*_{\text{lock}} = R^1_{\text{lock}}$.

But $\mathcal{C}$ is unsatisfiable and $\square \notin R^*_{\text{lock}}(\mathcal{C})$; we see that the identification above destroys completeness. Note that in the clauses $\{P^4(a), \neg R^8(a)\}$and $\{\neg R^3(a), P^4(a)\}$ the "index-status" of the literals is different (P(a) is minimal in the first, but not in the second clause).

By deriving correctly we get the following continuation:

$$R^3_{\text{lock}}(\mathcal{C}) = R^2_{\text{lock}}(\mathcal{C}) \cup \{\{\neg R^8(a)\}, \{\neg P^6(a), \neg R^8(a)\}\},$$

$$\{\neg P^6(a)\} \in R^4_{\text{lock}}(\mathcal{C}),\ \{R^2(a)\} \in R^5_{\text{lock}}(\mathcal{C}),\ \square \in R^6_{\text{lock}}(\mathcal{C}).$$

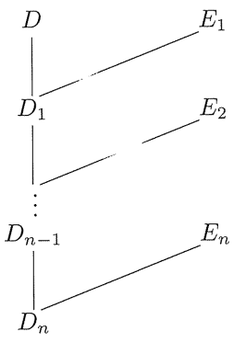
It follows $\square \in R^*_{\text{lock}}(\mathcal{C})$.

Lock resolution is a very efficient refinement, but its weakness consists in its incompatibility with usual deletion methods. We will see in chapter 5 that tautology deletion and subsumption destroy completeness.

4.3 Restrictions on the Form of Deductions

The most natural form of refuting a sentence is to start with the negated conclusion of a theorem and to continue the derivation in a top-down manner.

For illustration consider a theorem $A_1 \wedge \cdots \wedge A_n \rightarrow C$ where the $A'_i s$ are generalized disjunctions of literals and C is of the form $(\exists \overline{x})(L_1 \wedge \cdots \wedge L_m)$ for literals L_i. For a proof by contradiction we consider $A_1 \wedge \cdots \wedge A_n \wedge \neg C$, which corresponds to the set of clauses $\mathcal{C} = \{C_1, \ldots, C_n, D\}$ for $D = \{L^d_1, \ldots, L^d_m\}$. In order to refute $\mathcal{C}$ it is quite reasonable to start with the clause D, as deductions based on $A_1, \ldots, A_n$ only may create lemmata which are of no importance to a proof of C. The following form of deduction guarantees that the negated conclusion D is "present" in every newly deduced clause:



In the deduction above the E_i are either variants of clauses in $\mathcal{C}$ or variants of previously derived D_j for $j < i$.

Definition 4.10 Let $\mathcal{C}$ be a set of clauses and D be a clause in $\mathcal{C}$. A sequence Γ:

$$D_0, E_1, D_1, \ldots, E_n, D_n$$

is called a linear R-deduction of D_n from $\mathcal{C}$ if Γ is an R-deduction of D_n from $\mathcal{C}$ s.t.

a) $D_0 = D$,

b) every D_i is resolvent of E_i and D_{i-1} for $1 \leq i \leq n$.

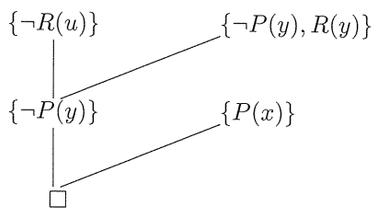
D is called top clause (of Γ), the D_i are called center clauses and the E_i side clauses.

Linear resolution is not complete under all circumstances, because an inadequate choice of the top clause may have bad effects.

Example 4.6 Let $\mathcal{C}$ be the set $\{C_1, C_2, C_3, C_4\}$ of clauses with

$$\begin{aligned} C_1 &= \{P(x)\}, \\ C_2 &= \{\neg P(y), R(y)\}, \\ C_3 &= \{\neg R(u)\}, \\ C_4 &= \{Q(a)\}. \end{aligned}$$

$\mathcal{C}$ is unsatisfiable. But when C_4 is selected as top clause, $\Gamma = C_4$ is the only linear deduction possible; obviously Γ is not a refutation. However, selecting C_3 as top clause we get the linear refutation:



The reason for the effect above is that $\{C_1, C_2\}$ can be considered as a (satisfiable) set of axioms and C_3 as negated conclusion. It is essential that $\{C_1, C_2\}$ is satisfiable, but $\{C_1, C_2, C_3\}$ is unsatisfiable. On the other hand, $\{C_1, C_2, C_3\}$ is already unsatisfiable and, adding C_4, this status is not changed.

Theorem 4.3 *Let $\mathcal{C}$ be an unsatisfiable set of clauses and D be a clause in $\mathcal{C}$ fulfilling the following condition: there exists a subset $\mathcal{D} \subseteq \mathcal{C}$ s.t. $\mathcal{D}$ is satisfiable, but $\mathcal{D} \cup \{D\}$ is unsatisfiable. Then there exists a linear refutation of $\mathcal{C}$ with top clause D.*

Proof: [CL73]

The theorem above shows that we need some knowledge and understanding of the set of clauses (identification of "axioms") before refuting them via linear resolution.

Linear resolution cannot be described by an operator R_{lin}, which is defined on sets of clauses; rather R_{lin} has to be considered as operator on sets of deductions. Thus linear resolution does not fit the set theoretical definition of refinement in section 4.1 (unless we label clauses by deductions).

A further refinement, called linear input resolution, can be obtained by restricting the side clauses E_i in definition 4.10 to input clauses. Linear input deduction, however, is incomplete:

Example 4.7

$$\begin{aligned} \mathcal{C} &= \{C_1, C_2, C_3, C_4\} \\ &= \{\{P(x), Q(x)\},\ \{\neg P(x), Q(x)\},\ \{P(x), \neg Q(x)\},\ \{\neg P(x), \neg Q(x)\}\}. \end{aligned}$$

Select C_4 as top clause. Then C_4 fulfills the requirements of theorem 4.3 and there exists a linear refutation of $\mathcal{C}$. However there can be no linear input refutation of $\mathcal{C}$; the reason is the following: Suppose that Γ is an arbitrary refutation of $\mathcal{C}$. Then $\square$ is obtained by resolving two clauses D_1, D_2 in Γ which either are both unit clauses or possess unit factors; but the input clauses (= the clauses in $\mathcal{C}$) are neither unit nor do they possess unit factors. It follows that Γ cannot be a linear input deduction.

Although linear input deduction is incomplete on clause logic, it is complete on Horn logic (i.e. the class of all sets of clauses consisting of Horn clauses only). The completeness of linear input deduction easily follows from theorem 4.3.

Theorem 4.4 *Let $\mathcal{C}$ be an unsatisfiable set of Horn clauses. Then there exists a linear input refutation of $\mathcal{C}$ with a negative top clause.*

Proof: Note that every unsatisfiable set of clauses must contain negative clauses. So let $D \in \mathcal{C}$ s.t. $D = D_-$ and let $\mathcal{D}$ be a subset of $\mathcal{C}$ s.t. $\mathcal{D}$ is satisfiable, but $\mathcal{D} \cup \{D\}$ is unsatisfiable. By theorem 4.3 we know that linear deduction is complete. That means there exists a linear refutation $\Gamma = D, E_1, D_1, \ldots, E_n, \square$ of Γ with side clauses E_i. But, as D is negative and all side clauses contain at most one positive literal, all center clauses D_i are negative. Because two negative clauses cannot resolve, all side clauses E_i must be in $\mathcal{C}$; it follows that Γ is a linear input deduction.

While linear input resolution is a restriction of linear resolution, set of support resolution is a generalization. Instead of selection a single top clause D we specify a subset $\mathcal{D}$ of $\mathcal{C}$ (the "set of support") s.t. every resolvent (ever derived) has an ancestor from $\mathcal{D}$. The set of support refinement admits a formalization via a set operator in the sense of section 4.1:

$$\begin{aligned} R^0_{\text{ssp}}(\mathcal{C}) &= \mathcal{C}, \mathrm{SP}^0(\mathcal{C}) = \mathcal{D}, \\ R^{i+1}_{\text{ssp}}(\mathcal{C}) &= R^i_{\text{ssp}}(\mathcal{C}) \cup \text{set of all resolvents from clauses in } R^i_{\text{ssp}}(\mathcal{C}) \text{ where at} \\ &\qquad \text{least one parent is in } \mathrm{SP}^i(\mathcal{C}), \\ \mathrm{SP}^{i+1}(\mathcal{C}) &= SP^i(\mathcal{C}) \cup (R^{i+1}_{\text{ssp}}(\mathcal{C}) - R^i_{\text{ssp}}(\mathcal{C})), \\ R^*_{\text{ssp}}(\mathcal{C}) &= \textstyle\bigcup_{i=0}^{\infty} R^i_{\text{ssp}}(\mathcal{C}). \end{aligned}$$

The completeness of the set of support refinement depends on the semantical status of the set of support $\mathcal{D}$.

Theorem 4.5 *Let $\mathcal{C}$ be an unsatisfiable set of clauses and $\mathcal{D} \subseteq \mathcal{C}$ s.t. $\mathcal{C} - \mathcal{D}$ is satisfiable. Then there exists a set of support refutation $\mathcal{C}$ with set of support $\mathcal{D}$ (for $\mathrm{SP}^0(\mathcal{C}) = \mathcal{D}$ we obtain $\square \in R^*_{\text{ssp}}(\mathcal{C})$).*

Proof: Because $\mathcal{C} - \mathcal{D}$ is satisfiable and $\mathcal{C}$ is unsatisfiable there exists a $D \in \mathcal{D}$ and a subset $\mathcal{F} \subseteq \mathcal{D} - \{D\}$ s.t. $(\mathcal{C} - \mathcal{D}) \cup \mathcal{F}$ is satisfiable, but $(\mathcal{C} - \mathcal{D}) \cup \mathcal{F} \cup \{D\}$ is unsatisfiable. Thus by theorem 4.3 there exists a linear refutation Γ of $\mathcal{C}$ with top clause D. But the set of all clauses derivable by linear deductions with top clause D is also set of support - derivable, i.e. it is contained in $R^*_{ssp}(\mathcal{C})$ w.r.t. set of support $\mathcal{D}$. It follows $\square \in R^*_{\text{ssp}}(\mathcal{C})$.

The proof of completeness for theorem 4.3 itself shows the usual pattern of argumentation: Prove that for a finite, unsatisfiable set of ground instances $\mathcal{C}'$ of $\mathcal{C}$ ($\mathcal{C}'$ exists due to Herbrand's theorem) there is a linear refutation Γ' of $\mathcal{C}'$; afterwards lift Γ' to a refutation Γ of $\mathcal{C}$.

4.4 Semantic Clash Resolution

Clash resolution is a variant of resolution where several resolution steps are contracted into one single inference step. The single resolution steps in such a "macro"-inference form a linear deduction and are defined by some "semantical" conditions. There are several different definitions of semantic resolution; the most general one is from Slage [Sla67], but we base our consideration on a concept which is closer to Robinson's hyperresolution [Rob65a].

The first step consists in defining specific Herbrand interpretations which we call settings.

Definition 4.11 Let $\mathcal{C}$ be a set of clauses and $\{P_1, \ldots, P_n\}$ be the set of all predicate symbols occurring in $\mathcal{C}$. A setting is a Herbrand interpretation which, for every P_i, assigns all ground atoms $P_i(t)$ to true or all $P_i(t)$ to false ($\bar{t}$ is a ground term vector of appropriate arity). The setting which assigns true (false) for all $P_i(\bar{t})$ is called positive (negative) setting and is denoted by $\mathcal{M}_p(\mathcal{M}_n)$.

