Refinements Of Resolution
Aanvulling op de syllabus, Resolution in Propositional and Predicate Logic

Hans de Nivelle

February 2, 2001

In this document we discuss *refinements* of resolution. It assumes that the reader is familiar with the contents of ([Doets97]). In ([Doets97]) the unrestricted resolution rule has been introduced, both for propositional logic, and for predicate logic. In this document we will further improve the resolution rule by imposing so called *refinenements*.

1 Resolution with Factoring

In this document the resolution rule will be slightly different from the resolution rule that was used in ([Doets97]). In ([Doets97]) there was the following resolution rule:

Definition 1.1 1. Let C and D be clauses, with no overlapping variables.

2. Let $P \subseteq C$ be a non-empty subset of C that consists of positive literals.
3. Let $N \subseteq D$ be a non-empty subset of D that consists of negative literals.
4. If there exists a substitution σ, such that
   (a) σ makes all literals in P equal to one literal A.
   (b) σ makes all literals in N equal to the opposite $\neg A$,

   then let θ be the mgu.
5. Then $(C - P)\theta \cup (D - N)\theta$ is a resolvent.

The resolution rule as it stands here does two things at the same time. It looks for unifiable literals within one clause, and it looks for unifiable literals in two distinct clauses. It is practically better to separate these two tasks. The resulting two rules are (also) called *resolution*, and *factorization*.

Definition 1.2 We define the following two rules:

Factorization Let C be a clause, which has a subset $S \subseteq C$, consisting of more than 1 element, such that its elements are unifiable. Let θ be the mgu. Then $C\theta$ is called a *factor* of C. If S consists of exactly two elements, then $C\theta$ is called a *binary* factor of C.

Resolution Let C and D be clauses, s.t. there is a positive literal $A_1 \in C$, and a negative literal $\neg\ A_2 \in D$, such that A_1 and A_2 are unifiable. Let θ be the mgu. Then $(C - \{A_1\})\theta \cup (D - \{\neg\ A_2\})\theta$ is called a *resolvent* of C and D.

The advantages of this separation are the following: Suppose that one clause C with unifiable subset P can resolve with many $D_1, \ldots, D_n$. Then for each of the D_i, the computer will unify the elements of P. This will be repeated n times. If the computer first constructs a factor of C, and after that the resolvents, then this unification of the elements of P takes place only once. Even if a clause C has no unifiable literals, then an implementation of the first resolution rule still will *try* to unify all the subsets of C every time it constructs a resolvent.
There is also another advantage: Most resolution provers have a preference for using short clauses over long clauses. This is based on the empirical rule that in proofs short and interesting facts play a rôle. A long clause which the system does not like to use, may have a short factor which the system likes to use. The first variant of resolution cannot notice this situation.
All existing implementations use the second variation of resolution, and most have factorization restricted to the case where S consists of exactly two elements. More is not necessary since the other factors can be obtained by iterating the binary factorization rule.

Example 1.3 The following clause set demonstrates that the factorization rule is necessary. Let Σ be the following clause set:

(1) $\{p(X), p(Y)\}$
(2) $\{\neg\ p(X), \neg\ p(Y)\}$

Without factorization, the only clause that can be derived equals $\{\neg\ p(X), p(Y)\}$. With factorization both clauses 1 and 2 can be factored, resulting in the clauses $\{p(X)\}$ and $\{\neg\ p(X)\}$. The factors resolve into the empty clause.

2 Ordering Refinements

Although resolution was a large step forward compared to the previous methods, which were all based on enumeration of the Herbrand base, it is still not good enough to get somewhere. For a part this is due to the inherent complexity of the problem of finding a proof. It is a search problem with a large, possibly infinite

search space, and there exist no better approaches than searching through this search space.
Fortunately however there are also improvements possible on the resolution rule.
It is possible to further restrict the resolution rule. Such restrictions are called refinements.

Definition 2.1 A *refinement* of resolution is a restriction of the resolution/factorization rule. There are the following types:

1. Certain literals in a clause may be blocked from being used in forming resolvents, or forming factors. The main class of refinements of this type is the class of *ordering refinements*.

2. Complete clauses may be blocked from being used. Refinements of this type are *subsumption*, and the *weight strategies*.

3. Certain conditions on the structure of the resolution derivation may be imposed. The main restriction of this type is *hyperresolution*. Another one is *input* resolution, and *linear* resolution.

There is another fundamental distinction between refinements: *Complete* and *incomplete* refinements. A refinement is complete if every unsatisfiable set of clauses Σ has a derivation of the empty clause $\{\ \}$.

The reader may think that incomplete refinements are useless, but this is not the case. For example a refinement may be incomplete in general, but complete for a certain subclass of first order logic. If the user knoww that the problem is in such a subclass, then such a refinement can be safely used. It is an empirical fact that incomplete refinements work surprisingly well sometimes, much better than complete refinements. An example of such refinements are the weight strategies used by OTTER.
In the rest of this section we will discuss ordering refinements. Most ordering refinements are known to be complete, but the completeness of some types of orders is unknown.

