Resolution in FOL

Ralitsa Dardjonova

June 2020

Abstract

In this paper we consider resolution as a procedure for deciding unsatisfiability of a formula. We start with presentation of resolution in propositional logic and we show it is a valid calculus which always terminates. We extend the resolution technique to predicate logic. We again show it is sound and complete, although it might not terminate for some formulas. We discus that resolution proofs may require exponential time and we consider some strategies and refinements for improving the time complexity.

1 Introduction

Resolution is a method that is applied to formulas and decides their unsatisfiability. First it was introduced by Robinson in 1965 and now it is extensively used in Automated Reasoning. Other algorithms for deciding satisfiability of a program existed before that.

One such algorithm is DPLL, which was introduced in 1962 by Martin Davis, George Logemann and Donald W. Loveland [2]. DPLL is a backtracking algorithm that decides satisfiability of propositional formulas.

DPLL tries to find a suitable assignment where the formula is true. This means the unsatisfiability of a formula might result in an exhaustive search over the possible assignments, which corresponds to exponential time complexity. In this paper, we describe the resolution calculus that checks for unsatisfiability of a formula. This makes it suitable for high range of problems that previously required exponential time.

For example, consider the formula F:

$$F = (A \lor B \lor \neg C) \land (A \lor B \lor C) \land (A \lor \neg B) \land (\neg A)$$

Using DPLL, it will require exponential time(2^3 possible assignments to the variables). Later we will see that using resolution it will require only linear steps to show that F is unsatisfiable.

Another advantage of resolution is that it can be used in predicate logic, whereas DPLL is only suited for propositional formulas. In addition, resolution can be refined, in order to improve the time complexity. Nowadays, most of the SAT solvers consists of resolution in some form.

Since resolution checks for unsatisfiability of a formula, we can use it to prove theorems (tautologies) by applying the resolution method to the negation of the formula. Thus, resolution is frequently used in theorem solvers and other similar programs.

Another important thing is that resolution is a syntactic technique. This means we don't need the semantics of a formula to apply resolution. We only apply one simple rule to the formula and then use the result. The inference rule can easily be reasoned upon and proven correct. This results in a straightforward algorithm, which is easy to be programmed.

Before using resolution we need to show that it is a valid calculus. This means we need to prove its correctness (soundness) and completeness. Correctness says that when the resolution algorithm, applied to a formula F, produces the empty set, then the formula must be unsatisfiable. Completeness, on the other hand, shows that if a formula F is unsatisfiable, then the resolution method applied to F results in an empty set.

In this paper, we are going to consider resolution in propositional logic. Later we will show resolution can be extended to predicate logic and we are going to present algorithm. We will also prove correctness and completeness of those methods and comment on their efficiency.

2 Propositional Logic

To apply resolution, the formulas need to be in special from. First, we will begin with the definition of literal.

Definition 2.1. *Formula L is a literal, if it is an atomic formula or its negation.*

Definition 2.2. *Dual to the literal L is such literal, which is equivalent to $\neg L$. It is denoted by $\overline{L}$.*

$$\overline{L} := \begin{cases} \neg A, & if \;\; L = A \\ L, & if \;\; L = \neg A \end{cases},$$

where A is an atomic formula.

We need a definition of dual literal, because $\neg L$ might not be a literal. For example $\neg\neg L$ is not a literal.
It can be shown that for every formula there exists equivalent to it formula, which can be represented by a conjunction of disjunctions of literals. Such representation is called Conjunctive Normal Form (**CNF**).

Definition 2.3. *Formula F is in Conjunctive Normal Form (***CNF***) if it has the form:*

$$F = (L_{1,1} \vee L_{1,2} \vee ... \vee L_{1,n_1}) \wedge ... \wedge (L_{k,1} \vee ... \vee L_{k,n_k}),$$

where $\{L_{i,n_j}\}$ are literals.

Definition 2.4. *A clause is a finite set of literals. Disjunctive clause is true when at least one of the literals in it is true. Here we will refer to it as just a clause.*

Every formula F in **CNF** can be represented by a set of clauses. We need to transform every formula in such form to apply resolution. The equivalent representation of the aforementioned formula F is:

$$F = \{\{L_{1,1}, L_{1,2}, ..., L_{1,n_1}\}, ..., \{L_{k,1}, ..., L_{k,n_k}\}\}.$$

Definition 2.5. *Two formulas F and G are said to be equivalent if they have the same truth value for all possible assignments to the variables in the formulas. We write $F \equiv G$.*

Different formulas can have the same representation using clauses, since we are using sets. For example the disjunction $(A \vee A \vee B)$ corresponds to the same clause as $(A \vee B)$, namely $\{A, B\}$. We can see that if two formulas correspond to the same set of clauses, then they must be equivalent.

Sometimes however we don't need to know if two formulas are exactly equivalent. We can use a slightly weaker rule.

Definition 2.6. *Two formulas F and G are said to be equisatisfiable, when F is satisfiable iff G is satisfiable.*

Since we have shown what clauses are, we can present the resolution inference rule:

Definition 2.7. *Lets have $C_1 = A \cup \{L\}$ and $C_2 = B \cup \{\overline{L}\}$ clauses. Then the resolution inference rule is:*

$$\frac{A \cup \{L\} \quad B \cup \{\overline{L}\}}{A \cup B}$$

We say the clause $R = A \cup B$ is a resolvent of the clauses C_1 and C_2.