Settings represent some kind of "propositional" interpretation. Indeed, if propositional clause logic is considered, the concept of setting coincides with that of interpretation; this is not the case for predicate logic.

Example 4.8

$$\begin{array}{lcl} \mathcal{C} & = & \{\{P(a)\}, \{\neg P(x), P(f(x))\}, \{\neg P(f(f(a)))\}\}; \\ H & = & \{a, f(a), f(f(a)), \ldots\}. \end{array}$$

$\mathcal{M}_p = \{P(t) | t \in H\}$, $\mathcal{M}_n = \{\neg P(t)/t \in H\}$ are the only possible settings for $\mathcal{C}$.

$$\mathcal{M} = \{P(a), \neg P(f(a)), P(f(f(a))), \ldots, P(f^{2i}(a)), \neg P(f^{2i+1}(a)), \ldots\}$$

is a Herbrand interpretation, but not a setting.

With respect to $\mathcal{M}_p$ $\{P(a)\}, \{\neg P(x), P(f(x))\}$ are true, but $\{\neg P(f(f(a)))\}$ is false.

Definition 4.12 Let $\mathcal{C}$ be a set of clauses and $\mathcal{M}$ be a setting for $\mathcal{C}$. A resolvent of two clauses C_1, C_2 in $\mathcal{C}$ is called $\mathcal{M}$-resolvent if one of C_1, C_2 is false in $\mathcal{M}$ (note that, by definition of a setting, two clauses both false in $\mathcal{M}$ are not resolvable).

For $\mathcal{C}$ and $\mathcal{M}_p$ from example 4.8 $\{\neg P(f(a))\}$ is a $\mathcal{M}_p$ resolvent, but $\{P(f(a))\}$ is not. The idea of semantic clash resolution is to produce clauses only which are false in a setting $\mathcal{M}$; this is not always possible within a single resolution step and thus clusters, so called clashes, are resolved.

Definition 4.13 A semantic clash sequence is a sequence γ of the form

$$(C; D_1, \ldots, D_n)$$

where $C, D_1, \ldots, D_n$ are clauses in some set of clauses $\mathcal{C}$, C is true and all D_i are false in a setting $\mathcal{M}$ for $\mathcal{C}$. C is called nucleus, the D_i electrons of γ. Let $R_0 = C$ R_{i+1} = a $\mathcal{M}$-resolvent of R_i and D_{i+1} for $i < n$ (if it exists). If R_n (defined by a n-step linear deduction with top clause C) is false in $\mathcal{M}$ (R_n may be $\square$) then R_n is called a semantic clash resolvent of γ w.r.t. $\mathcal{M}$.

Example 4.9

$$\mathcal{C} = \{C_1, C_2, C_3\} = \{\{P(x), Q(x,y), \neg Q(x, g(y)), \neg Q(x, f(y))\}, \{Q(u,v), R(v)\}, \{Q(w, g(w)), S(g(w))\}\}.$$

We define $\mathcal{M}_n = \{\neg P(s), \neg Q(s,t), \neg R(s)/s, t \in H(\mathcal{C})\}$.
Then $\gamma = (C_1; C_2, C_3)$ is a clash sequence w.r.t. $\mathcal{M}_n$ and

$$\begin{array}{lcl} R_0 & = & C_1 \\ R_1 & = & \{P(x), Q(x,y), R(f(y)), \neg Q(x, g(y))\} \\ R_2 & = & \{P(x), Q(x,x), R(f(x)), S(g(x))\} \end{array}$$

R_2, being purely positive, is false in $\mathcal{M}$ and is a semantic clash resolvent of γ. Note that there are two possibilities to define R_1 out of γ: Resolving the literals $\neg Q(x, g(y))$ and $Q(u,v)$ we get $R_1' = \{P(x), Q(x,y), R(g(y)), \neg Q(x, f(y))\}$; but R_1' cannot be resolved with C_3. We see that, in this case, γ defines only one clash resolvent, although generally there may be several of them.

The most common semantic clash refinements are positive – and negative hyperresolution. Positive hyperresolution is semantic clash resolution based on $\mathcal{M}_n$ (clash resolvents consist of positive literals only), while negative hyperresolution is based is based on $\mathcal{M}_p$ (clash resolvents consist of negative literals only).

In the notation of resolution operators we define:
$R_{scm}(\mathcal{C})$ = set of a clash resolvents w.r.t. $\mathcal{M}$ which are definable in $\mathcal{C}$, and –as always–

$$R^0_{scm}(\mathcal{C}) = \mathcal{C}/_{\sim_v},\ R^{i+1}_{scm}(\mathcal{C}) = (R^i_{scm}(\mathcal{C}) \cup R_{scm}(R^i_{scm}(\mathcal{C})))/_{\sim_v},$$

$$R^*_{scm}(\mathcal{C})) = \bigcup_{i=0}^{\infty} R^i_{scm}(\mathcal{C}).$$

Theorem 4.6 *Semantic clash resolution is complete, i.e. for every unsatisfiable set of clauses $\mathcal{C}$ and for every setting $\mathcal{M}$ for $\mathcal{C}$ it holds $\square \in R^*_{scm}(\mathcal{C})$.*

Proof: [Sla67] – via Herbrand's theorem, completeness on sets of ground clauses and lifting.

Positive hyperresolution plays an important role in Horn logic. Horn logic is the class of clause sets Γ, where for every $\mathcal{C} \in \Gamma$ and every $C \in \mathcal{C}$, C contains at most one positive literal. It is easy to verify that Horn logic is closed under resolution. As

positive hyperresolution produces positive clauses (i.e. clauses with positive literals only) and nothing else, and positive clause are unit- in Horn we get in fact a unit-clause production method. In fact $R^*_{scm_n}(\mathcal{C}) - \mathcal{C}_0$ ($\mathcal{C}_0$ being the set of nonpositive clauses in $\mathcal{C}$) is a set of unit clauses; if $\mathcal{C}$ is satisfiable then this set may be interpreted as a (minimal) Herbrand model of $\mathcal{C}$. Thus if $R^*_{scm_n}(\mathcal{C})$ is finite and $\Box \notin R^*_{scm_n}(\mathcal{C})$ we know (by the completeness of R_{scm_n} that $\mathcal{C}$ is satisfiable and-at the same time-have a description of a Herbrand model of $\mathcal{C}$. Even in case of non-Horn clause sets, semantic clash resolution is a powerful decision procedure and model building procedure for some classes of clause sets, an aspect which will be discussed further in chapter 6.

In case of Horn logic R_{scm_n} may be refined further: Nuclei can be considered as lists, rather than sets, of clauses and only the last negative literal has to be resolved away (in every stage of the clash resolution). This creates the fortunate effect that every clash sequence defines at most one resolvent.

Chapter 5

Deletion Methods

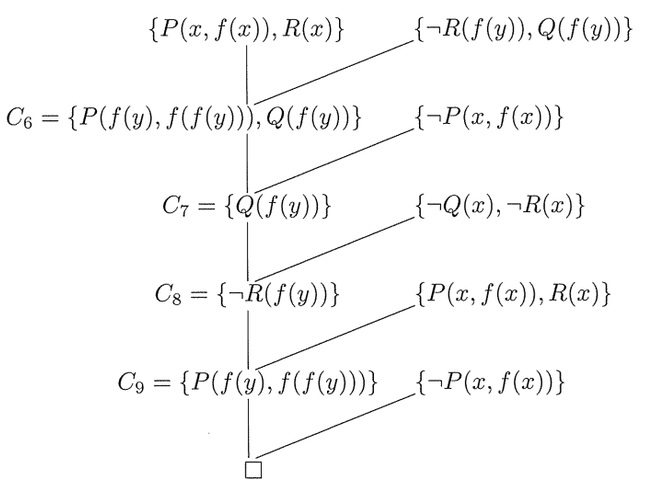
5.1 Subsumption

During the search for a refutation a theorem prover generates a lot of useless, redundant clauses even when strong refinements are applied. There is a specific kind of redundancy which can easily be recognized in clause logic: Because the philosophy of resolution is to work on the most general level only, clauses which are instances (or contain instances) of previously derived clauses are useless (in the sense that they cannot lead to shorter refutations).

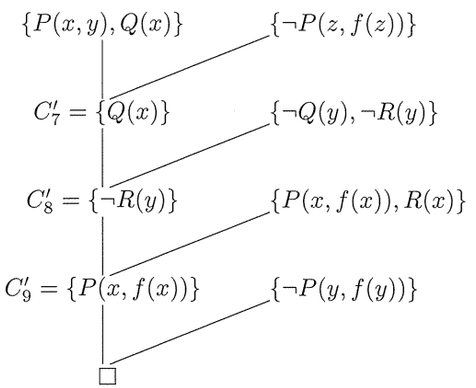
Example 5.1 Let $\mathcal{C}$ be the set $\{C_1, C_2, C_3, C_4, C_5\}$ of clauses with

$$\begin{array}{lcl} C_1 & = & \{P(x, f(x)), R(x)\}, \\ C_2 & = & \{P(x, y), Q(x)\}, \\ C_3 & = & \{\neg R(f(x)), Q(f(x))\}, \\ C_4 & = & \{\neg Q(y), \neg R(y)\}, \\ C_5 & = & \{\neg P(x, f(x))\}. \end{array}$$

Consider the following refutation Γ:



Now consider the clause C_6 of the refutation. C_6 is a substitution instance of C_2, namely $C_2\{x \leftarrow f(y), y \leftarrow f(f(y))\}$. Thus there is no reason to continue with C_6, as using C_2 directly leads to the shorter refutation Δ:



Δ is not only shorter than Γ, but also more general than the corresponding segment Γ_1 of Γ which starts with C_6; indeed every clause in Γ_1 is an instance of the corresponding clause in Δ.

Using the subsumption principle we recognize that C_6 is redundant (w.r.t C_2) and we don't continue the derivation with C_6. There are different ways to modify the derivation: Either we try to derive a new resolvent or we replace C_6 by C_2.

In general we will delete clauses not only if they are instances of other clauses, but also if they contain instances of other clauses. Note that if $C\vartheta \subseteq D$ for two clauses C, D then D is redundant w.r.t. C: First we see that $F(C) \rightarrow F(C\vartheta)$ is valid (where $F(E)$ is the formula corresponding to the clause E), then that $F(C\vartheta) \rightarrow F(D)$ is valid (remember that a clause represents a generalized disjunction). We are lead to the following definition:

Definition 5.1 A clause C subsumes a clause D (we write $C \leq_s D$) if there is a substitution ϑ s.t. $C\vartheta \subseteq D$.

A necessary condition for the applicability of subsumption within a deduction process is the decidability of the subsumption property. In fact subsumption is decidable, but NP-complete [GJ79]; consequently all known algorithms are (at least) exponential. Because subsumption tests have to be performed very often in order to be effective, the efficiency of subsumption algorithms is of central importance to the performance of theorem provers. A standard algorithm is that of Stillman [Sti73] where ϑ is searched via depth first search; because of useless backtracking this algorithm can become very expensive, especially when long clauses are involved. There are faster algorithms based on the divide and conquer method [GL85].

