Ordering Refinements of Resolution

Hans de Nivelle

Ordering Refinements of Resolution

PROEFSCHRIFT

ter verkrijging van de graad van doctor
aan de Technische Universiteit Delft,
op gezag van de Rector Magnificus Prof. ir. K.F. Wakker,
in het openbaar te verdedigen ten overstaan van een commissie,
door het College van Dekanen aangewezen,
op donderdag 12 oktober te 10.30 uur

door

Jean Marie Guillaume Gérard de NIVELLE,

ingenieur in de Informatica,

geboren te Waalre.

Dit proefschrift is goedgekeurd door de promotor:
Prof. dr. S.C. van Westrhenen,

Toegevoegd promotor:
Dr. R. Sommerhalder.

Samenstelling promotiecommissie:

Rector Magnificus, voorzitter
Prof. dr. S.C. van Westrhenen, (T.U. Delft) promotor,
Dr. ir. R. Sommerhalder, (T.U. Delft) toegevoegd promotor,
Prof. S.J. Doorman, M. Sc., (T.U. Delft)
Prof. dr. A. Leitsch, (T.U. Wien)
Prof. dr. J.-J. Ch. Meyer, (R.U. Utrecht)
Prof. dr. H.C.M. de Swart. (K.U. Brabant)

Contents

1 Introduction to this Thesis **7**
- 1.1 Resolution . . . 7
- 1.2 Ordering Refinements . . . 9
- 1.3 Resolution Games . . . 11
- 1.4 Outline of this Thesis . . . 12

2 Preliminaries **15**
- 2.1 Orders and Ordinal numbers . . . 15
- 2.2 The Axiom of Choice . . . 17
- 2.3 Trees . . . 18
- 2.4 Multisets . . . 22

3 Logic Preliminaries **23**
- 3.1 First order languages . . . 23
- 3.2 Semantics of first order logic . . . 25
- 3.3 A Sequent Calculus . . . 26
- 3.4 Semantic Tableaux . . . 30
- 3.5 Skolemisation . . . 32
- 3.6 Theorem of Herbrand . . . 36
- 3.7 Unification . . . 38
- 3.8 Resolution Theorem Proving . . . 45
- 3.9 Refinements of Resolution . . . 53

4 How to Obtain Resolution Calculi **59**
- 4.1 Introduction . . . 59
- 4.2 From Sequent Calculi to Resolution . . . 61
- 4.3 Maslov's inverse method . . . 72
- 4.4 From Semantic Tableaux to Resolution . . . 73
- 4.5 Refinements . . . 87
- 4.6 Comparison between the 2 methods . . . 90

4.7 Why do ordering refinements work? 95

5 Resolution in modal logic **97**
5.1 Modal Logic 98
5.2 Semantics of Modal logic 101
5.3 Normal Forms 104
5.4 Examples to Resolution 107
5.5 Technical Preparation 110
5.6 Resolution 115
5.7 Conclusions 124

6 Resolution Games **127**
6.1 Blue Resolution Games 128
6.2 Completeness by Combination 133
6.3 Direct Completeness 137
6.3.1 Completeness of Restricted Resolution Games 138
6.3.2 Completeness of the Full Resolution Game 140
6.4 Simultaneous Resolution Games 143
6.5 Yellow Resolution Games 147
6.6 Completeness of Yellow Resolution Games 150
6.7 Factorisation Causes Incompleteness 152

7 Non-liftable Orderings **157**
7.1 Example 159
7.2 Preliminaries 160
7.3 Decreasing Orders 168
7.4 Decomposable Clauses 171
7.5 Clauses consisting of two literals 176
7.6 Conclusions and Future Work 181

8 Epilogue **183**

A Ordening Verfijningen van Resolutie **191**
A.1 Inleiding 191
A.2 Samenvatting 194

B Curriculum Vitae **197**

Chapter 1

Introduction to this Thesis

1.1 Resolution

This thesis is about techniques for automated proof search. Resolution is one of the most successful methods for automated proof search. It was developed in ([Robins65]). It is based on the following rule: Suppose that the following formulae are true:

1. not(A) or B_1 or $\cdots$ or B_p
2. A or C_1 or $\cdots$ or C_q.