The next lemma shows why such inference rule is reasonable.

Lemma 2.8 (Resolution lemma)**.** *Let F be a formula in* **CNF***, represented as a set of clauses, and let R be a resolvent of two clauses C_1 and C_2 in F. Then the formulas F and $F \cup \{R\}$ are equivalent.*

Proof. Let $\mathcal{A}$ be a suitable assignment for F and $F \cup R$. Then :

- If $F \cup R$ is true in $\mathcal{A}$, then F is also true in $\mathcal{A}$.
- If F is true in $\mathcal{A}$. Then all clauses $C \in F$ must be true in $\mathcal{A}$. Assume the resolvent R has the form $R = (C_1 - \{L\}) \cup (C_2 - \{\overline{L}\})$ where $L \in C_1, \overline{L} \in C_2$. If L is true, then there is a literal in C_2, which must be true and from there R is true in $\mathcal{A}$. The same follows for the case if $\overline{L}$ is true.

■

This lemma tells us that we can apply the resolution inference rule to a clause set F and check if the new set is unsatisfiable. If we find that the empty clause (empty set) is in the new set, we will know that F is unsatisfiable. If we have that the empty clause is in the set, then it needs to be a resolvent of two clauses or to be originally from F. The next definition formalizes this reasoning.

Definition 2.9. *Resolution derivation (proof) of a clause C from a clause set F is a sequence of clauses $C_1, C_2, ..., C_m$, where $C_m = C$ and every clause C_i is either:*

- *a clause in F*
- *a resolvent of two clauses C_a, C_b with $a, b < i$*

Then we write $F \vdash_{Res} C$.

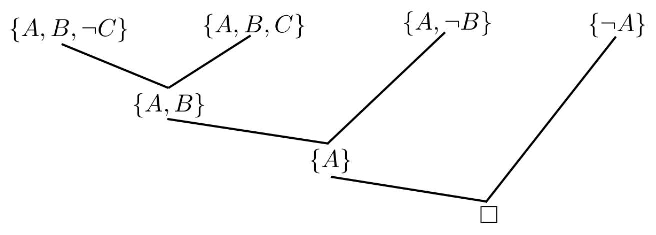
Then if we have $F \vdash_{Res} \square$, we will know that F is unsatisfiable.

Example 2.10. *Consider the formula F from the introduction:*

$$F = (A \vee B \vee \neg C) \wedge (A \vee B \vee C) \wedge (A \vee \neg B) \wedge (\neg A)$$

.

We can represent a resolution derivation as a directed acyclic graph:



We can also express the resolution derivation as a sequence of resolution steps:

1. $\{A, B, \neg C\}$
2. $\{A, B, C\}$
3. $\{A, \neg B\}$
4. $\{\neg A\}$
5. $\{A, B\}$ *// from 1 and 2*
6. $\{A\}$ *// from 3 and 5*
7. $\Box$ *// from 4 and 6*

where $\Box$ is the empty clause.

Even for such a small example, we see that we need fewer steps than an exhaustive search over all possible assignments.
We now proceed with the proof of correctness of the resolution calculus. For this we will need the next theorem in order to deal with infinite sets of clauses.

Theorem 2.11 (Compactness theorem)**.** *A set M of formulas is satisfiable iff every finite subset of M is satisfiable.*

Proof. In every assignment $\mathcal{A}$, where M is true, every subset of M is also true. This applies, in particular, to every finite subset of M. Therefore the direction from right to left is immediate.
The proof of the other direction is presented in detail in Section (1.4) in [3]. ∎

From this theorem follows a very important fact: A set of formulas M is unsatisfiable iff there exists a finite subset of M that is unsatisfiable.

Theorem 2.12. *Let F be a set of clauses. If $F \vdash_{Res} C$ then C follows semantically from F (if we have F is true in some assignment of the variables, then C is also true in this assignment).*

Proof. Lets have the resolution derivation $C_1, C_2, ..., C_n = C$. We can prove this theorem by induction on i. Then for every C_i we have:

- $C_i \in F$ and then trivially C_i follows from F;
- C_i is a resolvent of C_a and C_b, where $a, b < i$. Then by the Resolution lemma we have that C_i follows from F.

∎

Then if C is the empty clause, we have $\Box$ follows from F, which means F is unsatisfaiable. We get the corollary:

Corollary 2.12.1 (Correctness of resolution)**.** *Let F be set of clauses. If $F \vdash_{Res} \Box$ then F is unsatisfiable.*

Theorem 2.13 (Completeness of resolution)**.** *Let F be a set of clauses. If F is unsatisfiable then $F \vdash_{Res} \Box$.*

Proof. If F is infinite, there must be a finite unsatisfiable subset of F (by the Compactness theorem). Then we can assume F is finite. We can prove $F \vdash_{Res} \Box$ by an induction on the number of atoms in F. Detailed proof can be seen in Section (1.5) in [3] ∎

From Corollary (2.12.1) and Theorem (2.13) we can conclude:

Corollary 2.13.1. *Let F be a set of clauses. F is unsatisfiable iff $F \vdash_{Res} \Box$.*

Now that we know the resolution calculus is valid, we can create an algorithm for checking unsatisfiability.
The process is as follows:

```
1. Convert formula $F$ to **CNF**
2. Create the set of clauses $G$ corresponding to $F$
3. while (there are clauses $C_a, C_b \in G$ and resolvent R of $C_a$ and $C_b$ such that $R \notin G$)
   do $G = G \cup \{R\}$
       if $R$ is $\Box$ then break
```

Then F is unsatisfiable iff $\square$ is in G after execution.