The subsumption principle is not only a method to control proof search, but it can also be considered as a resolution refinement because of the following fact: If $\mathcal{C}$ is unsatisfiable then there exists a nonredundant refutation Γ of $\mathcal{C}$, i.e. no clause C appearing in Γ is subsumed by a clause appearing before C in Γ.
Reconsider example 5.1:

$$\Gamma = C_1, C_2, C_3, \ldots, C_9, \square \text{ is redundant because } C_2 \leq_s C_6,$$

while

$$\Delta = C_1, C_2, C_3, C_4, C_5, C'_7, C'_8, C'_9, \square \text{ is not redundant.}$$

Subsumption can also be used as a pretext on a set of clauses $\mathcal{C}$, i.e. to remove subsumed clauses form $\mathcal{C}$ before the derivation starts. The logical justification is obvious: If C, D are in $\mathcal{C}$, $C \neq D$ and $C \leq_s D$ then $F(\mathcal{C}) \leftrightarrow F(\mathcal{C} - \{D\})$ is valid, because $F(C) \rightarrow F(D)$ is valid.

Using subsumption as pretext serves a double purpose: First it helps to reduce redundancy in derivations and second it reduces the space of the representation (which is essential to clausal knowledge bases).

There are the following ways to apply subsumption in proof search:

1. forward subsumption
2. backward subsumption
3. (total) replacement

In forward subsumption newly derived clauses are removed if subsumed by clauses derived earlier. In backward subsumption clauses which are derived earlier, but are subsumed by clauses derived later, are set inactive (till the search removes the subsumption relation). In replacement, the set of derived clauses is completely reduced under subsumption in every stage of the deduction. These different methods can be modelled in the operator description for resolution. We treat the cases 1. and 3. only. We first define $R_x - sub$, the operator corresponding to the refinement R_x combined with replacement.
Let $sub(\mathcal{C}) = \mathcal{C}$ after removal of subsumed clauses. $sub(\mathcal{C})$ is not per se unique, but can be made unique by some specific selection strategy).
In $sub(\mathcal{C})$ there are no clauses C_1, C_2 s.t. $C_1 \neq C_2$ and $C_1 \leq_s C_2$. We define

$$\begin{aligned} R_x^0 - sub(\mathcal{C}) &= sub(\mathcal{C}), \\ R_x^{i+1} - sub(\mathcal{C}) &= sub(R_x(R_x^i - sub(\mathcal{C})) \cup R_x^i - sub(\mathcal{C})), \\ R_x^* - sub(\mathcal{C}) &= \bigcup_{i=0}^{\infty} R_x^i - sub(\mathcal{C}) \end{aligned}$$

$R_x - sub$ models a very strong deletion method: Every subsumed clause is removed no matter what is its position within the derivation. If C subsumes a clause D

derived before, then a completely new derivation with C instead of D is started. $R_x - sub$ clearly is correct, but is impossible to extract a refutation of $\mathcal{C}$ directly if $\square$ is found in $R_x^* - sub(\mathcal{C})$. Note that applying sub to $\mathcal{C}$ as preprocessing is always possible. Forward subsumption is weaker, but yields refutations of $\mathcal{C}$ directly. For forward subsumption we introduce:

$$\begin{aligned} sf(\mathcal{C},\mathcal{D}) &= \{D/D\in\mathcal{D},(\nexists C\in\mathcal{C})C\leq_s D\} \text{ and}\\ R_x^0 - fs(\mathcal{C}) &= \mathcal{C},\\ R_x^{i+1} - fs(\mathcal{C}) &= R_x^i - fs(\mathcal{C}) \cup sf(R_x^i - fs(\mathcal{C}), R_x(R_x^i - fs(\mathcal{C}))),\\ R_x^* - fs(\mathcal{C}) &= \bigcup_{i=0}^{\infty} R_x^i - fs(\mathcal{C}). \end{aligned}$$

$R_x - fs$ is a refinement in the sense of chapter 4, while $R_x - sub$ is not. The reason is that $R_x^i - sub$ is not necessarily monotonic; indeed we may have

$$R_x^i - sub(\mathcal{C}) - R_x^{i+1} - sub(\mathcal{C}) \neq \emptyset.$$

As $\square \leq_s C$ for all clauses C, we even have:

$$\square \in R_x^i - sub(\mathcal{C}) \text{ implies } R_x^i - sub(\mathcal{C}) = \{\square\}.$$

Subsumption is compatible with many refinements, that is: completeness is preserved under subsumption. E.g.,

$$R_{<_A} - fs, R_{\text{ssp}} - fs, R_{sc} - sub \text{ (and thus } R_{sc} - fs)$$

are all complete. But forward subsumption is not compatible with lock resolution:

Let unlock(C) = the "unlocked" clause corresponding to the indexed clause C.

Suppose now, we define $C \leq_s D$ iff unlock $(C) \leq_s$ unlock (D). With this definition of subsumption for indexed clauses, $R_{lock} - sf$ is incomplete, as the following example shows.

Example 5.2 (compare to example 4.5)
Let

$$\mathcal{C} = \{\{P^1(x), R^2(x)\},\ \{\neg R^3(y), P^4(y)\},\ \{R^5(u), \neg P^6(u)\},\ \{\neg P^7(v), \neg R^8(v)\}\}.$$

We compute $R_{lock}^i - fs(\mathcal{C})$:

$$R_{lock}^1 - fs(\mathcal{C}) = \mathcal{C} \cup \{\{R^2(x), \neg R^8(x)\},\ \{P^4(y), \neg P^6(y)\}\}.$$

Now

$$R_{lock}(R_{lock}^1 - fs(\mathcal{C})) = R_{lock} - fs(\mathcal{C}) \cup \{\{P^4(x), \neg R^8(x)\},\ \{\neg P^6(y), \neg R^8(y)\}\}.$$

But

$$\{\neg R^3(y), P^4(y)\} \leq_s \{P^4(x), \neg R^8(x)\}$$

and

$$\{\neg P^7(v), \neg R^8(v)\} \leq_s \{\neg P^6(y), \neg R^8(y)\}.$$

Therefore

$$sf(R_{lock}^1 - fs(\mathcal{C}), R_{lock}(R_{lock}^1 - fs(\mathcal{C}))) = \emptyset$$

and thus

$$R_{lock}^2 - fs(\mathcal{C}) = R_{lock}^1 - fs(\mathcal{C}).$$

But $\mathcal{C}$ is unsatisfiable and $\square \notin R_{lock}^* - fs(\mathcal{C})$; it follows that $R_{lock} - fs$ is incomplete.

We can overcome the problem in example 5.2 by omitting the technique of unlocking. For this purpose we define for indexed clauses $C, D : C \leq_s D$ iff there is a ϑ s.t. $C\vartheta \subseteq D$ (in complete analogy to the concept for ordinary clauses). Under this definition lock resolution with forward subsumption $(R_{lock} - fs)$ is complete. (Note that the deletion steps in example 5.2 are impossible now).

5.2 Tautology-Elimination and Condensing

The purpose of subsumption is to test, whether a clause is redundant w.r.t. another clause. But there may be clauses which, in some sense, are absolutely redundant; this is the case for clauses which are tautologies, i.e. which contain a pair of complementary literals.

Example 5.3 Let

$$C = \{P(x)\} \cup D,\ E = \{\neg P(t), P(t)\} \cup F$$

for some t and $V(C) \cap V(E) = \emptyset$. Then there exists a resolvent C_1 of C and E where $C_1 = \{P(t)\} \cup D\{x \leftarrow t\} \cup F$. But $C \leq_s C_1$ and thus C_1 is redundant.

In most resolution refinements deletion of tautologies does not destroy completeness. In fact it is easy to realize that all refinements which admit forward subsumption are also compatible with tautology elimination. Like for subsumption, lock resolution does not admit tautology elimination. First we define $R_x - T$, the refinement R_x combined with elimination of tautologies:

- $R_x^0 - T(\mathcal{C}) = TAUTEL(\mathcal{C})$, where $TAUTEL(\mathcal{C}) = \mathcal{C}$ after deletion of all tautological clauses in $\mathcal{C}$.
- $R_x^{i+1} - T(\mathcal{C}) = R_r^i - T(\mathcal{C}) \cup TAUTEL(R_x(R_x^i - T(\mathcal{C})))$.

Let us consider example 5.2 again.
In the first generation the lock resolvents $\{R(x), \neg R(x)\}$ and $\{P(y), \neg P(y)\}$ are both tautologies. Therefore

$$R^1_{lock} - T(\mathcal{C}) = \mathcal{C} \quad \text{and} \quad R^*_{lock} - T(\mathcal{C}) = \mathcal{C}$$

for $\mathcal{C}$ from example 5.2; but $\mathcal{C}$ is unsatisfiable and thus $R_{lock} - T$ is incomplete. Most of the usual refinements admit forward subsumption and deletion of tautologies. If R_x is such a refinement, then we get complete refinements of type $R_x - fsT$:

$$\begin{aligned} R^0_x - fsT(\mathcal{C}) &= \mathcal{C} \\ R^{i+1}_x - fsT(\mathcal{C}) &= R^i_x - fsT(\mathcal{C}) \cup TAUTEL(R^{i+1}_x - fs(\mathcal{C})). \end{aligned}$$

$R_x - fsT$ is a refinement in the sense of chapter 4; If $\mathcal{C}$ is reduced under tautology elimination then $R^*_x - fsT(\mathcal{C})$ is free of tautological clauses. All complete refinements $R_x - fs$ mentioned in section 5.1 also yield complete refinements $R_x - fsT$.

While a tautology is also "semantically" redundant, there may be also some syntactical redundance in representations of clauses; that is clauses of specific forms can be replaced by shorter logically equivalent clauses without affecting completeness.

Example 5.4 Suppose that the clause $C = \{P(x,z), P(f(y),z), Q(z)\}$ appears in a set of clauses $\mathcal{C}$ or was derived from $\mathcal{C}$. Then

$$C' = \{P(f(y),z), Q(z)\}$$

is a factor of C which, at the same time, is a subclause of C. Because $F(C') \rightarrow F(C)$ and $F(C) \rightarrow F(C')$ are both valid, C and C' are logically equivalent. C' is a "condensed" version of C and we may replace C by C'; note that this effect does not occur if the literal $Q(z)$ in C is replaced by $Q(x)$.

Definition 5.2 Let C be a clause. C' is called a condensation of C if C' is a smallest instance of C which is also contained in C.

Condensation is not unique ($\{P(x)\}$ and $\{P(y)\}$ are both condensations of

$$\{P(x), P(y)\}),$$

but different condensations are always variants of each other.

Although rarely used in "classical" theorem proving, condensing plays an important role in resolution decision procedures, as it can prevent unlimited growth of clauses; as an example consider the sequence of clauses

$$\{P(x,x_1), P(x,x_2), \ldots, P(x,x_n)\}$$

having all the condensation $\{P(x,x_1)\}$.

Formally we can define $cond(\mathcal{C})$ as the set of all condensed clauses defined by clauses in $\mathcal{C}$. If ρ is an arbitrary refinement we define

$$\rho - cond(\mathcal{C}) = cond(\rho(\mathcal{C})).$$

$\rho - cond$ is no longer a refinement in the strict sense, because $\rho - cond(\mathcal{C})$ generally is not contained in $R^*(\mathcal{C})$ (Syntactically new clauses may be built which cannot be derived by ordinary resolution).

Condensing is compatible with A-ordering refinements [Joy76] and semantic clash resolution [FLTZ92], i.e. $R_{<A'} - fsTcond$ and $R_{sc} - subcond$ are complete. Results about compatibility of condensation are still rare, such as about its performance. At least it can be said that to compute a condensation is at least as expensive as the subsumption test [GF92].

5.3 Clause Implication

We have observed that $C \leq_s D$ implies $F(C) \rightarrow F(D)$; it is natural to ask whether the converse holds as well and whether subsumption and implication for clauses are the same concepts. We will see in this chapter that this is not (at all) the case and that there are barriers to the extension of redundancy tests.