Then the following must also be true:

3. B_1 or $\cdots$ or B_p or C_1 or $\cdots$ or C_q.

This is due to the following: If a formula of the form X_1 or $\cdots$ or X_n is true, this means that it is possible to select an X_i which is true. The same holds for formulae (1) and (2), but it is impossible to choose A and not(A) simultaneously, because A and not A cannot be true together at the same time. Therefore, in order to make (1) and (2) true on of the $B_1, \ldots, B_q$, or $C_1, \ldots, C_q$ must be chosen. But this can be written as B_1 or $\cdots$ or B_p or C_1 or $\cdots$ or C_q.

Example 1.1.1 If either it is night or day, and it is either not night, or dark, then one can derive that it is either day or dark.

It is also possible to apply this rule when the B_i are absent:

1. $not(A)$, with

2. A or C_1 or $\cdots$ or C_q.

gives

3. C_1 or $\cdots$ or C_q.

When the C_i are absent this rule takes the form:

1. $not(A)$ or B_1 or $\cdots$ or B_p, with
2. A

gives

3. B_1 or $\cdots$ or B_p.

For example if we are very certain that it is either light or dark, and we know that it is not light, then we can have confidence that it is dark.

Definition 1.1.2 We will call formulae of the type X_1 or $\cdots$ or X_n, a *clause*.

It is also possible to iterate the resolution rule: When a new clause is derived with resolution, it may be used again.

Example 1.1.3 Suppose, for example that there is the following:

1. A or B or C,
2. not(A) or B or C,
3. not(B) or C,
4. not(C).

(1) and (2) resolve to B or C or B or C, Because B or B has the same meaning as B, we can remove the double occurrences. This is called *factorisation*. Using factorisation the result will be

5. B or C.

This can be resolved with (3) to obtain (after factorisation):

6. C.

Now there is the following situation:

4. not(C),

6. C

This is a special situation, because this combination of C and not(C) is impossible. Because not (C) is obtained by resolution, it must be that not(C) is true if the initial clauses are true. But not(C) cannot be true because of the presence of C. This means that we have shown that the initial set $(1), \ldots, (4)$ can never be true at the same time. The interesting thing now is that the reverse also holds: If it is impossible that a certain initial set of clauses is true, then iteration of the resolution rule will derive a contradiction. This is in fact, what is proven in ([Robins65]).

Definition 1.1.4 A list of clauses is called *inconsistent* if every list of choices for the or's in the clauses will produce a contradictory pair of the form A and not(A).

For example, the initial clause list of Example 1.1.3 is inconsistent.

Theorem 1.1.5 If a set of clauses is inconsistent, then it is possible to derive a contradiction with resolution.

The resolution technique, although a bit childish, is very suited for automated proof search. It will be used as follows: Suppose that one wants to prove that A follows logically from a set of formulae $F_1, \ldots, F_n$. Then $F_1, \ldots, F_n$ and not (A) cannot be true at the same time. They can be transformed into clauses. On the resulting clauses resolution can be applied. If A was indeed a logical consequence of $F_1, \ldots, F_n$, then a contradiction will be derived, because of Theorem 1.1.5.

1.2 Ordering Refinements

It is possible to add preference rules to resolution. These rules can have the following form:

- Never use the A in a clause, when there is a B in the same clause.

Then the first step of Example 1.1.3 is not possible anymore, because clause (5) has been derived using the A of clause (1), while there is a B present in clause (1). It is possible to combine preference rules:

Never use an A if there is a B,
never use a B if there is a C,
never use C if there is not (C).

Every list of preference rules will be allowed, as long as it is consistent.

Consistency means that it is not allowed to have cyclic preferences. For example, the following is not possible: Prefer A over B, prefer B over C, and prefer C over A.