For some formulas, resolution was proven to be fast and efficient. However, for an arbitrary clause set, the derivation of the empty clause might contain an exponential number of steps. It was shown in 1984 by Haken [5] that for infinitely many formulas the shortest resolution proof cannot be bounded by any polynomial w.r.t the length of these formulas. Later is was proved there is an exponential lower bound on the length [6]. To achieve this the famous pigeonhole problem was used. The idea is as follows: we have n pigeons and m holes, where $n > m$. We want to put every pigeon in one hole, and each hole to contain only one pigeon. This problem can be expressed as a formula in **CNF**:

$$P_{i,1} \vee P_{i,2} \vee ... \vee P_{i,m}, \text{ for each } i \leq n; \text{ and}$$

$$\neg P_{i,k} \vee \neg P_{j,k}, \text{ for each } i, j \leq n, k \leq m, i \neq j.$$

Tseitin [7] introduced the so-called extended resolution and it is further discussed in [5] by Haken. For the case $n = m + 1$, extended resolution yields proofs with complexity n^4.

Another important property of propositional resolution is that it always terminates. There are finitely many clauses that can be constructed from a finite set of proposition constants. Then the procedure eventually runs out of possible resolution steps and the algorithm halts.

Now we are able to extend resolution to predicate logic.

3 Predicate logic

We saw how resolution works in propositional logic and now we want to extend it to predicate logic. For this reason, we need to know what to do with variables, functional symbols and quantifiers. We need to define semantics for predicate logic.

Definition 3.1 (Structure)**.** *A structure is a pair $\mathcal{A} = (U_\mathcal{A}, I_\mathcal{A})$. $U_\mathcal{A}$ is a non-empty set called universe(ground set). $I_\mathcal{A}$ is a mapping that maps:*

- *each k-ary predicate symbol P to a k-ary predicate on $U_\mathcal{A}$*
- *each k-ary function symbol f to a k-ary function on $U_\mathcal{A}$*
- *each variable x to an element of $U_\mathcal{A}$*

We denote the value of term as $\mathcal{A}(t)$. Similarly, we define inductively the value of a formula F, denoted by $\mathcal{A}(F)$. We are going to use the standard definition of semantics in predicate logic, which is thoroughly described in Section (2.1) in [3].

Definition 3.2 (Model)**.** *Let $\mathcal{A}$ be a suitable structure for F. We say $\mathcal{A}$ is a model for F if $\mathcal{A}(F) = 1$. We write $(A) \models F$ (F is true in $\mathcal{A}$).*

If a formula F is true in every suitable $\mathcal{A}$, we say F is valid and we write $\models F$. If there is at least one model for a formula F, then F is satisfiable, otherwise unsatisfiable.

Definition 3.3 (Equivalence)**.** *We say two formulas F and G are equivalent, if for all suitable structures $\mathcal{A}$ for both F and G, we have $\mathcal{A}(F) = \mathcal{A}(F)$. We write $F \equiv G$.*

Definition 3.4 (Substitution)**.** *Let F be a formula, x a variable, and t a term. Then by $F[x/t]$ we denote a formula, where every free occurrence of x is replaced by t.*

We can look at substitutions [x/t] as functions and compose them. For example $\sigma_1\sigma_2$ first applies the substitution σ_1 and then σ_2

Lemma 3.5. *Let $F = QxG$ be a formula and Q be a quantifier, $Q \in \{\exists, \forall\}$. If y is a variable that does not occur free in F, then $F \equiv QyG[x/y]$.*

Before applying resolution we need to preprocess our formulas.

Definition 3.6 (Rectified form)**.** *We say that a formula F is in rectified form, if:*

- *no variable occurs both bound and free, and*
- *all quantifiers in the formula refer to different variables.*

Then using Lemma (3.5) and Definition (3.6) we can show the next theorem.

Theorem 3.7. *For every formula F exists a rectified formula G, where $F \equiv G$.*

Example 3.8. *Consider the formula:*

$$F = \forall x \exists y P(x, y) \wedge \forall y \exists x Q(y, x).$$

We have that G_1 is in rectified form and it is equivalent to F:

$$G_1 = \forall u \exists v P(u, v) \wedge \forall y \exists x Q(y, x).$$

Now we are able to transform our formula to the next form.

Definition 3.9 (Prenex form)**.** *A formula F is in prenex form if it is in the form:*

$$F = Q_1x_1Q_2x_2...Q_nx_nG,$$

where $Q_i \in \forall, \exists$ and x_i are variables, and G doesn't contain quantifiers.

This just means that we put all quantifiers to the front of the formula. When we have our formula in rectified form, this is easily done, because we do not have bindings using the same variable name. Using Lemma (3.5) and Definition (3.8) we can show the next theorem.