First consider the tautology $D = \{\neg P(x), P(x)\}$ and the clause $C = \{Q(y)\}$. C does not subsume D, but $F(C) \rightarrow F(D)$ is clearly valid. So one might guess that tautologies are an exception and that, for "reasonable" clauses, subsumption and implication coincide. Again there is the following counterexample:

Example 5.5

$$\mathcal{C} = \{\neg P(x), P(f(x))\},\ D = \{\neg P(a), P(f(f(a)))\}.$$

C does not subsume D because there is no ϑ with $C\vartheta \subseteq D$. On the other hand $F(C) \rightarrow F(D)$ is valid. To prove this fact produce the two substitution instances

$$\{\neg P(a), P(f(a))\},\ \{\neg P(f(a)), P(f(f(a)))\}$$

from $\{\neg P(x), P(f(x)\}$; then propositional resolution gives $\{\neg P(a), P(f(f(a))\}$.

In example 5.5 C is capable of resolving with a renamed copy of itself. The obtained resolvent $E = \{\neg P(x), P(f(f(x)))\}$ is not subsumed by C, but $E \leq_s D$. Generally it can be shown that, for nontautological clauses D, $F(C) \rightarrow F(D)$ is valid iff there is a clause $E \in R^*(\{C\})$ s.t. $E \leq_s D$. This property follows from the theorem of Lee [Lee67], which states that for sets of clauses $\mathcal{C}$ and nontautological clauses D: $F(\mathcal{C}) \rightarrow D$ iff there exists an $E \in R^*(\mathcal{C})$ s.t. $E \leq_s D$.

Clause implication can be used as preprocessing in order to reduce redundancy before derivation. But unfortunately (and contrary to subsumption) clause implication is undecidable [Sch88]. Being undecidable, clause implication in its general form can hardly be applied as "realistic" redundancy principle in clause logic. However there are many cases where clause implication is decidable. First of all, consider

$F(C) \wedge \neg F(D)$ as negated form of the problem. Clearly $F(C) \rightarrow F(D)$ is valid iff the Skolemized form of $F(C) \wedge \neg F(D)$ is unsatisfiable; note that the Skolemized form of $F(C) \wedge \neg F(C)$ is $\{C, \{L'_1\}, \ldots, \{L'_m\}\}$ where the $\{L'_i\}$ are unit ground clauses. So we have transformed the clause implication problem to a satisfiability problem in clause logic.

For example 5.5 we get the set of clauses

$$\mathcal{C} = \{\{\neg P(x), P(f(x))\}, \{P(a)\}, \{\neg P(f(f(a)))\}\}.$$

We prove that C implies D by deriving a contradiction from $\mathcal{C}$. E.g.,

$$\{\neg P(x), P(f(x))\},\ \{\neg P(f(f(a)))\},\ \{\neg P(f(a))\},\ \{\neg P(a)\},\ \{P(a)\},\ \Box.$$

As the clause implication problem is undecidable and resolution is complete, we cannot expect that resolution always terminates on satisfiable clause sets $\mathcal{C}$ representing clause implication. More concretely: For every complete resolution refinement R_x there must be a satisfiable set of clauses $\mathcal{C}$ (representing $F(C) \rightarrow F(D)$) s.t. $R^*_x(\mathcal{C})$ is infinite. In chapter 6 we will investigate resolution as decision procedure (on decidable classes Γ) and guarantee termination of complete refinements R_x on Γ.
One such decidable class is

$$\Gamma = \{\mathcal{C} \mid \text{ for all } C \in \mathcal{C} : |V(C)| \leq 1\};$$

it follows that $F(C) \rightarrow F(D)$ is decidable for

$$|V(C)| \leq 1 \quad (\{C\} \cup clf(\neg D) \text{ is in } \Gamma).$$

Very recently the Horn clause implication problem was shown undecidable [MP92]; formally this decision problem is represented by $\{(C, D) \mid C \text{ Horn}, F(C) \rightarrow F(D)\}$. The Horn clause implication problem, naturally occurs in the attempt to reduce redundancy of logic programs. By representing the Horn clause implication problem as clausal class we get forms $\mathcal{C} = \{C, \{K'_1\}, \ldots, \{K'_m\}\}$ where C is a Horn clause and the K'_i are unit ground clauses. Some subclasses of the Horn clause implication problem have been decided by means of positive hyperresolution. As $\mathcal{C}$ is a set of Horn clauses with the only nucleus C, positive hyperresolution only derives unit clauses which are instances of the positive literal in C; this makes it possible (for some forms of C) to compute uniform depth-bounds for the unit clauses to be computed and thus to get a decision procedure. For details we refer to [Lei90], [Rud91].

The clause implication problem $F(C) \rightarrow F(D)$ is trivial if C does not resolve with a renamed copy of itself. In this case the theorem of Lee yields that C must subsume D (as $R^*(\{C\}) = \{C\}$) or D is a tautology. Thus for "nonresolving" C clause implication essentially coincides with subsumption.

It is not investigated how clause implication (provided it can be tested effectively) behaves as redundancy technique during deduction. It might be that clause implication destroys completeness, where subsumption is allowed. There are some trivial

cases where clause implication may be applied like forward subsumption, simply because it must coincide with subsumption.

In positive hyperresolution only positive clauses are deduced. But a positive clause can only be implied by another positive clause (for C nonpositive, $R^*(\{C\})$ does not contain positive clauses). But because positive clauses are incapable of self-inference the problem reduces to subsumption. A similar situation (only negative clauses) holds for linear input resolution with negative top clause in Horn logic.

Chapter 6

Resolution as Decision Procedure

6.1 The Proof Theoretical Approach to the Decision Problem

In chapter 3 we have seen that for every unsatisfiable set of clauses there exists a resolution refutation. In chapter 4 and 5 we have shown, how the deduction concept can be refined under preserving completeness. Suppose now, we start a theorem prover on an arbitrary set of clauses, possibly on a satisfiable one; satisfiable sets of clauses occur naturally in clausal knowledge bases or in theorem proving problems which are not sufficiently axiomatized. From Church's result [Chu36] we know that (the validity problem of) predicate logic is undecidable; it is easily verified that the validity- and the satisfiability problem are recursively equivalent. Now let F be a sentence of predicate logic. By techniques defined in chapter 3 we can transform F to a set of clauses $\mathcal{C}$ s.t. F and $\mathcal{C}$ are satisfiability equivalent. An immediate consequence is the undecidability of the satisfiability problem for clause logic (in shorthand: clause logic is undecidable). As resolution is complete for clause logic we can conclude that there must exist a set of clauses $\mathcal{C}$ s.t. $R^*(\mathcal{C})$ is infinite and $\square \notin R^*(\mathcal{C})$. (Otherwise we would get a decision procedure).

Moreover there can be no (recursively defined) refinement R' s.t. R' is complete and $R'^*(\mathcal{C})$ is finite on all satisfiable set of clauses $\mathcal{C}$. This fundamental negative result does not imply that it is senseless at all to use resolution as decision procedure; instead we have to focus on decidable subclasses of clause logic and to investigate the behaviour of resolution on such classes.

The area of mathematical logic commonly named the "decision problem" originated in the begin of this century. One of the first results on decidability of predicate logic classes was obtained by Löwenheim in 1915 [Löw15]; he proved the decidability of the monadic class, i.e. the subclass of closed function-free first order formulas with one-place predicate symbols only. In the time between world war I and world war II (and before Church's fundamental result) many new decidable predicate logic classes have been found. We mention the prefix classes (i.e. prenex, closed formula with function free matrix) $\forall\exists^*$ (the Ackermann class [Ack28]), $\forall\forall\exists^*$ (the Gödel-Kalmar- Schütte class [Göd32]) and $\exists^*\forall^*$ (the Bernays- Schönfinkel class [BS28]). The method to prove decidability of these classes was a model theoretic one: Prove that the class (Γ) is finitely controllable, i.e. if some formula in Γ is satisfiable then it also has a finite model. As the PL-formulas having finite models are recursively enumerable such classes are decidable (just run a complete theorem prover and a

model finding procedure in parallel). This model theoretic method of argumentation has two main disadvantages:

a) the proofs are relatively complicated

b) there is no valuable information for the design of decision algorithms (exhaustive search is not the best one can think of).

As already sketched in section 5.3 there exists a proof theoretical alternative in handling the decision problem of a PL-class Γ (we speak about the satisfiability problem): Find a complete refutational calculus Ω for PL s.t. only finitely many derivations are possible on formulas in Γ, then Ω can be used for the following decision procedure:
For $F \in \Gamma$ compute all possible deductions on F (the set Λ); if Λ contains a contradiction then F is unsatisfiable, else F is satisfiable. Such a method (although for the validity problem and thus for the dual prefix class) was used by S.Y.Maslov in 1964 [Mas64] to prove decidability of the class $\exists^*\forall^*\exists^*$Krom (where "Krom" stands for a matrix in Krom form). He used the so called inverse method, which can be considered as a version of the resolution method based on the sequent calculus. A typical feature of his method (and of resolution) is the use of the unification principle. Note that for decision purposes, calculi are needed which cannot produce infinitely many different formulas in one step (a tool to get such a "finiteness"-condition by restriction of the substitution rule is unification). So the unification principle is not only a powerful principle in the design of computational algorithms, but also a useful tool for proving theorems about decidable classes.

In the same spirit as Maslov, but on the basis of the resolution calculus, Joyner showed in his thesis [Joy73] that resolution theorem provers can be used as decision procedures for some classical prefix classes. His idea is basic to all results presented in this chapter: For a (decidable) class Γ (being the clause class corresponding to some PL-class) find a complete resolution refinement R_x s.t. $R_x^*(\mathcal{C})$ is finite for all $\mathcal{C} \in \Gamma$. To get a decision procedure even less would be sufficient (R_x must be complete on Γ and $R_x^*(\mathcal{C})$ must be finite for satisfiable $\mathcal{C} \in \Gamma$). Although Joyner investigated clause forms of prefix classes only, his method can be extended to classes which are not of the "prefix type" (that means prenex and function free). A typical example of such a class is the set of all closed PL-formulas containing only one $\forall$-quantifier and having full functional structure otherwise [Gur73]. In more recent time decidability results for several functional clause classes have been obtained by resolution [FLTZ92]; some of these classes will be discussed in this chapter.

Being complete resolution refinements, resolution decision procedures can be used as "ordinary" theorem provers. Because these refinements are (mostly) very strong and favour the production of clauses having low complexity, they prove to be quite efficient in practice.

6.2 A-Ordering Refinements as Decision Procedures

In section 6.1 we mentioned the Ackermann class $\forall\exists^*$ and the class $\forall M$ where M is an arbitrary matrix which may contain function symbols. By Skolemization we can reduce the decidability of $\exists^*\forall\exists^*$ (the extended Ackermann class) to that of $\forall M$. On the other hand, $\forall M$ contains formulas which cannot be obtained by Skolemization from the class $\exists^*\forall\exists^*$. If, in $\forall M$, M is transformed to conjunctive normal form we obtain the clausal class

$$\text{VAR1} = \{\mathcal{C} \mid \text{for all } C \in \mathcal{C} : |V(C)| \leq 1\};$$

Note that we may rename the variables in different clauses. Thus VAR1 semantically corresponds to the set of all clause sets $\mathcal{C}$ with $|V(\mathcal{C})| = 1$.