Definition 2.2 An *order* is a relation $\sqsubset$ with the following properties:

IRREFL Relation $\sqsubset$ is irreflexive, i.e. never $A \sqsubset A$.

TRANS Relation $\sqsubset$ is transitive: For all A_1, A_2, A_3,

$$A_1 \sqsubset A_2,\ A_2 \sqsubset A_3 \Rightarrow A_1 \sqsubset A_3.$$

Examples of orders are $<$ on the integer numbers, or on the reals. Another is the alfabetic lexicographic order on words of the Dutch language.

Definition 2.3 Let C be a clause. Let $\sqsubset$ be an order on literals: A literal $A \in C$ is *maximal*, if there is no literal $B \in C$, for which $A \sqsubset B$.

For every order $\sqsubset$, every clause C, different from the empty clause, has maximal elements. This can be easily proven by induction on the size of C.

Definition 2.4 We redefine the resolution rule and the factoring rule, making use of the order:

RES Resolvents are constructed in the same manner as in 1.2, but there are the additional conditions that A_1 should be maximal in $\{A_1\} \cup C$, and $\neg\ A_2$ be maximal in $\{\neg\ A_2\} \cup D$.

FACT Factors are constructed in the same manner as in 1.2, but there is the additional condition that one of the literals in S should be maximal in $S \cup C$.

Example 2.5 Consider the following (propositional) clause set, which is unsatisfiable. It has the following unrestricted resolution refutation. Before each clause we list its generation number, and the number of the clause:

(1)	(1)	$\{a, b\}$
(1)	(2)	$\{a, \neg\ b\}$
(1)	(3)	$\{\neg\ a, b\}$
(1)	(4)	$\{\neg\ a, \neg b\}$
(2)	(5)	$\{\neg\ b\}$
(2)	(6)	$\{b\}$
(2)	(7)	$\{\neg\ a\}$
(2)	(8)	$\{a\}$
(2)	(9)	$\{b, \neg\ b\}$
(2)	(10)	$\{a, \neg\ a\}$
(3)	(11)	$\{\ \}$

Now let $\sqsubset$ be defined from $a \sqsubset b \sqsubset \neg\ a \sqsubset \neg\ b$. When we restrict resolution by this order, there is the following refutation:

(1) (1) $\{a, b\}$
(1) (2) $\{a, \neg\ b\}$
(1) (3) $\{b, \neg\ a\}$
(1) (4) $\{\neg\ a, \neg\ b\}$

(2) (5) $\{a\}$
(2) (6) $\{a, \neg\ a\}$

(3) (7) $\{b\}$

(4) (8) $\{\neg\ a\}$

(5) (9) $\{\ \}$

In total there are less clauses generated, using the order $\sqsubset$. Without $\sqsubset$, there are 11 clauses, with $\sqsubset$, there are 9 clauses. (The difference is small, and the reader is probably not impressed, but this is because the example is small, see Exercise 2.6). Without $\sqsubset$ the proof is found in the 3-d generation. With $\sqsubset$, the proof is found in the 5-th generation. This is a general effect with ordering refinements. Ordering refinements do decrease the total number of generated clauses, but they increase the length of the proof that is found.

Exercise 2.6 Let C be the following clause set:

(1) $\{a_1\}$
(2) $\{\neg\ a_1, a_2\}$
(3) $\{\neg\ a_2, a_3\}$
(4) $\{\neg\ a_3, a_4\}$
(5) $\{\neg\ a_4, a_5\}$
(6) $\{\neg\ a_5\}$

1. Construct a resolution refutation of C, without using a refinement.

2. Construct a resolution refutation, using the following order:

$$a_1 \sqsubset a_2 \sqsubset a_3 \sqsubset a_4 \sqsubset a_5 \sqsubset \neg\ a_1 \sqsubset \neg\ a_2 \sqsubset \neg\ a_3 \sqsubset \neg\ a_4 \sqsubset \neg\ a_5.$$

3. Construct a resolution refutation, using the following order:

$$a_3 \sqsubset a_4 \sqsubset \neg\ a_2 \sqsubset a_5 \sqsubset \neg\ a_3 \sqsubset a_1 \sqsubset \neg\ a_1 \sqsubset a_2 \sqsubset \neg\ a_4 \sqsubset \neg\ a_5$$

We will prove now that in the case of propositional logic every order results in a complete refinement. We will do this by modifying the proof of Theorem 1.17 in ([Doets97]) by constructing the meet more carefully. First remember Definition 1.18: Let P be a collection of sets. A set J is a *minimal meet* of P if

1. for every $C \in P$, we have $J \cap C \neq \{\ \}$.