Theorem 3.10. *For every formula F exists a rectified formula G in prenex form, and $F \equiv G$.*

Proof. The proof of theorem is shown in Section (2.2) in [3]. ■

Example 3.11. *We consider again the same formula F:*

$$F = \forall x\exists yP(x, y) \wedge \forall y\exists xQ(y, x).$$

We can use the formula G_1 in rectified form which we have found earlier. Then we can find the formula G_2 in rectified prenex form, for which $F \equiv G_2$. We can do it by moving quantifiers to the front:

$$G_2 = \forall u\exists v\forall y\exists x(P(u, v) \wedge Q(y, x)).$$

We say a formula F is in **RPF** if it is in rectified prenex form.

Now that we have a formula in **RPF** we can apply the next procedure.

Definition 3.12 (Skolem form)**.** *Reduction of a formula F to Skolem form is a method for removing existential quantifiers from a formula and replacing its bounded variables with a functional symbol. This means, for every variable y which is bounded to an existential quantifier and is preceded by universal quantifiers:*

$$F = \forall x_1\forall x_2...\forall x_n\exists yG,$$

F is transformed to:

$$F = \forall x_1\forall x_2...\forall x_nG[y/f(y_1, ..., y_n)],$$

where f is a functional symbol which does not occur in F.

Example 3.13. *We consider again the same formula F:*

$$F = \forall x\exists yP(x, y) \wedge \exists xQ(y, x).$$

. *We have found $G_2 \equiv F$, where G_2 is in* **RPF***. Then we can reduce G_2 to Skolem form:*

$$G_2 = \forall u\forall y(P(u, f(u)) \wedge Q(y, g(u, y)).$$

Skolemization does not preserve equivalence, but preserves satisfiability. However, our goal is to check if a formula F is unsatisfiable, so we can use the formula in Skolem from to achieve this.

Theorem 3.14. *For each formula F in* **RPF***, F is satisfiable iff the Skolem form of F is satisfiable.*

For lifting resolution to predicate level we need special type of structures.

Definition 3.15 (Herbrand universe)**.** *Herbrand universe is the set of all variable free terms, which can be constructed from function symbols and constants occurring in F. If there is no constant in F, we add an arbitrary constant a, which is used to build the elements of the Herbrand universe. We write it as D(F).*

Example 3.16. *Lets have the formula $F = \forall x\forall y.p(a, f(x, y))$. Then the Herbrand universe is:*

$$D(F) = \{a, f(a, a), f(f(a, a), a), f(a, f(a, a)), ...\}$$

Definition 3.17 (Herbrand structure)**.** *For a formula F, we say a structure $\mathcal{A} = (U_\mathcal{A}, I_\mathcal{A})$ is Herbrand if:*

- *$U_\mathcal{A}$ is the Herbrand universe of F*
- *$c^\mathcal{A} = c$*
- *$f^\mathcal{A}(t_1, ..., t_k) = f(t_1, ..., t_k)$*

Example 3.18. *for the formula F from the previous example, we have the following Herbrand structure $\mathcal{H}$:*

$$a^\mathcal{H} = a$$

$$f^\mathcal{H}(f(a, a), a) = f(f(a, a), a)$$

$$etc...$$

This means the semantic of the Herbrand structures corresponds to the syntax. Although functional symbols are fixed, we can have a different definition for the predicate symbols. Every functional symbol can be just considered as a literal in propositional logic. This way we are able to apply resolution as considered in the previous chapter.

Theorem 3.19. *Let F be a closed formula in Skolem form. Then F is satisfiable iff F has a Herbrand model.*

Proof. It is obvious that if F has an Herbrand model, then it is satisfiable.
Conversely, if F is satisfiable, let $\mathcal{A}$ be an arbitrary model for F. We define a Herbrand structure $\mathcal{B}$. We define the predicate symbols in $\mathcal{A}$ to imitate the definitions of the predicates in $\mathcal{A}$(we get the values of the terms in a tuple in $\mathcal{A}$ and check if the transformed tuple satisfies the predicates in $\mathcal{A}$). Then $\mathcal{B}$ is a Herbrand model of F. ■

Using this theorem we need only to find a Herbrand model for F, in order to show that F is satisfiable. We can simplify our task even more. For this we will need the next special set, but first we will recall what is a ground instance.

Definition 3.20. *A ground term/formula/instance/etc. is a term/formula/instance/etc. which doesn't contain any variables.*

Definition 3.21 (Herbrand expansion)**.** *Let $F = \forall x_1 \forall x_2 ... \forall x_n G$ be a closed formula in Skolem form. Then the Herbrand expansion denoted as E(F) is:*

$$E(F) := \{G[x_1/t_1][x_2/t_2]...[x_n/t_n] | t_1, t_2, ..., t_n \in D(F)\}.$$

The Herbrand expansion is just the set of all ground instances of a formula F.

Example 3.22. *If we consider the formula $F = \forall x \forall y . p(a, f(x, y))$ from the previous example. Then*

$$E(F) = \{p(a, f(a, a)), p(a, f(f(a, a), a)), p(a, f(f(a, a), f(a, a)), ...\}.$$

The set $E(F)$ has no free variables and it is like a set of propositional formulas. The next theorem makes the connection between satisfiability of formulas in predicate and propositional logic. We can approximate a formula in predicate logic using propositional logic. Then we will be able to use resolution in proposition logic in order to prove unsatisfiability or satisfiability.