Example 6.1 (compare to example 4.3) Let $\mathcal{C}$ be the set $\{C_1, C_2, C_3, C_4\}$ of clauses with

$$\begin{array}{lcl} C_1 & = & \{P(a)\}, \\ C_2 & = & \{\neg P(x), R(f(x))\}, \\ C_3 & = & \{\neg R(y), R(f(y))\}, \\ C_4 & = & \{\neg R(f(f(b)))\}. \end{array}$$

$\mathcal{C}$ is obviously an element of VAR1. It is easy to realize that $\mathcal{C}$ is satisfiable. Moreover $R^*(\mathcal{C})$ is infinite, as for all $n \geq 1$

$$\{R(f^n(a))\} \in R^*(\mathcal{C}).$$

Thus unrestricted resolution does not terminate on $\mathcal{C}$ and consequently cannot serve as decision procedure on VAR1. Positive hyperresolution R^*_{PH} does not terminate either, because (again)

$$\{R(f^n(a))\} \in R^*_{PH} \quad \text{for all } n \geq 1.$$

Now remember the A-ordering $<_d$ defined in section 4.2:
$A <_d B$ iff

(1) $\tau(A) < \tau(B)$ and

(2) For all $x \in V(A)$: $\tau_{max}(x, A) <_d \tau_{max}(x, B)$ (what implies $V(A) \subseteq V(B)$).

$\{R(f(a))\} \notin R^*_{<_d}(\mathcal{C})$. Instead we get

$$R^*_{<_d}(\mathcal{C}) = \mathcal{C} \cup \{\{\neg R(f(b))\},\ \{\neg P(f(b))\},\ \{\neg P(b)\}\ \{\neg R(b)\}\}.$$

So $R^*_{<_d}(\mathcal{C})$ is finite and $\square \notin R^*_{<_d}(\mathcal{C})$, and we have shown that $\mathcal{C}$ is indeed satisfiable. It can be shown that $R_{<_d}$ always terminates on VAR1 [Fer91a] and we get:

Theorem 6.1 *$R_{<_d}$ (+ condensing) is a decision procedure for* VAR1, *i.e. for all $\mathcal{C} \in$ VAR1: $R^*_{<_d}(\mathcal{C})$ is finite.*

Corollary 6.2 *$R_{<_d}$ is a decision procedure for the extended Ackermann class.*

VAR1 can be generalized to a clausal class fulfilling the following conditions: All literals in a clause contain the same variables or are variable disjoint and every function symbol in a literal containing variables contains all of them. More formally we define:

Definition 6.1 A functional term t is called weakly covering iff for all nonground functional subterms s of t we have $V(s) = V(t)$. An atom or literal A is called weakly covering iff each argument of A is either a ground term, a variable or a weakly covering term s.t. $V(t) = V(A)$. Let $E^+ = \{\mathcal{C} \mid 1) \wedge 2)\}$ where

1) for all $C \in \mathcal{C}$: All literals in C are weakly covering, and

2) for all $C \in \mathcal{C}$ and $L, M \in C$: Either $V(L) = V(M)$ or $V(L) \cap V(M) = \emptyset$.

Clearly, E^+ contains VAR1. Unfortunately, we have no proof yet that E^+ can be decided by $R_{<_d}$. Instead one takes a slightly different A-ordering $<_{v_d}$ and combines it with a saturation method. Saturation is a method which, after computation of the set of resolvents, adds a finite set of instances of the resolvents (a set of instances which cannot be generated by resolution or factoring). Saturation techniques are also required to decide the Gödel class $\forall\forall\exists^*$, Maslov's $\forall^*\exists^*$Krom class, and the Skolem class([Joy76], [Fer91]). Saturation is a deviation from the pure resolution paradigm, as substitutions are involved which are not m.g.u.'s. From the classical prefix classes, the monadic class and the extended Ackermann class can be decided by "pure" resolution. The Herbrand class (i.e. the set of all $\mathcal{C}$ consisting of unit clauses only) is even decidable by unrefined resolution (and thus by any refinement). The argumentation is as follows:
Let $\mathcal{C} = \{U_1, \ldots, U_n\}$ be a set of unit clauses. $\mathcal{C}$ is unsatisfiable iff $\Box \in R^*(\mathcal{C})$. But because there are only unit clauses in C, we either have $R(\mathcal{C}) = \emptyset$ or $R(\mathcal{C}) = \{\Box\}$. A simple decision procedure consists in applying the unification algorithm to all sets $\{L_i^d, L_j\}$ where $i \neq j$ and $U_k = \{L_k\}$.

6.3 Semantic Clash Resolution as Decision Procedure

Refinements based on A-ordering or on other ordering refinements do not suffice to get decision procedures for all relevant decision classes. Particularly in the case of the Bernays-Schönfinkel class ($\exists^*\forall^*$–prefix class) and of many functional clausal classes, semantic clash resolution is superior to ordering refinements.

We start our investigations with a subclass of the Bernays-Schönfinkel class, namely the class of all closed formulas of the form $(\exists^*\overline{x})(\forall^*\overline{y})M(\overline{x}, \overline{y})$, where M is a Horn formula. The corresponding clause classes for the Bernays-Schönfinkel class and for the subclass defined above can be defined as:

- BS $= \{\mathcal{C} \mid \tau(\mathcal{C}) = 0\}$ and
- BSH $= \{\mathcal{C} \mid \tau(\mathcal{C}) = 0$, $\mathcal{C}$ is a set of Horn clauses$\}$.

The condition $\tau(\mathcal{C}) = 0$ guarantees that there are no function symbols in $\mathcal{C}$; in fact, the Skolemization of a prefix form $\exists^*\forall^*$ only generates new constant symbols.

There is a trivial proof of the decidability of BS and BSH based on Herbrand's theorem: Take a $\mathcal{C} \in$ BS, compute the set of all ground instances of clauses in $\mathcal{C}$ (this set is finite!) and test this set for satisfiability. In this case (provided the theorem of Herbrand is available), the model theoretic method to prove decidability is clearly superior to the proof theoretic one. However the computation of $\mathcal{C}'$ (the set of all ground instances from clauses in $\mathcal{C}$) may be very expensive from the computational point of view. Moreover BS, although easily proved to be decidable, is of highest computational complexity among the classical prefix classes [DL84].

Example 6.2

$$\begin{aligned} \mathcal{C} &= \{C_1, C_2, C_3, C_4\} \\ &= \{\{P(a,b)\}, \{\neg P(x,y), P(y,x)\}, \{\neg P(x,y), \neg P(y,z), P(x,z)\}, \{\neg P(b,c)\}\}. \end{aligned}$$

C_2 denotes symmetry, C_3 transitivity. $\mathcal{C}$ is satisfiable because $P(b,c)$ cannot be derived from $P(a,b)$ via symmetry and transitivity. It is easy to see that any A-ordering must fail as decision method on $\mathcal{C}$, as no clause deduced from C_2, C_3 can be excluded by the ordering criterion (the literals in the resolvent always unify with the resolved literals). Thus for any A-ordering $<_A$ $R^*_{<_A}(\mathcal{C})$ contains $R^*(\{C_2, C_3\})$ which is infinite; moreover none of the clauses

$$C_n : \{\neg P(x_1, x_2), \neg P(x_2, x_3), \ldots, \neg P(x_{n-1}, x_n), P(x_1, x_n)\} \in R^*(\{C_2, C_3\}),$$

can be removed by subsumption or by condensing. However BSH can be decided easily by R_{scm_n}. In fact

$$R^*_{scm_n}(\mathcal{C}) = \mathcal{C} \cup \{\{P(b,a)\}, \{P(a,a)\}, \{P(b,b)\}\}.$$

Moreover, a well-known fact in data-base theory, the set

$$\{\{P(a,b), \{P(b,a)\}, \{P(a,a)\}, \{P(b,b)\}\}$$

defines a minimal Herbrand model of $\mathcal{C}$.

Positive hyperresolution (R_{scm_n}) decides BSH, because the set of all positive unit clauses U with $\tau(U) = 0$ over the term universe of a set of all positive unit clauses $\mathcal{C}$ is finite (under renaming) and $R^*_{scm_n}(\mathcal{C}) - \mathcal{C}_0$ (where $\mathcal{C}_0$ is the set of nonpositive clauses in $\mathcal{C}$) is a subset of this set. We might hope that semantic clash resolution will do the job for the whole class BS. Unfortunately this is not the case:

Example 6.3

$$\mathcal{C} = \{\{P(x,z,u), \neg P(x,y,u), \neg P(y,z,u)\}, \{P(x,x,a)\},$$
$$\{\neg P(x,z,u), P(x,y,u), P(y,z,u)\}, \{\neg P(x,x,b)\}\}.$$

$\mathcal{C}$ is essentially non-Horn (it cannot be transformed to a set of Horn clause by changing the signs of the literals). Both positive and negative hyperresolution don't terminate on $\mathcal{C}$, because clauses of arbitrary size are derivable which cannot be deleted by subsumption and condensing. Moreover no "standard" refinement is known to terminate on $\mathcal{C}$; an exception is general semantic clash resolution as defined in Slagle's paper [Sla67], which is based on arbitrary interpretations of clause sets.

So we face the fact that no standard resolution refinement decides BS. It is trivial that adding saturation is an escape (because the Herbrand universe is finite). We will see later in this chapter that there exists a nontrivial very limited form of saturation which also does the job.

In the next part of this chapter we characterize some functional clause classes which can be decided by semantic clash resolution. These classes can be considered as generalizations of DATALOG.

Let $\mathcal{M}$ be an arbitrary setting for a set of clauses $\mathcal{C}$. For every $C \in \mathcal{C}$ we define C_{neg} as the maximal subclause D of C s.t. D is false in $\mathcal{M}$; $C_{pos} = C - C_{neg}$. The set of false clauses in $\mathcal{C}$ can then be defined as

$$\mathcal{C}_{neg} = \{C \mid C \in \mathcal{C}, C_{\text{neg}} = C\},$$

the complementary set as

$$\mathcal{C}_{pos} = \mathcal{C} - \mathcal{C}_{neg}.$$

Definition 6.2 A set of clauses $\mathcal{C}$ belongs to the class PVD (positive variable dominated) iff there exists a setting $\mathcal{M}$ for $\mathcal{C}$ s.t. for all $C \in \mathcal{C}$:

PVD-1 $V(C_{\text{neg}}) \leq V(C_{\text{pos}})$,

PVD-2 $\tau_{\max}(x, C_{\text{neg}}) \leq \tau_{\max}(x, C_{\text{pos}})$ for all $x \in V(C_{\text{neg}})$.

DATALOG (including negative query clauses) is a simple subset of PVD, as negative setting ($C_+ = C_{\text{neg}}, C_- = C_{\text{pos}}$) fulfills PVD-1, PVD-2. PVD-1 implies that for $C_{\text{pos}} = \emptyset$ we must have $V(C_{\text{neg}}) = \emptyset$, i.e. semantically false clauses are ground. Restricted to BS and $\mathcal{M}_n$ (syntactically) positive clauses must be ground and $V(C_+) \subseteq V(C_-)$ for all clauses in $\mathcal{C}$.

By changing signs appropriately, PVD can be reduced to PVDN s.t.

PVDN-1 $V(C_+) \subseteq V(C_-)$ and

PVDN-2 $\tau_{\max}(x, C_+) \leq \tau_{\max}(x, C_-)$ for all $x \in V(C_+)$.