2. No $J' \subset J$ is also a meet of C.

Theorem 2.7 Let Σ be an unsatisfiable, finite clause set. Let $\sqsubset$ be an order. Then Σ has a resolution refutation that is restricted by $\sqsubset$.

proof: First assume without loss of generality that $\sqsubset$ it total. If $\sqsubset$ is not total then $\sqsubset$ can be made total. (See Exercise 2.8) Let $\overline{\Sigma}$ be the set of clauses that can be obtained from Σ by $\sqsubset$-ordered resolution. We prove that if $\overline{\Sigma}$ does not contain the empty clause, then $\overline{\Sigma}$ has a model. Because $\Sigma \subseteq \overline{\Sigma}$, then also Σ has a model. This implies that if Σ has no model, then a derivation of the empty clause is possible.
There is only a finite number of literals in Σ, and hence in $\overline{\Sigma}$. We construct a minimal meet M using the following algorithm: n is the number of literals in Σ.

$M := \{\ \}$;
for $i := 1$ to n do
begin
    $L :=$ the i-th literal in the order $\sqsubset$.
    if there is a clause $C \in \overline{\Sigma}$, such that $M \cap C = \emptyset$, and L is the maximal literal of C,
    then
        $M := M \cup \{L\}$;
end

It is clear that M is a meet. Let $C \in \overline{\Sigma}$. At some point the algorithm will reach the maximal literal L of C. If $C \cap M = \emptyset$ at this moment, then L will be added to M. After that certainly $M \cap C \neq \{\ \}$.
Now M is minimal because of the following property:

MAXUNIQUE For every literal $L \in M$, there is a clause $C \in \overline{\Sigma}$, such that L is the maximal element of C, and there is no literal different from L, that occurs in C and M.

MAXUNIQUE holds by the way in which M is constructed. At the moment that a literal L is added to M, it is maximal and unique in a clause C, because of the condition of the if-statement. After that the uniqueness cannot be spoilt because only larger clauses are added, and L is already maximal in C.
From MAXUNIQUE follows that M minimal because no literal can be deleted from M.
Using MAXUNIQUE we prove: There is no literal L for which both $L \in M$, and $\neg\ L \in M$. Suppose there were one. For both L and $\neg\ L$ there would be a

clause $C_L \in \overline{\Sigma}$, s.t. L is the only literal common to M and C_L, and a clause $C_{\neg L} \in \overline{\Sigma}$, for which $\neg\ L$ is the only literal common to both M and $C_{\neg L}$.
Since L is maximal in C_L, and $\neg\ L$ is maximal in $C_{\neg L}$, the resolvent $C_r = (C_L - \{L\}) \cup (C_{\neg L} - \{\neg\ L\})$ is possible. Because M is a meet, $C_r \cap M \neq \{\ \}$.
Let L' be common to M and C_r. L' is inherited either from C_L or from $C_{\neg L}$. Whichever it is, in both cases L' contradicts MAXUNIQUE.
So we have constructed a meet M of $\overline{\Sigma}$, that does not contain a complementary pair of literals. Using M we easily construct a model of $\overline{\Sigma}$ by putting

$$I(A) = t \text{ iff } A \in M.$$

What we have proven here is a bit weaker than Theorem 1.17, because we proved completeness only for finite sets. This is sufficient for the completeness, because of the compactness property of propositional logic. We could also have proven Theorem 2.7 directly for infinite sets, but this needs the principle of transfinite induction.

Exercise 2.8 Prove the claim that is made in the first line of the proof: If $\sqsubset \subseteq \sqsubset'$, and $\sqsubset'$ allows a certain derivation, then $\sqsubset$ also allows this derivation.

Exercise 2.9 Show that I is indeed a model of $\overline{\Sigma}$.

Exercise 2.10 Let C be the satisfiable clause set: $C = \{\{a, b\}, \{\neg\ a, b\}, \{a, \neg\ b\}\}$. Let $\sqsubset$ be the order

$$b \sqsubset \neg\ a \sqsubset a \sqsubset \neg\ b.$$

Construct all resolvents of C. Simulate the algorithm in the proof of Theorem 2.7, and construct a model of C.

We will now turn our attention to predicate logic. Here the situation is more complicated than in the case of propositional logic. The problem is the following: Let $\{p(X), q(X), r(X)\}$ be a clause, s.t. $p(X) \sqsubset q(X) \sqsubset r(X)$. If C resolves with $D = \{\neg\ r(0)\}$, the result equals $\{p(0), q(0)\}$. Now it may be that $q(0) \sqsubset r(0)$, so the literals may change their position in the ordering during the deduction process. Because of this effect it is not known for all orders whether or not resolution in predicate logic is complete. However the following types of orders are known to be complete:

Definition 2.11 Let $\sqsubset$ be an order on literals:

- We call $\sqsubset$ *liftable* if $A \sqsubset B$ implies $A\theta \sqsubset B\theta$ or $A\theta = B\theta$, for every substitution θ.
- We call $\sqsubset$ *descending* if
  1. $A \sqsubset B$ implies $A\theta_1 \sqsubset B\theta_2$ for all renaming substitutions θ_1 and θ_2.

2. $A\theta \sqsubset A$, whenever $A\theta$ is not a renaming of A.

We will prove the completeness only for liftable orders. The proof will be in the section on subsumption. The completeness of descending orders can be proven by a technique called the *resolution game*.