Definition 1.2.1 An acyclic list of preference rules is called an *ordering*. When it is applied to resolution, it is called an *ordering refinement* of resolution.

There is the following:

Theorem 1.2.2 If a set of clauses is inconsistent, then for every ordering (= acyclic list of preferences), when this ordering will be applied to resolution, a contradiction will be obtained.

Proofs of this can be found in, for example ([Reynlds66] and [KH69]).

Example 1.2.3 Consider the list of preferences:

A	1
B	2
not(A)	3
C	4
not(C)	5
not(B)	6

The item with the highest number is always preferred. We apply this list on the clauses of Example 1.1.3. We sort the clauses with the most preferred literal in front.

1. C or B or A
2. C or not(A) or B
3. not(B) or C
4. not (C)

Then it is possible to derive:

5. B or A

6. not(A) or B

After that:

7. C or A, and
8. A

Using this it is possible to derive

9. B and
10. C,

and a contradiction is obtained.

The relevance of these ordering refinements lies in the fact that they improve the efficiency of the proof search process tremendously.

1.3 Resolution Games

The resolution game is obtained by making the preferences variable. It is played by two players: One will try to derive a contradiction from a clause set, and is therefore called the *opponent* of the clause set. The other player decides what preferences the opponent must respect. The other player has the right to suddenly change his (or her) taste, and impose a different list of preferences.

He (or she) will try to prevent the derivation of the contradiction and is therefore called the *defender* of the clause set. For example it is possible to make 3 lists of preferences:

(1) $A, B, C, \text{not}(A), \text{not } B, \text{not}(C)$,
(2) $\text{not}(A), B, \text{not}(B), \text{not}(C), A, C$
(3) $\text{not}(B), B, \text{not}(A), A, C, \text{not}(C)$

(The lists are ordered in decreasing preference) The game begins with the first list of preferences. Then at any moment the defender can fancy to replace the list of preferences by a higher one. The following is proven in this thesis:

Theorem 1.3.1 If a list of clauses is inconsistent, then for every number of lists of preferences, the opponent can derive a contradiction.

1.4 Outline of this Thesis

This thesis is written as a report of the activities of the author during the last four years. It is for this reason that its contents is not selected completely thematically, but by the fact that the author has worked on it. As a consequence the contents of this thesis is not as coherent as it would have been, in the case that it would have been selected completely by subject.
It is for the same reason that the research which underlies this thesis is not complete. Especially concerning non-liftable orders (Chapter 7), there are many open questions, and things to be done.
The main contributions of this thesis (and the underlying research) to automated theorem proving are the resolution game, (Chapter 6) and the non-liftable ordering refinements (Chapter 7).
The completeness of resolution with liftable orderings was known for a long time. (According to ([Lovelnd78]), the idea of ordering atoms is from ([Reynlds66]). In ([Nivelle95]) the completeness of a large class of non-liftable orderings is proven. With this result it was possible to solve an open problem in ([FLTZ93]).
It is expected that non-liftable orderings can improve the efficiency of resolution a lot, and that with non-liftable orderings new decidable classes can be found, but the research on this is not finished.
This thesis is an elaboration of the following main insights:

1. Resolution calculi can be obtained in a standard way from semantic tableau calculi.

2. Moreover, it is possible to obtain ordering refinements of resolution with this method.

3. It is possible to prove the completeness of non-deterministic ordering refinements. This results in the resolution game.

In Chapter 2, some preliminaries are given, which will be used later in the thesis.
In Chapter 3 we will introduce some basic concepts of logic which we need in the rest of the thesis. We introduce first order languages, sequent calculi, semantic tableaux, and prove the completeness of resolution, and the standard refinements of resolution in classical logic.
Chapter 4 and the following contain our work. We show that it is possible to obtain resolution calculi in a standard way from semantic tableau calculi. It is also possible to obtain ordered resolution from semantic tableaux. We compare this method to another standard method, namely ([Mints88]),