Theorem 3.23. *Let F be a closed formula in Skolem form. Then F is satisfiable iff the set of formulas E(F) is satisfiable.*

Proof. Using Theorem (3.14) it suffices to show that F has a Herbrand model iff E (F) is satisfiable.
Let F be in the form $F = \forall x_1 ... \forall x_n G$. Then:
$\mathcal{A}$ is a Herbrand model
iff for all $t_1, ..., t_n \in D(F),\ \mathcal{A}_{[x_1/t_1]...[x_n/t_n]}(G) = 1$
iff for all $t_1, ..., t_n \in D(F),\ \mathcal{A}(G[x_1/t_1]...[x_n/t_n]) = 1$
iff for all $F' \in E(F),\ \mathcal{A}(F') = 1$
iff $\mathcal{A}$ is a model for $E(F)$. ■

Using the above theorem we arrive at the next one.

Theorem 3.24 (Herbrand's theorem)**.** *A closed formula F in Skolem form is unsatisfiable iff there is a finite subset of E(F) which is unsatisfiable.*

Proof. The proof is a direct combination of the previous theorem and the Compactness theorem in propositional logic. ■

Using the above theorem we can construct an algorithm that checks for unsatisfiability of a formula in predicate logic. It uses the Herbrand expansion and in every pass of the loop we add new ground instance of the formula to the conjunction.
Gilmore′s Algorithm:

```
1. Transform $F$ to a closed formula in Skolem form
2. Compute sequence $\{F_1, F_2, F_3, ...,\}$, which is an enumeration of $E(F)$
3. n := 0
   until $(F_1 \wedge F_2 \wedge ... \wedge F_n)$ is "unsatisfiable"
   do $n := n + 1$
   return "unsatisfiable" and halt
```

We can see that if a formula F is satisfiable, the Gilmore's Algorithm is not going to halt, because the loop condition would always be true. Another issue is that we have not stated how we will check the unsatisfiability of $(F_1 \wedge F_2 \wedge ... \wedge F_n)$. This means we need to find a way to improve this algorithm. One of the things we can do is to use resolution for checking the unsatisfiability. However, in propositional resolution we work with clauses. Then we need to simplify our predicate formula to a set of ground instances of clauses (not of the whole formula).

Definition 3.25 (Universal closure)**.** *The universal closure of a formula H with free variables $x_1, ..., x_n$ is the formula:*

$$\forall H := \forall x_1, ..., \forall x_n H$$

This means that if F is a closed formula in Skolem form, then:

$$F \equiv \forall G \equiv \bigwedge_{C \in G} \forall C$$

Definition 3.26 (clause Herbrand expansion)**.** *Let $F = \forall x_1 \forall x_2 ... \forall x_n G$ be a closed formula in Skolem form and G is in* **CNF***. Let $C_1, C_2, ..., C_m$ are the clauses of G. The clause Herbrand expansion of F is the set of ground clauses:*

$$CE(F) := \bigcup_{i=1}^{m} E(\forall C_i)$$

Then next theorem makes the connection between clause Herbrand expansion and Herbrand expansion, and from there to the original formula F.

Lemma 3.27. *CE(F) is unsatisfiable iff E(F) is unsatisfiable.*

Proof. Follows immediately from the definition of satisfiability for sets of formulas. ■

Using the previous lemma we can conclude F is unsatisfiable iff $CE(F)$ is unsatisfiable. Then we can construct the following algorithm, which is an improvement of the Gilmore's one.
Ground resolution algorithm:

```
1. Transform $F$ to a closed formula in Skolem form with $G$ in CNF.
2. Compute sequence $C_1, C_2, C_3,...$, which is an enumeration of $CE(F)$
3. $n := 0$
   $S := \emptyset$
   do $n := n + 1$
       $S := S \cup \{C_n\}$
   until $S \vdash_{Res} \square$
   return "unsatisfiable"
```

Again, we have reached a point where we reason upon clauses of a formula. Then we can use similar approach to resolution derivation in order to prove the correctness of the ground resolution algorithm.

Theorem 3.28 (Ground resolution theorem)**.** *A formula $F = \forall x_1 ... \forall x_n G$ with G in* **CNF** *is unsatisfiable iff there is a sequence of ground clauses $C_1, ..., C_m = \square$ such that for every i = 1,...,m:*

- *either C_i is a ground instance of a clause $C \in G, i.e. C_i = C[x_1/t_1]...[x_n/t_n]$ where $t_1, ..., t_n \in D(F)$,*
- *or C_i is a resolvent of two clauses C_a, C_b with $a < i$ and $b < i$*

Although this algorithm might seem nice, it has several issues. Here we compute all ground instances of clauses in F. This task is expensive and most of the time we will need only a small part of them. Then if we have infinite number of ground instances, we might never conclude that a formula is satisfiable. However we can use this algorithm later to make the connection between resolution in predicate logic and the resolution in propositional one. To define resolution in predicate logic we need a way to reason about predicates. We will start with some definitions.

Definition 3.29. *Two substitutions σ_1 and σ_2 are equivalent, if for every term t we have $t\sigma_1 = t\sigma_2$.*

We can swap substitutions $[x/t]$ and σ only if x does not occur in σ. Then $[x/t]\sigma = \sigma[x/t]$.

Definition 3.30 (Unifier)**.** *Let $L = \{L_1, ..., L_k\}$ be a set of literals of predicate clauses. A substitution σ is an unifier of L if:*

$$L_1\sigma = L_2\sigma = ... = L_k\sigma.$$

This is equivalent to saying $|L\sigma| = 1$.