Example 2.12 Let Σ be the following clause set:

(1) (1) $\{p(0)\}$
(1) (2) $\{\neg\, p(X), p(s(X))\}$
(1) (3) $\{\neg\, p(s(s(s(0))))\}$

Let $\sqsubset$ be defined from: $p(X) \sqsubset p(s(X))$, together will all its instances: There is the following resolution refutation:

(2) (4) $\{\neg\, p(s(s(0)))\}$
(3) (5) $\{\neg\, p(s(0))\}$
(4) (6) $\{\neg\, p(0)\}$
(5) (7) $\{\ \}$.

Exercise 2.13 Let $\sqsubset$ be defined from: $A \sqsubset B$ if A is a negative literal, and B is a positive literal. Construct an ordered refutation of Σ from the previous example. Is $\sqsubset$ liftable?

Exercise 2.14 Let $\sqsubset$ be defined from: $A \sqsubset B$ if the maximal depth of an occurrence of a variable in A is less than the maximal depth of an occurrence of a variable in B. Is $\sqsubset$ liftable? Is $\sqsubset$ or $\sqsupset$ descending?

3 Hyperresolution

Another important refinement is hyperresolution. Hyperresolution was also introduced by Robinson, but in another paper. ([**?**]). Hyperresolution does not construct resolvents of two clauses, but of many clauses at the same time.

Definition 3.1 Let $C = \{\neg\, A_1, \ldots, \neg\, A_p, B_1, \ldots, B_q\}$ be a clause, where all the A_i are negative literals, and all the B_j are positive literals. Let $C_1 = \{A'_1\} \cup R_1, \ldots, C_p = \{A'_p\} \cup R_p$ be p clauses, which contain only positive literals. Assume that the clauses contain no overlapping variables. (If they do, then rename them. This is even necessary if one positive clause occurs twice) If there exists a substitution σ, such that $A_1\sigma = A'_1\sigma, \ldots, A_p\sigma = A'_p\sigma$, then let θ be a most general unifier. The following clause is called a *hyperresolvent* of C, and $C_1, \ldots, C_p$:

$$\{B_1\theta, \ldots, B_q\theta, R_1\theta, \ldots, R_p\theta\}.$$

Sometimes C is called the *nucleus*, and the C_i are called *electrons*, or also *satellites*

We give an example:

Example 3.2 The clauses $\{a, b\}$ and $\{a, \neg\ b\}$ hyperresolve into $\{a\}$. $\{a, b\}$ is an electron, and $\{a, \neg\ b\}$ is the nucleus. Clauses $\{a, b\}, \{b, c\}$ and $\{\neg\ a, \neg\ b\}$ hyperresolve into $\{b, c\}$. Here $\{a, b\}$ and $\{b, c\}$ are the electrons, and $\{\neg\ a, \neg\ b\}$ is the nucleus.
Consider the clause set:

(1) (1) $\{a, b\}$
(1) (2) $\{a, \neg\ b\}$
(1) (3) $\{\neg\ a, b\}$
(1) (4) $\{\neg\ a, \neg\ b\}$

It has the following hyperresolution refutation:

(2) (5) $\{a\}$
(2) (6) $\{b\}$
(3) (7) $\{\ \}$

Example 3.3 Look at the following clause set:

(1) (1) $\{\neg\ p(X), \neg\ q(X), r(X, Y)\}$
(1) (2) $\{p(a)\}$
(1) (3) $\{q(b)\}$
(1) (4) $\{\neg\ r(X, Y), s(X, Y)\}$
(1) (5) $\{\neg\ s(X, Y)\}$

It has the following hyperresolution refutation:

(2) (6) $\{r(a, b)\}$
(3) (7) $\{s(a, b)\}$
(4) (8) $\{\ \}$.

There is no other refutation.

Before we prove the completeness we give two variations of hyperresolution:

Definition 3.4 Let $\sqsubset$ be an order on the positive literals. *Ordered* hyperresolution is obtained by demanding that the A'_i in the purely positive clauses are maximal in their clauses.
Negative hyperresolution is obtained by replacing the rôle of positive and negative literals in Definition 3.1.

What we will show is that hyperresolution (almost) can be simulated by ordered resolution. From this the completeness of hyperresolution can be proven.
Let $\sqsubset$ be an order on positive literals. We define the following order on all literals: $A \prec B$ iff either

1. Both A and B are positive, and $A \sqsubset B$.

2. A is positive, and B is negative.

The order $\prec$ forces the derivation to have the structure of a $\sqsubset$-ordered hyperderivation. Let $\{\neg\ A_1, \ldots, \neg\ A_p, B_1, \ldots, B_q\}$ be a mixed clause. All literals $\neg\ A_i$ are maximal. Now if C resolves with another clause C_1, this must be a purely positive clause, because the literal resolved upon must be positive, and maximal. Any negative literal in C_1 would make all positive literals non-maximal. So C_1 has form $\{A_1'\} \cup R_1$. If $p > 1$, the result $R_1 = \{\neg\ A_2\theta, \ldots, \neg\ A_p\theta, B_1\theta, \ldots, B_q\theta, R_1\theta\}$ contains $p-1$ negative literals and must again resolve with a purely positive clause $\{A_2'\} \cup R_2$. This repeats p times. Eventually R_p is a purely positive clause. This R_p is a hyperresolvent of C and $C_1, \ldots, C_p$. Now if $\sqsubset$ is a descending/liftable order on the positive literals, then $\prec$ is a descending/liftable order. It follows from the results of the previous section that hyperresolution is a complete refinement, even when it is combined with an order on the positive literals.
The reason that this argument is not complete, is that there may be applications of the factorization rule in between, on the R_i. In order to make the argument complete one has to show that applications of the factorization rule can be postponed until a purely positive clause is reached.