Theorem 6.3 *Semantic clash resolution decides* PVD, *i.e. for every* $\mathcal{C} \in$ PVD *and every setting* $\mathcal{M}$ *fulfilling* PVD1), PVD2) $R^*_{sc_\mathcal{M}}(\mathcal{C})$ *is finite.*

PVDN is relatively sharp w.r.t. to undecidable classes: If we add the clause

$$T^- = \{\neg P(x,z), P(x,y), P(y,z)\},$$

which does not fulfill PVDN1) we can encode the word problem for arbitrary equational theories. It follows that

$$\Gamma = \{\mathcal{C} \cup \{T^-\} | \mathcal{C} \in \text{PVDN}\}$$

is an undecidable class. The proof of theorem 6.3 consists of two parts: First show that all clauses in $R^*_{sc_\mathcal{M}}(\mathcal{C}) - C_{\text{pos}}$ are ground; afterwards show the existence of a number d s.t. $\tau(C) \leq d$ for all $C \in R^*_{sc_\mathcal{M}}(\mathcal{C})$. Because there are only finitely many ground clauses of fixed depth (over the Herbrand universe of $\mathcal{C}$) $R^*_{sc_\mathcal{M}}(\mathcal{C})$ must be finite.

The class PVD can generalized by replacing PVD2) by the condition $T(C_{\text{neg}}) \leq T(C_{\text{pos}})$ where T is an arbitrary atom complexity measure fulfilling some very general axiomatic properties [Lei92].

Because BS is not a subclass of PVD we cannot hope to decide BS via semantic resolution directly. However there exists an easy method to transform a set $\mathcal{C} \in$ BS into a set $\mathcal{C}' \in$ PVD$\cap$BS under preservation of sat- equivalence. $\mathcal{C}'$ is obtained from $\mathcal{C}$ by limited saturation on clauses which do not fulfill $V(C_{\text{neg}}) \subseteq V(C_{\text{pos}})$. After transformation of $\mathcal{C}$ to $\mathcal{C}'$, $R_{sc\mathcal{M}}$ can be applied as decision procedure on $\mathcal{C}'$.

By these considerations we come to a decision algorithm for the Bernays-Schönfinkel class, listed in Fig. 6.1. It is easy to verify that $\mathcal{C}' \sim_{sat} \mathcal{C}$ and $\mathcal{C}' \in$ PVD; therefore BSALG is indeed a decision procedure for BS: Crucial for the performance can be the adequate selection of the setting $\mathcal{M}$; in case $\mathcal{C} \notin$ PVD some heuristics have to be applied (e.g. minimizing the number of clauses violating PVD1), PVD2)).

If $R^*_{scm_n}(\mathcal{C})$ terminates on a set of Horn clauses $\mathcal{C}$ from PVDN then the derived positive clauses are all unit and ground. In this case we know more than just that $\mathcal{C}$ is satisfiable; here $R^*_{scm_n}(\mathcal{C})$ directly represents a Herbrand model (namely the ground atoms which must be set to true).

6.4 Decision Procedures as Theorem Provers

Experimental results have shown [FLTZ92] that resolution decision procedures can be used as powerful "ordinary" theorem provers. Particularly it turned out, that theorem provers R_x are fast on sets of clauses $\mathcal{C}$ where $\mathcal{C}$ belongs to a decidable class decided by refinement R_x. Of course this observation cannot be transformed into a general (always applicable) method, as clause logic cannot be obtained by a finite union of decidable classes. However it makes sense to proceed as follows:
{Input is a set of clauses $\mathcal{C}$ to be refuted}

BSALG (* input is a set $\mathcal{C} \in$ BSALG *)
begin
Case a) $\mathcal{C} \in$ PVD: $\mathcal{C}' := \mathcal{C}$;
Case b) $\mathcal{C} \notin$ PVD:
begin
Select a setting $\mathcal{M}$ for $\mathcal{C}$;
for all $C \in \mathcal{C}$ compute $T(C)$:
if $V(C_{\text{neg}}) \subseteq V(C_{\text{pos}})$ then
$T(C) := \{C\}$
else
$T(C) := \{C\lambda \mid \text{dom}(\lambda) \subseteq (V(C_{\text{neg}}) - V(C_{\text{pos}})),\ \text{rg}(\lambda) \subseteq H(\mathcal{C})\}$
end if
end for;
$\mathcal{C}' := \bigcup_{C \in \mathcal{C}} T(C)$
end;
Compute $R^*_m(\mathcal{C}')$
end.

Figure 6.1: Decision Procedure for the Bernays-Schönfinkel class

1) Try to locate $\mathcal{C}$ in some decidable class$\Gamma \in CL$ {CL is a finite set of decidable classes}

2) If $\mathcal{C} \in \Gamma$ for some $\Gamma \in CL$ then compute $R^*_\Gamma(\mathcal{C})$ {R_Γ is the refinement deciding Γ}
else

begin {heuristics}
find a $\Gamma \in CL$ which is the "closest" to $\mathcal{C}$
and compute $R^*_\Gamma(\mathcal{C})$
end

Point 1) is computationally easy for most "reasonable" decidable classes Γ, as membership in a class mostly is determined by simple syntactical properties. Even if the syntactical properties are more complex (such as for PVD) it pays out to spend this effort, by which hours of computing time may be saved. Suppose now that we have found a $\Gamma \in$ CL and $\mathcal{C} \in \Gamma$; it is by no means mysterious that R_Γ is fast on $\mathcal{C}$, because a decision procedure necessarily favours the production of clauses having low complexity (term depth and clause length must be bounded, frequently they do not increase at all). If $\mathcal{C} \notin \Gamma$ for all $\Gamma \in$ CL we face the challenge to select a refinement R_Γ although $\mathcal{C}$ does not belong to the corresponding decision class. Although

such a selection must be of heuristical nature (see the procedure defined before), the corresponding heuristic can be logically motivated. Suppose for example that $\mathcal{C} \notin \Gamma$ but $\mathcal{D} \in \Gamma$ for a subset $\mathcal{D} \in \mathcal{C}$; then we know that $R^*_\Gamma(\mathcal{D})$ is finite. Therefore we may "saturate" $\mathcal{D}$ by first computing $R^*_\Gamma(\mathcal{D})$ and then apply a theorem prover R' to $(\mathcal{C} - \mathcal{D}) \cup R^*_\Gamma(\mathcal{D})$ (provided $\square$ is not already in $R^*_\Gamma(\mathcal{D})$).

Chapter 7

Complexity of Resolution and Function Introduction

7.1 The Length of Resolution Proofs

Because clausal predicate logic is undecidable, there can be no recursive bounds on the length of refutations of clause sets $\mathcal{C}$ in terms of the length of $\mathcal{C}$; thereby it does not matter what logical calculus and what concept of length is chosen. Thus we cannot develop a complexity theory similar to the theory of propositional proof systems. But there remain two possible mathematical approaches to predicate logic proof complexity:

a) Analyze the relative complexity of resolution versus other inference methods and

b) Find some absolute complexity measure for sets of clauses which is independent of deduction concepts and which can serve as a basis for complexity analysis.

The book of E.Eder [Ede92] is based on approach a) and gives a lot of relative complexity results for first order calculi (among them also resolution). In [BL92], a paper which mainly focuses on resolution complexity, Herbrand complexity is taken as absolute basic measure. Herbrand complexity is the minimal number of ground clauses (defined by ground instances of clauses in sets of clauses $\mathcal{C}$) required to get (propositional) unsatisfiability. Because, by Herbrand's theorem, every unsatisfiable set of clauses C possesses a finite, unsatisfiable set of ground clauses $\mathcal{C}'$, Herbrand complexity is well-defined and deduction-independent. In analyzing the length of R-refutations relative to Herbrand complexity, the set theoretical clause concept becomes problematic; rather it is convenient to represent a clause as disjunction or as an atomic sequent. Moreover it is advantageous to separate factoring from the resolution cut rule and to split factoring itself into a substitutional and a contraction component. A resolution proof is then defined as sequence of clauses like in definition 3.11. Instead of presenting the whole formal apparatus developed in [BL92] we give some illustrating examples.

Example 7.1

$$\Gamma = \{P(x), P(a), R(x)\}, \{\neg P(y), Q(y)\}, \{R(a), Q(a)\}$$

is a R-deduction according to definition 3.11. By formalizing clauses as disjunctions and by separating substitution from contraction we get

$$\Gamma' = P(x) \vee P(a) \vee R(x), P(a) \vee P(a) \vee R(a), P(a) \vee R(a), \neg P(y) \vee Q(y), R(a) \vee Q(a).$$

Note that we did not instantiate $\neg P(y) \vee Q(y)$ to $\neg P(a) \vee Q(a)$ but preserved the usual resolution rule (the instances of clauses under m.g.u.'s for binary resolution are not added to the sequence).

- Γ' can directly be translated into the following ground deduction Γ'' for
- $\Gamma'' = P(a) \vee R(a), P(a) \vee P(a) \vee R(a), P(a) \vee R(a), \neg P(a) \vee Q(a), R(a) \vee Q(a)$.
- $\Gamma'' = \Gamma'\eta$ for $\eta = \{x \leftarrow a, y \leftarrow a\}$.

We call Γ'' a ground projection of Γ'.

Definition 7.1 Let $\Pi = C_1, \ldots, C_n$ be a R-deduction from $\mathcal{C}$. Π' is called a ground projection of Π if it holds:

1) Π' is a R-deduction from a set of ground instances $\mathcal{C}'$ of $\mathcal{C}$.

2) There exists ground substitutions $\lambda_1, \ldots, \lambda_n$ s.t. $\Pi' = C_1\lambda_1, \ldots, C_n\lambda_n$.

Ground projections are, in some sense, inverse to lifted refutations. As every ground R-deduction can be lifted to a general R-deduction, we may ask whether the other direction holds too, i.e. whether every R-deduction possesses a ground projection. But this is not the case:

Example 7.2

$$\mathcal{C} = \{\neg P(x) \vee P(f(x)), P(a), \neg P(f^4(z))\}$$

We define the following R-refutation Π of $\mathcal{C}$ (variants of clauses required for resolution are added to the sequence explicitly)

$\Pi = \neg P(x) \vee P(f(x)), \neg P(y) \vee P(f(y)), \neg P(x) \vee P(f^2(x)), \neg P(y) \vee P(f^2(y)),$
$\neg P(x) \vee P(f^4(x)), P(a), P(f^4(a)), \neg P(f^4(z)), \square.$

The basic idea in the proof Π is iterated self-resolution of newly derived clauses.

Π has no ground projection, because for the 4th clause in Π there exists no adequate instance to get the 5th one; an exact proof of this fact is given in [BL92]. The following argumentation serves as proof sketch:
Start with

$$\neg P(a) \vee P(f(a)), \neg P(f(a)) \vee P(f^2(a)), \neg P(a) \vee P(f^2(a)).$$

At this point it is impossible to produce the clause $\neg P(f^2(a)) \vee P(f^4(a))$, as it is neither a ground instance from $\mathcal{C}$ nor derivable from the clauses listed in the sequence

so far. Thus the sequence cannot be extended to a ground projection. We see that there are R-deductions without ground projections. However there are resolution refinements admitting ground projections such as linear input deductions. Let $\mathcal{C}$ be an unsatisfiable set of clauses; we define the Herbrand complexity of $\mathcal{C}$ as

$$H(\mathcal{C}) = min\{|\mathcal{C}'| \quad |\mathcal{C}' \text{ is an unsatisfiable set of ground instances from } \mathcal{C}\}.$$

By $l(\Gamma)$ (length of deduction Γ) we denote the number of clauses occurring in Γ (multiple occurrences are counted).