Definition 3.31 (Most general unifier)**.** *A unifier σ of L is a most general unifier of L if for every unifier σ' of L there is a substitution s such that $\sigma' = \sigma s$.*

Theorem 3.32 (Unification)**.** *Every unifiable set of literals has a most general unifier.*

Proof. The proof of this theorem is the Unification algorithm. This algorithm checks if a set of literals has an unifier, in which case it returns the most general unifier. The main idea is as follows: While $L\sigma$ has more than 1 element, we find the first position at which two literals differ.

- if none of the characters is a variable, then we return "non-unifiable";
- if at least one of them is a variable x, then we substitute it with the other term t, if x does not appear in t.

Detailed proof can be seen in Section (2.5) in [3] ■

Now we are able to define the inference resolution rule for predicate logic.

Definition 3.33 (Resolution in predicate logic)**.** *Let C_1, C_2 and R be clauses (in predicate logic). Then R is called a resolvent of C_1, C_2 if the following holds:*

- *There exist certain substitutions s_1 and s_2 which are variable renamings so that $C_1 s_1$ and $C_2 s_2$ do not contain the same variable.*
- *If $C_1 s_1 = A \cup \{L_1, ..., L_m\}$ and $C_2 s_2 = B \cup \{L'_1, ..., L'_n\}$, and the set:*
$$L = \{L_1, ..., L_m, \overline{L'_1}, ..., \overline{L'_n}\}$$
is unifiable. Let σ be the most general unifier of L.
- *$R = (A \cup B)\sigma$.*

We have the inference rule:
$$\frac{A \cup \{L_1, ..., L_m\} \quad B \cup \{L'_1, ..., L'_n\}}{(A \cup B)\sigma}$$

Example 3.34. *Consider the following resolution step:*

$$\begin{tikzcd}
\{P(f(x)), \neg Q(z),\ P(z)\} \arrow[dr, dash, "s_1 = {[]}"'] & & \{\neg P(x), R(g(x))\} \arrow[dl, dash, "s_2 = {[x/u]}"] \\
& \sigma = [z/f(x)][u/f(x)] \arrow[d, dash] & \\
& \{\neg Q(f(x)), R(g(f(x)))\} &
\end{tikzcd}$$

The example is taken from [3].

Resolution calculus in propositional logic can be seen as a special case of resolution in predicate logic. There we have $s_1 = s_2 = \sigma = []$ and $n = m = 1$.

To prove the correctness and completeness of resolution in predicate logic, we need the next lifting lemma. It shows how some resolutions in propositional logic can be lifted to resolutions in predicate logic.

Lemma 3.35 (Lifting lemma)**.** *Let C_1, C_2 be predicate clauses and let C'_1, C'_2 be two ground instances of them that can be resolved into the resolvent R'. Then there is a predicate resolvent R of C_1, C_2 such that R' is a ground instance of R.*

Assumption of the lifting lemma

$$\begin{tikzcd}
C_1 \arrow[d] & & C_2 \arrow[d] \\
C'_1 \arrow[dr, dash] & & C'_2 \arrow[dl, dash] \\
& R' &
\end{tikzcd}$$

Conclusion of the lifting lemma

$$\begin{tikzcd}
C_1 \arrow[dr, dash] & & C_2 \arrow[dl, dash] \\
& R \arrow[d] & \\
& R' &
\end{tikzcd}$$

These schemes are taken from [3].

Proof. Since C'_1 and C'_2 are ground instances of C_1 and C_2, they are also ground instances of $C_1 s_1$ and $C_2 s_2$. Let σ_1 and σ_2 be ground substitutions such that $C'_1 = C_1 s_1 \sigma_1$ and $C'_2 = C_2 s_2 \sigma_2$. Then we set $\sigma = \sigma_1 \sigma_2$ and we get $C'_1 = C_1 s_1 \sigma$ and $C'_2 = C_2 s_2 \sigma$. By assumption there is a resolvent R' of C_1 and C_2 in propositional logic. Then there is a literal $L \in C'_1$ and $\overline{L} \in C'_2$. This means L is a result of one or more literals in $C_1 s_1$ by the ground substitution σ. The situation is the same with $\overline{L}$ and $C_2 s_2$. Then there are literals $L_1, ..., L_n \in C_1 s_1$ and $L'_1, ..., L'_m \in C_2 s_2$. This means σ is a unifier of $L = \{L_1, ..., L_n, \overline{L'_1}, ..., \overline{L'_m}\}$. If σ_0 is the most general unifier of L, then there is a substitution s, for which $\sigma_0 s = \sigma$. Then we have $R' = Rs$, which means R' is a ground instance of R. ■

Now we are able to show the resolution theorem:

Theorem 3.36 (Resolution theorem)**.** *Let F be a closed formula in Skolem form. Let $F = \forall x_1 ... \forall x_n G$ and G is in* **CNF**. *Then, F is unsatisfiable iff $G \vdash_{Res} \Box$.*

Proof. We are going to show the two directions of the proof.