Exercise 3.5 1. Prove that if $\sqsubset$ is liftable, then $\prec$ is liftable.

2. Prove that if $\sqsubset$ is descending, then $\prec$ is descending.

4 Subsumption

In the previous section we discussed ordering refinements. Ordering refinements block certain literals in clauses from being used in forming resolvents. The next refinement is called *subsumption* and blocks complete clauses from the derivation process. Subsumption was already present in the original paper on resolution ([Robins65]). The underlying idea is simple. Suppose that we have two clauses $C_1 = \{A\} \cup R_1$ and $C_2 = \{A\} \cup R_1 \cup R_2$, and a clause $D = \{\neg\ A\} \cup S$. We have $C_1 \subseteq C_2$. The resolvent of C_1 with D equals $R_1 \cup S$, which is a subset of the resolvent of C_2 with D, $R_1 \cup R_2 \cup S$.
Now suppose that we have a clause $C_3 = R_1 \cup R_2$. Clause C_3 cannot resolve with D, but C_3 itself is a subset of $R_1 \cup R_2 \cup S$.
So in general if we have two clauses C_1 and C_2 and $C_1 \subseteq C_2$, there is no point in using clause C_2 since for every clause that can be derived using C_2 there is a subset that can be derived using C_1. We say that C_1 *subsumes* C_2 if C_1 is a subset of C_2. When C_1 subsumes C_2, we can forget C_2.
In the case of predicate logic, the situation is more complicated:

Definition 4.1 Let C_1 and C_2 be clauses. We say that C_1 *subsumes* C_2 if

1. $\mid C_1 \mid\ \leq\ \mid C_2 \mid$, and

2. There is a substitution θ, such that $C_1\theta \subseteq C_2$.

We have to include substitutions because of the following: The clause $C_1 = \{q(X), r(X)\}$ is a subset of $C_2 = \{p(X), q(X), r(X)\}$. If the second clause is resolved with $\{\neg\, p(0)\}$, the result equals $\{q(0), r(0)\}$. Clearly C_1 is not a subset of this clause. However $C_1\{X := 0\}$ is.
Also, apart from this consideration, it is better to include substitutions, because this leads to a more general notion of subsumption. The stronger subsumption we have, the more clauses can be deleted.
The length condition is necessary because of interaction with the factorization rule. Suppose that C_2 is a factor of C_1. Then $C_2 = C_1\theta$, and as a consequence C_2 subsumes C_1. Since the plan is to block inferences from clauses that are subsumed this would effectively mean that we delete the factorization rule. Since we know that factorization is necessary, this should be avoided.
We will now formally prove that resolution with subsumption is complete. First we need a technical fact, stating that the length condition is harmless.

Lemma 4.2 Suppose that there are two clauses C_1 and C_2, and that $C_1\theta \subseteq C_2$, but $\mid C_1 \mid > \mid C_2. \mid$. Then there is a (possibly) iterated factor F of C_1, such that F subsumes C_2.

proof: We prove the property by induction to $k = \mid C_1 \mid - \mid C_2 \mid$.
Supppose that $k > 0$, and that $C_1\theta \subseteq C_2$. Because $\mid C_1 \mid > \mid C_2 \mid$, there must be two literals in C_1, such that $A_1\theta = A_2\theta$.
Let $F' = C_1\sigma$, be the binary factor of C_1 that is obtained by unifying A_1 and A_2. Because σ is a most general unifier, of A_1 and A_2, and θ is a unifier of A_1 and A_2, there must exist a substitution ξ, such that $\theta = \sigma \cdot \xi$. It follows that $F'\xi \subseteq C_2$. Also the length of F' is at least one less than the length of C_1. By the induction hypothesis there exists a (possibly) iterated factor F of F', such that F subsumes C_2. Because F is also an iterated factor of C_1, the proof is complete.

Lemma 4.3 Let C_1 and C_2 be clauses, such that C_1 subsumes C_2. Let F_2 be a binary factor of C_2. Then either C_1 subsumes F_2, or there is a binary factor F_1 of C_1 that subsumes F_2.

proof: Write $C_2 = \{A_1, A_2\} \cup S$. Let θ be the most general unifier of A_1 and A_2. So we have $F_2 = \{A_1\theta\} \cup S\theta$.
We also know that C_1 subsumes C_2, so there exists a substitution σ such that $C_1\sigma \subseteq C_2$. Then $C_1\sigma\theta \subseteq C_2\theta$. It follows that C_1 and F_2 satisfy the first condition of subsumption. If C_1 is too long then apply Lemma 4.2.