Proposition 7.1 *Let $\mathcal{C}$ be an unsatisfiable set of clauses and Π be a ground R-refutation of an unsatisfiable set of ground instances $\mathcal{C}'$ (from $\mathcal{C}$). Then $HC(\mathcal{C}) \leq l(\Pi)$.*

Proof: Π can only be a refutation if the set of all clauses $\mathcal{D}$ appearing in Π is unsatisfiable. By definition of HC, $HC(\mathcal{C}) \leq |\mathcal{D}|$. $|\mathcal{D}| \leq l(\Pi)$ is evident by definition of l.

Corollary 7.2 *Let Π be a refutation of $\mathcal{C}$ which has a ground projection; then $HC(\mathcal{C}) \leq l(\Pi)$.*

Proof: For every ground projection Π' of Π we have $l(\Pi') = l(\Pi)$.

A consequence of the corollary is that Herbrand complexity is a lower bound to the length of linear input refutations. Because, as we already know, there are R-refutations without ground projection, we cannot conclude that Herbrand complexity provides a lower bound to all R-refutations; instead we get the following result:

Theorem 7.3 *There exists a sequence of clause sets $(\mathcal{C}_n)_{n\in N}$ s.t. $\mathcal{C}_n$ is refutable by R-refutations Γ_n with $l(\Gamma_n) = 2n + 5$, but $HC(\mathcal{C}_n) > 2^n$ (Herbrand complexity may be exponential relative to resolution proof complexity).*

Proof sketch (a detailed proof can be found in [BL92]:
Define

$$\mathcal{C}_n = \{P(a), \neg P(x) \vee P(f(x)), \neg P(f^{2^n}(a))\}$$

(compare $\mathcal{C}_n$ to $\mathcal{C}$ in example 7.2) and refutations Π_n as:

$$\begin{array}{rcl}
\Pi_n & = & \neg P(x) \vee P(f(x)), \quad \neg P(y) \vee P(f(y)), \quad \neg P(x) \vee P(f^2(x)), \\
 & & \vdots \qquad\qquad \vdots \qquad\qquad \vdots \\
 & & \neg P(x) \vee P(f^{2^i}(x)), \ \neg P(y) \vee P(f^{2^i}(y)), \ \neg P(x) \vee P(f^{2^{i+1}}(x)), \\
 & & \vdots \qquad\qquad \vdots \qquad\qquad \vdots \\
 & & \neg P(x) \vee P(f^{2^n}(x)), \ P(a), \ P(f^{2^n}(a)), \ \neg P(f^{2^n}(a)), \ \square\ .
\end{array}$$

Π_n are R-refutations and $l(\Pi_n) = 2n + 5$. The second part of the proof consists in showing that $HC(\mathcal{C}_n) > 2^n$; here one can prove that all ground instances $\neg P(f^k(a)) \vee P(f^{k+1}(a))$ for $k \leq 2^n - 1$ of the second clause are required to get an unsatisfiable set of ground clauses.

Corollary 7.4 *R-refutations may be exponentially shorter than ground R-refutations.*

Theorem 7.3 provides us with information about all refutational methods in clause logic which possess ground projections; such are Prawitz's method, Bibel's (original) connection method and Chang's V-resolution ([Pra69], [Bib82], [Cha72]). For all these methods, having Herbrand complexity as a lower bound to proof complexity, a consequence is that R-refutations may be exponentially shorter than representations of refutations in those systems. Note that Π_n in the proof of theorem 7.3 is a linear refutation. Because $\mathcal{C}_n$ is a sequence of Horn sets there exist also linear input refutations $\triangle_n$. but these $\triangle_n$ (as they possess ground projections) must be exponentially longer than Π_n. The deeper reason is that linear resolution is capable of lemmatization, while input resolution is not. The following theorem shows that more than exponential speed-up w.r.t. Herbrand complexity is impossible.

Theorem 7.5 *Let Γ be a R-refutation of $\mathcal{C}$. Then $HC(\mathcal{C}) \leq 2^{2l(\Gamma)}$.*

Proof: Transform Γ into a refutation $\triangle$ having a ground projection.

R.Statman has shown that Herbrand complexity may be nonelementary(!) w.r.t. the length of a shortest refutation in a full logical calculus (e.g. natural deduction or sequent calculus with cut). Because R-refutations maximally give an exponential gain versus Herbrand complexity, the length of the shortest R-refutation may also be nonelementarily greater than the length of a refutation in a full logical calculus.

Example 7.3 (Statman's example)

$$\mathcal{C}_n = ST \cup ID \cup \{\neg ab = a((T_n b)b)\}.$$

ab is an abbreviation for $f(a,b), f \in FS_2$ and association to the left is assumed. ST is a set of combinator equations and ID a set of equality axioms.

$$ST = \{Sxyz = (xz)(yz),\ Bxyz = x(yz),\ Cxyz = (xz)y,\ Ix = x,\ px = p(qx)\}$$

S,B,C,I are constant symbols defining the corresponding well-known combinators, the 5th clause in ST is an additional axiom. T_n in the definition of $\mathcal{C}_n$ is a metatheoretical abbreviation for terms defined as $T_1 = T,\ T_{k+1} = T_k T,\ T = (SB)((CB)I)$.

The set of equality axioms ID is defined as:

$$ID = \{x = x, \neg x = y \vee y = x, \neg x = y \vee \neg y \doteq z \vee x = z, \neg x = y \vee \neg u = v \vee xu = yv\}$$

Let $s(0) = 1, s(n+1) = 2^{s(n)}$ for all n; it is well known from recursion theory that s is not an elementary function. For the set $\mathcal{C}_n$ above Statman proved [St79] that $HC(\mathcal{C}_n) \geq s(n)/2$. From theorem 7.5 we conclude that the resolution complexity of $\mathcal{C}_n \geq cs(n-1)$ for some constant c (independent of n).

7.2 The Method of Function Introduction

The reason for the high complexity of resolution can be found in the weak means to express and use lemmas in resolution proofs. From a proof theoretical point of view it is the elimination of quantifiers and the (only) atomic cut rule of resolution which prevents the expression of short proofs. This problem was attacked by E. Eder [Ede92] by the introduction of two extension rules into the resolution calculus; by these extension rules it is possible to introduce new predicate– and function symbols which "encode" formulas built up by quantifiers and connectives. Eder's method is very strong and can be considered as a generalization of Tseitin's extended resolution for propositional logic. In a slightly different (i.e. more restrictive) way, new function symbols are applied in [BL92] to represent shifting of quantifiers within clause logic. Although more restricted than Eder's rules, the function introduction rule in [BL92] leads to nonelementary "speed-up" of resolution proofs; the rest of this chapter will be devoted to this function introduction rule. Introducing new symbols within a resolution calculus is somehow against the philosophy of resolution where terms must always be kept minimal. On the other hand, clinging to minimality prevents formulation of substantial lemmata and thus the finding of short proofs. But we must be careful to preserve the computational power of resolution and its relatively small search space (otherwise we may take LK or natural deduction at once). In this sense the function introduction rule, to be defined below, is computationally adequate as it is directly related to the syntax of the clause where it is applied and can be controlled by simple heuristics.

Example 7.4 (Baaz-Leitsch 1990)

$$\begin{array}{lcl} \mathcal{C}_n & = & \{C_1, C_2, C_3, C_4^n, C_5^n\} \text{ for} \\ C_1 & = & P(x, f(x), y) \vee Q(y, f(y), x), \\ C_2 & = & \neg P(u, v, w_1) \vee \neg P(v, z, w_2) \vee P(u, z, w_1), \\ C_3 & = & \neg Q(u, v, w_1) \vee \neg Q(v, z, w_2) \vee Q(u, z, w_1), \\ C_4 & = & \neg P(a, f^{2^n}(a), z), \\ C_5 & = & \neg Q(a, f^{2^n}(a), z). \end{array}$$

Every R-refutation of $\mathcal{C}_n$ is of exponential length (in n). Investigating the form of the R-refutations, we find that all resolutions performed on the P-literal in C_1 are stored in the Q-literal and vice versa. Giving preference to resolutions with the P-literals we get very (exponentially) long queues of Q-literals and vice versa ($\mathcal{C}_n$ is completely symmetric). This effect can be avoided by splitting the clause C_1; note that an ordinary split of $\mathcal{C}_n$ via C_1 is impossible because C_1 does not decompose into variable-disjoint subclauses. We need stronger means, in fact quantificational rules, to enforce a decomposition of C_1.
For this purpose consider C_1 as quantificational formula

$$(\forall x)(\forall y)(P(x, f(x), y) \vee Q(y, f(y), x)).$$

By the valid schema

$$(\forall x)(\forall y)(A(x,y) \vee B(x,y)) \rightarrow (\forall x)(\exists y)A(x,y) \vee (\exists x)(\forall y)B(x,y),$$

which can be derived by shifting quantifiers, we obtain the formula

$$F : (\forall x)(\exists y)P(x, f(x), y) \vee (\exists x)(\forall y)Q(y, f(y), x).$$

By skolemizing F we obtain the clause

$$C = \{P(x, f(x), g(x)), Q(y, f(y), c)\}$$

which is decomposed. C is not R-derivable from $\mathcal{C}_n$ because it contains new function- and constant symbols.
But $\mathcal{C}_n \sim_{sat} \mathcal{C}_n \cup \{C\}$.
By using C instead of C_1 we obtain a refutation of $\mathcal{C}_n$ having linear length (in n); because C is decomposed we may split $\mathcal{C}_n$ into $\mathcal{C}_n \cup \{L_1\}$ and $\mathcal{C}_n \cup \{L_2\}$ for $C = L_1 \vee L_2$ and treat both set of clauses in parallel.

There are many variants of function introduction, depending on the (quantifier shifting) theorem applied to a specific clause. For theoretical purposes we may restrict quantificational inference to the innermost quantifier:

Definition 7.2 Let $\mathcal{C}$ be a set of clauses and $C \in \mathcal{C}$. Suppose that

$$A \equiv (\forall \overline{x})(\forall y)(F_1 \vee F_2)$$

is a PL-form of C subjected to a minimization of the range of the $\forall$-quantifiers (F_1, F_2 may contain quantifiers). Then the (skolemized) clause form of

$$F(\mathcal{C}) \wedge (\forall \overline{x})((Qy)F_1 \vee (Q^d y)F_2) \text{ for } Q \in \{\forall, \exists\}$$

is called $1-F-$extension of $\mathcal{C}$.

Remark: If C decomposes then $A \equiv F_1 \vee F_2$ and $1-F-$extension is not applicable. $1-F-$extension is not a rule which applies to a clause only, but is global in the sense that newly introduced function symbols may not appear in $\mathcal{C}$. The rule would be incorrect if only the formula $(\forall \overline{x})((Qy)F_1 \vee (Q^d y)F_2)$ is skolemized without respect to the whole set of clauses.

By shifting k quantifiers at once we get $k-F-$extensions and by shifting quantifiers till the form becomes a disjunction we obtain $SF-$(splitting $F-$) extensions [BL92]. In example 7.4 a splitting F-extension was applied. The concepts of $k-F-$extensions and $1-F-$extensions are independent in the sense that generally it is impossible to derive a $k-F-$extension by iterating $1-F-$extensions. Suppose that Q in definition 7.2 is $\forall$; then the extension is of the form

$$\mathcal{C} \cup \{C_1 \vee C_2\{y \leftarrow f(x_1, \ldots, x_k)\}\}.$$

For $Q = \exists$ we obtain

$$\mathcal{C} \cup \{C_1\{y \leftarrow f(x_1, \ldots, x_k)\} \vee C_2\}.$$

While $y \in V(C_1) \cap V(C_2)$ (by minimization of the quantifiers), $y \notin V(C_1) \cap V(C_2\{y \leftarrow f(x_1, \ldots, x_k)\})$. We see that $1-F-$extension is some kind of variable decomposition step within a clause.