- Correctness (soundness)
  Let assume we have a derivation of $\Box$. We have that $F \equiv \forall G \equiv \bigwedge_{C \in G} \forall C$. Let $C_1, C_2 \in G$ and R is their resolvent. We are going to show $\forall R$ is a consequence of $\forall C_1 \wedge \forall C_2$.
  Let $\mathcal{A}$ be a structure such that $\mathcal{A}(\forall C_1) = \mathcal{A}(\forall C_2) = 1$. Let the resolvent have the form $R = (C_1 s_1 \sigma \setminus \{L\}) \cup (C_2 s_2 \sigma \setminus \{\overline{L}\})$, where L is an unificiation of literals in $C_1 s_1$ and dual literals in $C_2 s_2$. If we assume $\mathcal{A}(\forall R) = 0$, then there is a structure $\mathcal{A}'$, for which $\mathcal{A}'(R) = 0$. Here $\mathcal{A}'$ is the same as $\mathcal{A}$, but has suitable variable assignments. Since $\forall C_1$ and $\forall C_2$ are true in $\mathcal{A}$, then they must be true in $\mathcal{A}'$. We have $\mathcal{A}'(C_1 s_1 \sigma \setminus \{L\}) = 0$ and $\mathcal{A}'(C_2 s_2 \sigma \setminus \{\overline{L}\}) = 0$. Then it follows $\mathcal{A}'(L) = \mathcal{A}'(\overline{L}) = 1$, which is not possible. This means $\forall \Box = \Box$ is a conseqence of F, which shows that F is unsatisfiable.

- Completness
  Suppose F is unsatisfiable. From the Ground resolution theorem we know there is a sequence $C'_1, ..., C'_n = \Box$. For every $i = 1, .., n$, C'_i is a ground instance of clauses in G or is propositional resolvent of two clauses C'_a, C'_b, where $a, b < i$. We can now construct a sequence of predicate clauses $C_1, ..., C_n = \Box$. If C'_i is a ground instance of some clause $C \in F$ we set $C_i = C$. Otherwise, we have already determined the clauses C_a, C_b for which C'_a, C'_b are ground instances. By the Lifting lemma there is a clause C, which is a resolvent of C_a, C_b and C'_i is a ground instance of C. Then we set $C_i = C$. This way we have constructed a resolution proof and $G \vdash_{Res} \Box$.

■

We have constructed the Ground Resolution Algorithm and proved its correctness. In the above theorem, we used this to prove the completeness of resolution in predicate logic. Another essential part of the proof was the usage of the Lifting Lemma. Now we can construct an algorithm that performs resolution in predicate logic, as well as in proposition logic.

We can present the algorithm as follows for a formula $F = \forall x_1 ... \forall x_n G$, where G is in **CNF**.

```
1. transform F in **RPF**
2. find $F' = \forall x_1 ... \forall x_n G$ in Skolem form and $F'$ and $F$ are equisatisfiable.
3. Transform G in **CNF**
4. Create the set $S$ of clauses for $G$.
5. **while** (there are clauses $C_a, C_b \in G$ and resolvent R of $C_a$, $C_b$ and $R \notin S$)
   **do** $S = S \cup \{R\}$
      **if** $R$ is $\Box$ **then break**
```

Example 3.37. *Consider the following clause set F:*

$$F = \{\{T(a)\}, \{\neg T(x), \neg Q(f(x))\}, \{Q(f(y)), \neg P(x), Q(x)\}, \{P(x), \neg T(y)\}\}$$

We construct the derivation:

1. $\{T(a)\}$ *// clause in F*
2. $\{\neg T(x), \neg Q(f(x))\}$ *// clause in F*
3. $\{\neg Q(f(a))\}$ *// from 1 and 2*
4. $\{Q(f(y)), \neg P(x), Q(x)\}$ *// clause in F*
5. $\{\neg P(f(a))\}$ *// from 3 and 4*
6. $\{P(x), \neg T(y)\}$ *// caluse in F*
7. $\{P(x)\}$ *// from 1 and 6*
8. $\Box$ *// from 5 and 7*

In comparison with the propositional resolution, it is harder to compute the complexity of the algorithm for predicate logic. For this reason, the so-called Herbrand complexity is used. That is the minimal number of ground clauses (resulting from ground instances) required to get unsatisfiability. The length of resolution steps having ground projections is bounded from below by Herbrand complexity. It turns out that the length of the shortest resolution derivation of a set of clauses F may be logarithmic in the Herbrand complexity of F. Detailed information about this, can be seen in [4].

Using predicate resolution, we can obtain a procedure, which stops at the unsatisfiable formulas and on the valid formulas. For the other formulas we have to consider the size of their models. If a formula has a finite model, then there is a procedure that terminates. However, if a formula has infinite models, then the resolution algorithm might not terminate. This means that resolution is semi-decidable procedure in predicate logic. Using some refinements of resolution we can show the resolution algorithms terminates in specific cases.

The predicate resolution is a great improvement to the Ground Resolution Algorithm mentioned earlier. However, there are many possibilities for producing new resolvents. This leads to a bad time complexity, although only few of the clauses might be needed for the derivation of an empty clause. This was the reason, many resolution refinements were invented.

4 Resolution Refinements

We have seen that resolution in predicate logic acquires resolvents in quite a random way. Earlier we have considered only the general case of resolution. However, there are several resolution restrictions or strategies that can be applied to increase the efficiency in specific cases.

Strategies refer to heuristic rules, which determine the order in which the search space is explored. This means that the search space is the same, but we might be able to find the needed resolution derivation faster. Although some strategies lead to an improvement in the performance, there is little theoretical work behind it.