which obtains resolution calculi from sequent calculi. The main difference is that the resolution proofs, resulting from the two methods have opposite directions. This is an important difference when the axioms that will be used in the proof are not known in advance.
In Chapter 5 we apply the method of Chapter 4 in order to obtain resolution calculi for modal logics. This is a useful field for the application of such a transformation, because there are many modal logics, and it is a tedious task to develop resolution methods for all of them.
In modal logic two additional logical operators are present: A is necessary, and A is possible. The exact meaning of 'necessary' and 'possible' is disputable. This gives rise to many systems of modal logics. We will develop resolution calculi for a large class of modal logics by constructing semantic tableau calculi for them in a general way, and then using the method of Chapter 4 to obtain the resolution methods.
In Chapter 6 we introduce the resolution game. Resolution games are non-deterministic ordering refinements. They can be used to model ordering refinements which are deterministic in nature, but too difficult to understand. In order to obtain a result as general as possible we define the resolution game in the context of (4). As a consequence every semantic tableau calculus can be translated into a resolution game. Because not every non-deterministic ordering refinement is complete, it is necessary to impose some control on the non-determinism. This can be done in different ways, and this gives rise to different types of resolution games.
Finally in Chapter 7 we will apply the resolution game to non-liftable orderings. With this technique many completeness results for non-liftable orderings can be obtained.
In the examples in Section 1.2, we did not consider predicate logic. In predicate logic the formulae X_i can have arguments, like for example in $X_i(t_1, \ldots, t_l)$. Then the problem is how to compare the X_i when they have different arguments. The standard solution is to make the ordering (almost) independent of the arguments. This is called *liftability*. This is a severe restriction on the possible orderings. For this reason completeness results on non-liftable orderings are very desirable.
It is mainly on this subject that more work is needed. We present many partial results on completeness of non-liftable orders, but the general problem of completeness of non-liftable orders remains unsolved. We also do not know how efficient non-liftable orderings are in practice, and how they are compatible with subsumption. Finally there also remains a lot of work on decision problems to be done.

Chapter 2

Preliminaries

In this chapter we give some preliminaries. The main intention is to introduce some notation. The reader can take the risk and skip this chapter if he is in a hurry.

2.1 Orders and Ordinal numbers

In this section we define the concepts of order, well-order, and ordinal number. An order is a relation that specifies how to line up the elements in a set:

Definition 2.1.1 An *order* $\prec$ is a relation with the following properties:

O1 Never $d \prec d$.

O2 If $d_1 \prec d_2$, and $d_2 \prec d_3$, then $d_1 \prec d_3$.

An order $\prec$ is called *total* on a set S if

O3 For all $d_1, d_2 \in S$, either $d_1 = d_2, d_1 \prec d_2$, or $d_2 \prec d_1$.

It is possible to obtain a relation $\preceq$ from $\prec$ as follows:
$d_1 \preceq d_2$ iff $d_1 \prec d_2$, or $d_1 = d_2$. This relation $\preceq$ satisfies:

O1' Always $d \preceq d$.

O2' If $d_1 \preceq d_2$, and $d_2 \preceq d_3$, then $d_1 \preceq d_3$.

O3' If $d_1 \preceq d_2$, and $d_2 \preceq d_1$, then $d_1 = d_2$.

It is also possible to reverse the process: If $\preceq$ is a relation that satisfies O1', O2', and O3', then a relation $\prec$ can be extracted by

$$d_1 \prec d_2 \text{ iff } d_1 \preceq d_2, \text{ and } d_1 \neq d_2.$$

It can be shown that $\prec$ satisfies O1 and O2. As a consequence we will call both $\prec$ and $\preceq$ an order, and use the one that is most convenient, dependent on the situation. In the same way we will use the pair $<$, $\leq$, and $\sqsubset$, $\sqsubseteq$.

Definition 2.1.2 If $\prec$ is an order, and D is a set, and $d \in D$, then d is *maximal* in D if for no $d' \in D$ it is the case that $d \prec d'$. d is minimal in D if for no $d' \in D$, it is the case that $d' \prec d$. If a minimal element is guaranteed to exist and unique, we will write $\text{Min}(D)$ for the minimal element of D. In the same way we write $\text{Max}(D)$ for the maximal element of D.