Because $1-F-$extensions change the term universe we cannot expect to preserve strong correctness (the models remain the same during inference). But we know that

$$\mathcal{C} \sim_{sat} \mathcal{C} \cup \{C\}$$

if C is obtained by $1-F-$extension, what guarantees refutational correctness. Note that, whenever skolemization is applied (e.g. in transforming a formula to clause form), we must be content with refutational correctness. Although $F-$extensions model simple quantificational rules only, their effect can be very strong:

Theorem 7.6 *There exists a sequence of clauses $\mathcal{C}_n = \mathcal{C} \cup \{\{\neg P_n\}\}$ (the P_i are atoms) s.t. it holds:*

1) $l(\Pi) \geq cs(n-1)$ for all R-refutations Π of $\mathcal{C}_n$
(for some constant c and $s(0) = 1$, $s(n+1) = 2^{s(n)}$).

2) If $1-F-$extensions are admitted then, for every n, there exists R-refutations $\triangle_n$ s.t. $l(\triangle_n) \leq 2^{dn}$.

Idea of Proof: (exact proof in [BL92])
Take a modified version of Statman's example (example 7.3), formulate a short refutation in a calculus with (unrestricted) cut rule and derive the skolemized cut formulas via $1-F$-extension in some appropriate coding. The expense is "only" exponential.

Theorem 7.6 show that function introduction can lead to a nonelementary speed-up w.r.t. ordinary resolution. The question remains whether function introduction can be of real computational value. Answers can already be given for function introduction rules applied as splitting- and as strong factorization technique. The effect of clause splitting by function introduction was carefully investigated in [Egl90], where some remarkable speed-up was obtained for some classical theorem proving examples.

Function introduction is rather a principle than just a rule, as it may be infinitely varied and can adapted to different purposes. In [Egl91], [Egl92] function introduction was used as strong factorization rule leading to considerably shorter proofs.

Example 7.5 (Egly 1992)
Let $\mathcal{C}_n = \{C_1, C_2, C_3^n\}$ for $n \geq 0$ a sequence of clause sets s.t.

$$\begin{aligned} C_1 &= P(x, f(x,y), h(x,y)) \vee P(y, f(y,x), g(x,y)) \\ C_2 &= \neg P(u,v,w_1) \vee \neg P(v,z,w_2) \vee P(u,z,w_1) \\ C_3 &= \neg P(a, f(*,*)^{2^n}, z) \end{aligned}$$

$f(*,*)^m$ is defined as:

$$\begin{aligned} f(*,*)^1 &= f(a,a) \\ f(*,*)^{m+1} &= f(f(*,*)^m, f(*,*)^m) \end{aligned}$$

for $m > 0$.

Every R-refutation of $\mathcal{C}_n$ has a length $> 2^n$ (the reason is similar to that of example 7.4). However we can get much shorter proofs by manipulating the clause C_1 (as quantified formula). For this purpose represent C_1 as

$$A \equiv (\forall x)(\forall y)(P(x, f(x,y), h(x,y)) \vee P(y, f(y,x), g(x,y)))$$

Now we "unify" the third places of the atoms by introducing an existential quantifier via the valid formula:

$$A \rightarrow (\forall x)(\forall y)(\exists z)(P(x, f(x,y), z) \vee P(y, f(y,x), z))$$

By adding the right hand side of the formula to $\mathcal{C}_n$ and by skolemizing we obtain

$$P(x, f(x,y), r(x,y)) \vee P(y, f(y,x), r(x,y)).$$

But this clause, contrary to C_1, can be factorized to $P(x, f(x,x), r(x,x))$. Adding this last clause to $\mathcal{C}_n$ a refutation of linear length (in n) can be obtained.

Function introduction can be considered as a computational tool to apply quantificational rules in clause logic, where quantifiers don't belong to the syntax. Although some more investigations are required for an efficient application of such rules within resolution deductions, a starting point for "macro"- inference in clause logic is given. Function introduction is a purely predicate logic principle and is of no significance to propositional logic (it simply does not exist there). While many methods to improve inference owe their existence to prototypes in propositional logic, function introduction is of genuine quantificational nature; it is weaker than Hilbert's ε- formalism [HB34], where quantifiers are coded by terms under preservation of logical equivalence, but is intuitively related. Its specific characteristic is the skolemization principle as inference rule (similar to one of the extension rules in [Ede92]) instead of a preprocessing only.

Bibliography

[Ack28] W.Ackermann: Über die Erfüllbarkeit gewisser Zählausdrücke, M.A. 100, pp 638-649.

[Bib82] W.Bibel: Automated Theorem Proving, Vieweg, Braunschweig 1982.

[BL92] M.Baaz, A.Leitsch: Complexity of Resolution Proofs and Function Introduction, Annals of Pure and Applied Logic 57(1992), pp 181-215.

[Boy71] R.S.Boyer: Locking: A Restriction of Resolution. The University of Texas at Austin, Ph.D.dissertation, 1971.

[BS28] P.Bernays, M.Schönfinkel: Zum Entscheidungsproblem der mathematischen Logik. Math. Ann.99 (1928), pp 342-372.

[Cha72] C.L. Chang: Theorem Proving with Variable-Constraint Resolution, Information Sciences 4, pp 217-231.

[Chu36] A.Church: A Note on the Entscheidungsproblem. Journal of Symbolic Logic 1, pp 40-44.

[CL73] C.L.Chang, R.C.Lee: Symbolic Logic and Automated Theorem Proving. Academic press 1973.

[DL84] L.Denenberg, H.R.Lewis; Logical Syntax and Computational Complexity, Lecture Notes in Math. 1104, pp 101-115.

[DP60] M.Davis, H.Putnam: A Computing Procedure for Quantification Theory, Journal of the ACM 7 No.3, 1960, pp 201 - 215.

[Ede85] E.Eder: Properties of Substitutions and Unifications, J.Symbolic Computation 1 (1985) pp 31–46.

[Ede92] E.Eder: Relative Complexities of First Order Calculi, Vieweg Wiesbaden 1992.

[Egl90] U.Egly: Problem Reduction Methods and Clause Splitting in Automated Theorem Proving, Master thesis, Technische Universität Wien 1990.

[Egl91] U.Egly: A Generalized Factorization Rule Based on the Introduction of Skolem Terms. Proc. Seventh Austrian Conference on Artificial Intelligence, Informatik-Fachberichte 287, Springer 1991.

[Egl92] U.Egly: Shortening Proofs by Quantifier Introduction. Proc. LPAR'92, LNAI. 624, pp 148–159.

[Fer91] Ch.Fermüller: Deciding Classes of Clause Sets by Resolution, PhD-Thesis, Technical University Vienna 1991.

[Fer91a] Ch.Fermüller: A Resolution Variant Deciding some Classes of Clause Sets. CSL'90, LNCS 533, pp 128-144 (1991).

[FLTZ92] Ch.Fermüller, A.Leitsch, T.Tammet, N.Zamov: Resolution Methods for the Decision Problem (Monograph).

[Gen34] G.Gentzen: Untersuchungen über das logische Schließen I-II, Math.Z.39, 176-210, 405-431.

[GF92] G.Gottlob, Ch.Fermüller: Removing Redundancy from a Clause, to appear in Artificial Intelligence.

[Gil60] P.C.Gilmore: A Proof Method for Quantification Theory, its Justification and Realization, IBM J.Res. Develop. pp 28-35.

[GJ79] M.R.Garey, D.S.Johnson: Computers and Intractability. Freeman, San Francisco 1979.

[GL85] G.Gottlob, A.Leitsch: On the Efficiency of Subsumption Algorithms, J.ACM 32 No.2 pp 280-295.

[Göd30] K.Gödel: Die Vollständigkeit der Axiome des logischen Funktionenkalküls, Mh.Math. Phys. 37 pp 349-360.

[Göd31] K.Gödel: Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme, Mh. Math. Phys. 38 pp 175-198.

[Göd32] K.Gödel: Ein Spezialfall des Entscheidungsproblems der theoretischen Logik, Ergebn. math. Kolloq. 2, pp 27-28.

[Gur73] Y.Gurevich: Formuly s odnim $\forall$ (formulas with one $\forall$). In Izbrannye voprosy algebry i logiki (Selected Questions in Algebra and Logics; in memory of A.Mal'cev). Nauka, Nowosibirsk, 1973,pp 97-110.

[HB34] D.Hilbert, P.Bernays: Grundlagen der Mathematik II, Springer 1970.

[Her30] J.Herbrand: Recherches sur la theorie de la demonstration, Travaux de la Societe des Sciences et des Lettres de Varsovie no.33.

[Joy73] W.H.Joyner: Automated Theorem Proving and the Decision Problem. Ph.D.Thesis Harvard University, 1973.

[Joy76] W.H.Joyner: Resolution Strategies as Decision Procedures. J.ACM 23, 1976, pp 398-417.

[KH69] R.Kowalski, P.J.Hayes: Semantic Trees in Automated Theorem Proving, in Machine Intelligence 4, 1969, pp 87–101.

[Lee67] R.C.T.Lee: A Completeness Theorem and a Computer Program for Finding Theorems Derivable from Given Axioms, Ph.D.Thesis, Univ. of California at Berkley 1967.

[Lei89] A.Leitsch: On Different Concepts of Resolution, Zeitschr. für Math.Logik und Grundlagen der Mathematik 35, 1989, pp 71-77.

[Lei90] A.Leitsch: Deciding Horn Classes by Hyperresolution, CSL'89, LNCS 440, pp 225-241.

[Lei92] A.Leitsch: Deciding Clause Classes by Semantic Clash Resolution, to appear in Fundamenta Informaticae 1992.

[Lov78] D.Loveland: Automated Theorem Proving - A Logical Basis, North Holland Publ. Comp.1978.

[Löw15] L.Löwenheim: Über Möglichkeiten im Relativkalkül, Mathm.Ann.68, pp 169-207.

[Mas64] S.Y.Maslov: An Inverse Method of Establishing Deducibilities in the Classical Predicate Calculus, Dokl.Akad.Nauk SSSR 159, 1420-1424.

[MM82] A.Martelli, U.Montanari: An Efficient Unification Algorithm, ACM Transactions on Programming Languages and Systems Vol. 4, No2 (1982).

[MP92] J. Marcinkowski, L.Pacholski: Undecidability of the Horn-Clause Implication Problem, Groupe de recherche algorithmique & logique, rapport de recherche 1992-5, University of Caen.

[Pra69] D.Prawitz: Advances and Problems in Mechanical Proof Procedures, Machine Intelligence 4, American Elsevier, N.Y 1969.

[Rob65] J.A.Robinson: A Machine Oriented Logic Based on the Resolution Principle, Journal of the ACM 12, pp 23 -41.

[Rob65a] J.A.Robinson: Automated Deduction with Hyperresolution, Intern. Journal of Computer Mathematics 1, pp 227-334.

[Rud91] V.Rudenko: Decision of a Horn Clause Implication Class with ≤ 2 Variables in the Head, unpublished manuscript 1991.

[Sla67] J.R.Slage: Automatic Theorem Proving with Renable and Semantic Resolution, Journal of the ACM 14 No.4, pp 687 - 697.

[Sch88] M.Schmidt-Schauss: Implication of Clauses is Undecidable, Theoretical Computer Science 59 (1988), pp 287-296.

[St79] R.Statman: Lower Bounds on Herbrand's Theorem, Proc. AMS 75 (1979) pp 104-107.

[Sti73] R.B. Stillman: The Concept of Weak Substitution in Theorem-Proving. J.ACM 20, pp 648-667.