Resolution restrictions forbid certain steps to be executed. This means we have a smaller search space and better performance than the general resolution. However, when applying such restriction we have to be sure we do not lose the completeness property (if we no longer can find a resolution derivation for an unsatisfiable formula, then this method is not useful).

In the next subsections, we will mention briefly some refinements. These refinements are discussed in Section (2.6) in [3].

4.1 Linear resolution

Definition 4.1. *The empty clause is linearly resolvable from a clause set F, based on a clause $C \in F$, if there is a sequence of clauses $C_0, C_1, ..., C_n$ such that $C_0 = C, C_n = \Box$, and for $i = 1, 2, ..., n$ we have C_i is a resolvent of C_{i-1} and B_{i-1}, where $B_{i-1} \in F$ or $B_{i-1} = C_j$, for $j < i$.*

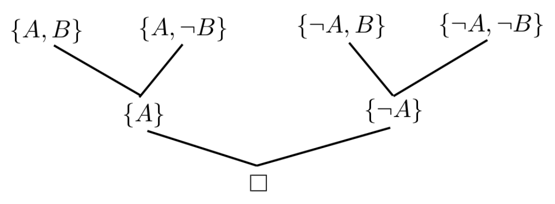
Informally speaking every resolvent in the derivation of the empty clause is a result of the previous resolvent and other clause (side clause).

Example 4.2. *Consider the following unsatisfiable set:*

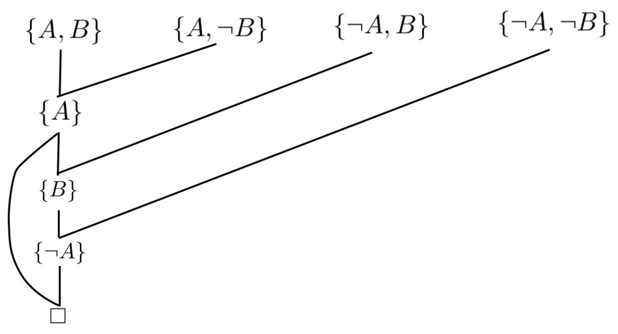
$$F = \{\{A, B\}, \{A, \neg B\}, \{\neg A, B\}, \{\neg A, \neg B\}\}.$$

One way to solve this with general resolution is:

Using general resolution we might have



Using linear resolution we might have



We can see the proof in the general case consists of 3 steps, and using linear resolution we have 4 steps. Using linear resolution we acquire longer proofs, although the search space is smaller (we find the proof faster).

The example is taken from [3].

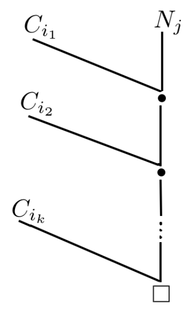
It can be shown that linear resolution is complete and it can be applied to every formula.

4.2 SLD-resolution

SLD-resolution stands for linear resolution with selection function for definite clauses. SLD-resolution is a restriction, which is only defined for Horn clause. Horn clauses are clauses, which contain at most one positive literal. This means a Horn clause has the form:

- Definite clause: $\{\neg L_1, ..., \neg L_n, K\}$
- Fact : $\{K\}$
- Goal clause: $\{\neg L_1, ..., \neg L_n\}$

SLD-resolution is a special case of linear resolution with the following restrictions. The base clause should be a goal clause, and every side clause - a definite clause or a fact. Let $F = C_1, C_2, ..., C_n, N_1, ..., N_m$. Here C_i are definite clauses, or facts, and N_i are goal clauses. A derivation of a proof then consist of a base clause N_j and a sequence $C_{i_1}, ..., C_{i_k}$ of side clauses. The derivation can be represented by a graphic.



This graphic was taken from [3].

It should be noted that every resolution represented in the graph as a dot is a goal clause (does not contain positive literal). There might be also a selection function, which selects the next definite clause to be used. It can be shown that SLD-resolution is correct (sound) and complete for Horn clauses.

5 Conclusion

In this paper, we have considered the resolution technique for deciding satisfiability of formulas. It was shown it is a valid calculus and it is suitable to be programmed. First, We have considered resolution in propositional logic. We have seen that resolution is a powerful method, but there are formulas, for which the derivation requires exponential time. In the section about predicate logic, we have seen how we can apply resolution to ground clauses of a predicate formula. We have seen this is a powerful algorithm, but requires the computation of all ground clauses of a formula. Later we defined resolution for predicate formulas with the help of a most general unifier. We have proved its validity using the Ground resolution theorem and the Lifting lemma. We have shown resolution can be refined and we have given two examples for refinements of resolution.

References

[1] J. A. Robinson, *A Machine-Oriented Logic Based on the Resolution Principle.* Argonne National Laboratory and Rice University

[2] Martin Davis, George Logemann, and Donald Loveland. *A machine program for theorem proving.* Communications of the ACM,5(7):394–397, 1962.

[3] Uwe Schöning, *Logic for Computer Scientists.* Birkäuser Boston, 1989

[4] Matthias Baaz, Alexander Leitsch, *Complexity of resolution proofs and function introduction.* Technische Universität Wien, 1991

[5] A. Haken, *The intractability of resolution, Theoretical Computer Science*, 1985

[6] Samuel R. Buss,György Turán, *Resolution Proofs of Generalized Pigeonhole Principles*, 1988

[7] G. S. Tseitin, *On the complexity of derivations in propositional calculus*, 1970