Minimal and maximal elements do not necessarily exist in a set D, and they are not necessarily unique if they exist. If however $\prec$ is total, then maximal and minimal elements are unique if they exist.
An order for which minimal elements do always exist, is called *well-founded*:

Definition 2.1.3 An order $\prec$ is *well-founded* on a set D, if every nonempty $X \subseteq D$ contains minimal elements. If $\prec$ is total, then $\prec$ is called a *well-order*.

It is possible to define a notion of number, which makes it possible to attach a length to each well-order. This type of number is called an *ordinal number*. The class of ordinal numbers has the following properties:

1. 0 is an ordinal number.
2. If α is an ordinal number, then $\alpha + 1$ is an ordinal number.
3. For every set S of ordinal numbers, there is an ordinal number α, which is the smallest ordinal greater than every element in S.

Definition 2.1.4 We call the class of ordinal numbers ORD. We use $<$ for the relation 'is less than'.

If X is a set of ordinals, then the smallest ordinal which is greater than every element in X, is called the *supremum* of X, which is written as $\text{Sup}(X)$.

The *maximal element* of X, notation $\text{Max}(X)$ is defined as the smallest ordinal n such that for all $x \in X$, we have $x \leq n$.

An ordinal is called a *successor ordinal* if it can be written as $\text{Sup}(\{n\})$, for an ordinal n. The other ordinals are called *limit ordinals*.

We have $\mathrm{Sup}(\emptyset) = 0, \mathrm{Sup}(\{n\} = n + 1$, $\mathrm{Sup}(\{0, 1, 2, 3, 4, \ldots\}) = \omega$. ω is a limit ordinal. $1, 2, 3, \ldots$, are successor ordinals.
If X is a finite set of ordinals, then $\mathrm{Max}(X) \in X$.

Definition 2.1.5 Let D be a set, let $\prec$ be a well-founded relation on D. We define the *rank* of d recursively as follows:

$$\mathrm{rank}(d) = \mathrm{Sup}(\{\mathrm{rank}(d') \mid d' \prec d\}).$$

We will often make recursive constructions using ordinal numbers: This is done as follows:

- Specify I_0.
- Specify how I_{i+1} can be obtained from I_i.
- Specify, for a limit ordinal i, how I_i can be obtained from the I_λ with $\lambda < i$.

Because every ordinal can be reached using these constructions, this will specify a value for each ordinal.

2.2 The Axiom of Choice

The axiom of choice is a controversial principle of set theory:

Definition 2.2.1 The axiom of choice (often abbreviated to AC) is the following principle: Let S be a set, such that the elements of S are nonempty sets. There exists a function f, with the property:
For all $s \in S$, it is the case that $f(s) \in s$.

The function f is called a *choice function*, because it chooses an element from each $s \in S$.
AC is controversial because it is difficult to specify a choice when S and the elements of the elements of S have no special properties which makes it possible to distinguish them. It is generally agreed that the application of the axiom of choice should be avoided where possible. In Section 3.5 we prove the completeness of Skolemisation, and take some effort to avoid an application of the axiom of choice.
With the axiom of choice the following can be proven:

Theorem 2.2.2 Every set has a well-order.

In fact the axiom of choice was first used in the proof of this theorem. In Chapter 6, we need the following generalisation of Theorem 2.2.2:

Theorem 2.2.3 Every well-founded order is included in a well-order.

Proof
Let D be a set, let $\prec$ we an arbitrary well-founded order on D. We construct a well-order $\prec'$, with $\prec \subseteq \prec'$. Let $\alpha = \mathrm{Sup}(\{\mathrm{rank}(d) \mid d \in D\})$. For λ, with $0 \leq \lambda < \alpha$, let D_λ be the elements in D with rank λ. By the well-ordering theorem, every D_λ has a well-order $\prec_\lambda$. Then define $d_1 \prec d_2$ if

1. $\mathrm{rank}(d_1) < \mathrm{rank}(d_2)$, or
2. $\mathrm{rank}(d_1) = \mathrm{rank}(d_2)$, and $d_1 \prec_\lambda d_2$, where $\lambda = \mathrm{rank}(d_1)$.