Lemma 4.4 Let C_1 and C_2 be clauses, such that C_1 subsumes C_2. Let R_2 be a resolvent of a clause D with C_2. Then either (an iterated factor of) C_1 subsumes R_2, or there is (an iterated factor of) a resolvent R_1 of C_1 and D that subsumes R_2.

proof: So we have $C_1\sigma \subseteq C_2$. Write $C_2 = \{A_1\} \cup S_2$. $D = \{\neg\ A_2\} \cup T$. Let θ_2 be the most general unifier of A_1 and A_2. So the resolvent R_2 equals $S_2\theta_2 \cup T\theta_2$. We distinguish three cases:

1. $A_1 \notin C_1\sigma$. Then $C_1\sigma \subseteq S_2$. Then $C_1\sigma\theta \subseteq S_2\theta$. Hence C_1 subsumes R_2 (with the possible exception of the length restriction). If C_1 is too long then by Lemma 4.2, there is a (iterated) factor of C_1 that subsumes R_2.

2. $A_1 \in C_1\sigma$, and there is exactly one literal $B \in C_1$, for which $B\sigma = A_1$.

   Write $C_1 = \{B\} \cup S_1$, and for no literal $A' \in S_1$, we have $A'\sigma = A_1$. Then $C_1\sigma \subseteq S_2$.

   Because $B\sigma = A_1$, and $A_1\theta = A_2\theta$ it is possible to resolve C_1 and D. Write $R_1 = S_1\theta_1 \cup T\theta_1$, where θ_1 is the most general unifier of B and A_2.

   $\sigma \cdot \theta_1$ is a substitution that makes B and A_2 equal. (Here we are using the fact that $C_1, C_2,\ D$ have no overlapping variables) Hence is possible to write $\sigma \cdot \theta_2 = \theta_1 \cdot \xi$, for some ξ. We will show that $R_1\xi \subseteq R_2$.

   Let $A' \in R_1\xi = S_1\theta_1\xi \cup T\theta_1\xi = S_1\sigma\theta_2 \cup T\sigma\theta_2$. If $A' \in T\sigma\theta_2$, then clearly $A' \in R_2$, because $T\sigma = T$. (because of the non-overlapping variables) Otherwise suppose that $A' \in S_1\sigma\theta_2$. Because $S_1\sigma \subseteq S_2$, it follows that $A' \in S_2\theta_2$.

3. Suppose that $A_1 \in C_1\sigma$, and there more than 1 literal $B_1, \ldots, B_n$, such that $B_i\sigma = A_1$. We will reduce this case to the second case by using the factorization rule.

   Let ρ be the most general unifier of B_1 and B_2. Since $B_1\sigma = B_2\sigma$, there must exist ξ, such that $C_1\rho\xi \subseteq C_2$. Hence $C_1\rho$ subsumes C_2. The length restriction is automatically satisfied. By iteration this procedure we eventually reach case 2.

So subsumption can be used to delete clauses that are subsumed. Usually two types of subsumption are distinguished.

1. If a new clause C is generated, but there is already a clause C_1, such that C_1 subsumes C, then C is *forward* subsumed by C_1. In that case C is not kept.

2. If a new clause C is derived, and C subsumes a clause C_1 that is already present in the database, then C_1 is *backward* subsumed by C. In that case C_1 is deleted.

We will now prove that subsumption can be combined with liftable orders, (This is the postponed proof that we promised in Section 2) without losing completeness.

Lemma 4.5 Let $C_1\theta \subseteq C_2$. Let $\sqsubset$ be an order that satisfies the liftability property. Let A be maximal in C_2. Let $A_1, \ldots, A_n$ be the elements of A for which $A_i\theta = A$, and assume that $n > 0$. Then one of the A_i is maximal in C_1.

proof: We show that $A_i \sqsubset B$ implies that B is among the A_j. Because there are only finitely many A_i we will reach a maximum within the A_i.
Suppose that $A_i \sqsubset B$. By liftability of $\sqsubset$, it follows that

$$A_i\theta \sqsubset B\theta \text{ or } A_i\theta = B\theta.$$

The first case cannot occur, because this would imply that $A = A_i\theta$ is not maximal in C_2, since $B\theta \in C_2$.
In the second case B is one of the A_j. This fact can be used in the proofs of Lemma 4.3 and the 3d case in the proof of Lemma 4.4. As a consequence Lemma 4.3 and Lemma 4.4 are true for ordered subsumption, when the order is liftable.

Exercise 4.6 Check that this is really the case.

Now we have proven the following: If $\{\ \}$ can be derived from a certain clause set Σ, using a liftable order $\sqsubset$, then this is also possible when subsumption is used.
We also proved that propositional resolution is complete when it is combined with an arbitrary order. But did we prove that $\sqsubset$-ordered resolution without subsumption is complete? The answer is surprisingly yes:

Theorem 4.7 Resolution remains complete when it is combined with a liftable order $\sqsubset$, and when it is combined with subsumption.

proof: Let Σ be unsatisfiable. By Herbrand's theorem (Theorem 2.28 in ([Doets97]), there exists a finite set Σ' of ground instances of Σ, that is unsatisfiable. By Lemma 2.7, there exists a $\sqsubset$-ordered resolution refutation of $\Sigma \cup \Sigma'$, using only the clauses from Σ'. Now the clauses in Σ subsume the clauses in Σ', and so, by Exercise 4.6, the clauses in Σ' can be deleted without losing the derivation.

Exercise 4.8 Prove that the subsumption relation is transitive: If C_1 subsumes C_2, and C_2 subsumes C_3, then C_1 subsumes C_3.

5 Weight Strategies

In this section we discuss the weight strategies that are used by OTTER. The strategy is simple. Assign to each clause a weight that depends on its complexity, for example by counting the total number of symbols in it. Then **(1)** prefer derivations that can be made from the clauses with the lowest weight, and **(2)**

don't keep any clauses that have a weight larger than a certain maximal weight. The idea behind this is that the empty clause has zero weight and clauses with a low weight are probably nearer to the empty clause, than clauses with a high weight. If the weight is too high than the clause will not play a rôle in the deduction, and it is better to delete it.

Definition 5.1 Let t be a term. We define the *weight* of t, notation $\#t$ recursively as follows:

1. If t is a variable X, or a constant c, then $\#c = 1$.
2. Otherwise t has form $f(t_1, \ldots, t_n)$. The weight $\#t = 1 + \#t_1 + \cdots + \#t_n$.

The weight of an atom $p(t_1, \ldots, t_n)$, and a negative literal $\neg\, p(t_1, \ldots, t_n)$ both equal $\#t_1 + \cdots + \#t_n$. The weight of a clause $\{A_1, \ldots, A_n\}$ equals $\#A_1 + \cdots + \#A_n$.

OTTER allows more sophisticated weight functions, for example by assigning different weights to the different function or predicate symbols. All these extensions are fairly trivial and they can be checked in the manual.
The derivation process in OTTER goes as follows: OTTER has two lists of clauses, **usable** and **sos**. Sos means *set of support*. Then: As long as **sos** is not empty, OTTER selects a clause c with minimal weight from **sos**, and moves c into **usable**. After that OTTER computes all possible resolvents between c, and the clauses in **usable**. Those resolvents which do not have too much weight, which are not forward subsumed, are kept and added to **sos** together with their factors.
This is repeated until **sos** is empty, or until the empty clause is derived.
Note that OTTER *never* resolves two clauses from **usable**. If you want OTTER to construct all possible resolvents, it is necessary to put all initial clauses in **sos**. If you put all initial clauses in **usable** OTTER will not compute any resolvents at all.
OTTER generates factors immediately when a clause is kept, and not when a clause is selected. This is because a factor possibly has a smaller weight than its parent. It is possible that the factor will be selected, and plays an important rôle in the proof, while the parent will never be selected.

Example 5.2 We will conclude with a larger example: We prove that the composition of an order with itself is also an order. We use ordered, positive hyperresolution and a weight strategy. We have the following premisses:

1. $\forall x[\neg\, x < x]$.
2. $\forall xyz[x < y \wedge y < z \rightarrow x < z]$.
3. $\forall xy[x \sqsubset y \leftrightarrow \exists z(x < z \wedge z < y)]$.

Formulae 1 and 2 state that $<$ is an order. Formula 3 states the definition of $\sqsubset$. We want to prove that $\sqsubset$ is also an order:

1. $\forall x[\neg\ x \sqsubset x]$.
2. $\forall xyz[x \sqsubset y \wedge y \sqsubset z \rightarrow x \sqsubset z]$.

Clausification gives the following clause set. Each clause has a generation number, an index, and a weight.

(1)	(1)	$\{\neg\ X < X\}$	[2]
(1)	(2)	$\{\neg\ X < Y, \neg\ Y < Z, X < Z\}$	[6]
(1)	(3)	$\{\neg\ X \sqsubset Y, X < f(X,Y)\}$	[6]
(1)	(4)	$\{\neg\ X \sqsubset Y, f(X,Y) < Y\}$	[6]
(1)	(5)	$\{\neg\ X < Z, \neg\ Z < Y, X \sqsubset Y\}$	[6]
(1)	(6)	$\{c \sqsubset c, d_1 \sqsubset d_2\}$	[4]
(1)	(7)	$\{c \sqsubset c, d_2 \sqsubset d_3\}$	[4]
(1)	(8)	$\{c \sqsubset c, \neg\ d_1 \sqsubset d_3\}$	[4]

Clauses 6,7,8 are obtained from the negation of the conclusion. The function f is a Skolem function. The constants c, d_1, d_2, d_3 are Skolem constants. We use a simple order on the positive literals. Atoms based on $<$ are preferred over atoms based on $\sqsubset$. We start with the positive clauses (6) and (7) in **sos**, and the other clauses in **usable**. When two clauses have equal weight the one with the lowest index is selected. Subsumed clauses are not deleted, but no inferences will be made from them.
First clause (6) is selected. Both literals in clause (6) are maximal, and there are 4 hyperresolvents, based based on the mixed clauses (3) and (4).

(2)	(9)	$\{c < f(c,c), d_1 \sqsubset d_2\}$	[6]
(2)	(10)	$\{f(c,c) < c, d_1 \sqsubset d_2\}$	[6]
(2)	(11)	$\{d_1 < f(d_1,d_2), c \sqsubset c\}$	[6]
(2)	(12)	$\{f(d_1,d_2) < d_2, c \sqsubset c\}$	[6]

Clause (7) is selected. This results in another 4 hyperresolvents:

(2)	(13)	$\{c < f(c,c), d_2 \sqsubset d_3\}$	[6]
(2)	(14)	$\{f(c,c) < c, d_2 \sqsubset d_3\}$	[6]
(2)	(15)	$\{d_2 < f(d_2,d_3), c \sqsubset c\}$	[6]
(2)	(16)	$\{f(d_2,d_3) < d_3, c \sqsubset c\}$	[6]

Clause (9) is selected. There are no hyperresolvents possible, because $c < f(c,c)$ is maximal. Clause (10) is selected. There are two hyperresolvents, using mixed clauses (2) and (5):

(3) (17) $\{c < c, d_1 \sqsubset d_2\}$ [4]
(3) (18) $\{f(c,c) < f(c,c), d_1 \sqsubset d_2\}$ [8]
(3) (19) $\{f(c,c) \sqsubset f(c,c), d_1 \sqsubset d_2\}$ [8]

The other clause equals $\{c \sqsubset c, d_1 \sqsubset d_2\}$, and is not kept, because it is forward subsumed by (6). Now clause (17) is selected. $c < c$ is the maximal literal. The result is:

(4) (20) $\{d_1 \sqsubset d_2\}$ [2]

This clause backward subsumes clauses (6), (9), (10), (17), (18) and (19). Clause (20) is immediately selected. The results are:

(5) (21) $\{d_1 < f(d_1, d_2)\}$ [4]
(5) (22) $\{f(d_1, d_2) < d_2\}$ [4]

Clause (21) is selected without result. Clause (22) is selected, and

(6) (23) $\{d_1 < d_2\}$ [2]

is born, which is immediately selected without result. Now clause (11) is selected. Zero is the result. Then clause (12) is selected. There are two hyperresolvents, but both are forward subsumed: $\{d_1 < d_2, c \sqsubset c\}$ and $\{d_1 \sqsubset d_2, c \sqsubset c\}$. After that clause (13) and (14) are selected. There are 4 hyperresolvents, but $\{c \sqsubset c, d_2 \sqsubset d_3\}$ is forward subsumed:

(3) (24) $\{c < c, d_2 \sqsubset d_3\}$ [4]
(3) (25) $\{f(c,c) < f(c,c), d_2 \sqsubset d_3\}$ [8]
(3) (26) $\{f(c,c) \sqsubset f(c,c), d_2 \sqsubset d_3\}$ [8]

Clause (24) is selected, and the result is:

(4) (27) $\{d_2 \sqsubset d_3\}$ [2]

Clause (27) backward subsumes (7), (13), (14), (24), (25), and (26). Clause (27) is selected and there appear:

(5) (28) $\{d_2 < f(d_2, d_3)\}$ [4]
(5) (29) $\{f(d_2, d_3) < d_3\}$ [4]

Clauses (15) and (16) are backward subsumed. Clause (28) is selected, and we derive:

(6) (30) $\{f(d_1, d_2) < f(d_2, d_3)\}$ [6]
(6) (31) $\{f(d_1, d_2) \sqsubset f(d_2, f_3)\}$ [6]
(6) (32) $\{d_1 < f(d_2, d_3)\}$ [4]
(6) (33) $\{d_1 \sqsubset f(d_2, d_3)\}$ [4]

After that (29) is selected, and we obtain:

(7) (34) $\{d_2 < d_3\}$ [2]

This short clause is immediately selected.

(8) (35) $\{d_1 < d_3\}$
(8) (36) $\{d_1 \sqsubset d_3\}$

Clause (35) is selected without result. After that (36) is selected. It should be clear by now that a proof is going to be found. (36) hyperresolves with (8) into $\{c \sqsubset c\}$. This clause hyperresolves into $\{c < f(c, c)\}$ and $\{f(c, c) < c\}$. The result is $\{c < c\}$ and from this clause the empty clause is derived.

References

[Doets97] H.C. Doets, Resolution in Propositional and Predicate Logic, Syllabus bij het college Theorem Proving, 1997.

[Robins65] J. A. Robinson, A Machine Oriented Logic Based on the Resolution Principle, Journal of the ACM, Vol. 12, pp. 23-41, 1965.