End of Proof

2.3 Trees

We use trees here to model choice sequences. They will be used for modelling games, or modelling the construction of an interpretation by choosing its truth values.
A tree is obtained as follows: Assume that there is an initial state. From this state a next state can be chosen and from each of these next states possibly another state can be chosen. If we add a condition, namely that when we have chosen for different directions, these direction are never going to meet again, the result will have a tree structure.
We will model a tree by a well-founded relation $<$. The minimal elements of $<$ are the roots of the tree. (So the choosing starts by first choosing a root) The set of successors of every node n are the nodes that can be chosen from n. The condition that paths never meet will be enforced by demanding that for every state n, the relation $<$ is total on the set $\{m \mid m < n\}$. This will ensure that there is only one path to n, because states from different paths are incomparable.

Definition 2.3.1 A *tree* is a partially ordered set $T = (N, <)$, which satisfies the following condition:
For every $n \in N$, the set $\{m \in N \mid m \leq n\}$ is well-ordered by $<$

- The elements of N are called *nodes* of N.
- The nodes n for which there are no $m \in N$, with $m < n$ are called the *roots* of N.

- A set $J \subseteq N$ is called *closed downward* if

$$n \in J \text{ implies for all } m \leq n : m \in J.$$

- Every set $J \subseteq N$, such that J is closed downward and totally ordered by $<$, is called an *initial segment* of a path. If J is a maximal such set then J is called *a path* of N. Every path is also an initial segment of a path.
- If J is an initial segment of a path, then the *tree of successor nodes* of J, written as $T(J)$, is defined as follows: $T(J) = (N(J), <)$ with

$$N(J) = \{n \in N \mid \forall j \in J, j < n\}.$$

  So $N(J)$ is the set of nodes of T that have to be accessed through J. If J is a path then $N(J) = \emptyset$. If $J = \emptyset$, then $T(J) = T$.
- The roots of $T(J)$ are called the *successors* of J.
- The *length* of an initial segment of a path is its ordinality. The *height* of a tree N is defined as

$$\text{Max}(\alpha \mid \alpha \text{ is the length of a path of } N).$$

- Sometimes we need a *labelled tree*. A labelled tree is a pair (T, Λ), where T is a tree, and Λ is a function which attaches an object to every node of T. We will also write $(N, <, \Lambda)$ for a labelled tree.

We define the process of *extending a path* of the tree. Let J be a path of T. We say that $T' = (N', <')$ is the result of extending J with $n_1, \ldots, n_k$ if

1. All n_i are not in N.
2. $N' = N \cup \{n_1, \ldots, n_k\}$.
3. $<' = < \cup \{(j, n_i) \mid j \in J, \text{ and } 1 \leq i \leq k\}$.

At some points our definition of tree is not completely standard. First it allows infinite paths in trees. Second the trees allow multiple roots. Standard is to use the nodes in N as states, and attach the choices to the branches. In that case there can be only one initial state, because initially no choice has been made yet. We think however that it is more natural to attach the choices to the nodes. In that case it is natural to have multiple roots since every node, including the roots, is the result of a choice.

We now prove a well-known fact, namely that every finitely branching tree, in which every path is finite, has a finite number of nodes: It is traditionally called Königs lemma, but it might be called a theorem as well.

Lemma 2.3.2 Let $T = (N, <)$ be a tree, such that

1. Every path in T is finite, and
2. Every initial segment of a path has only a finite number of successors.

Then N is finite.

Proof
Suppose that N is infinite. We define an infinite sequence of initial segments of a path

$$J_0 \subset J_1 \subset J_2 \subset \cdots \subset J_i \subset \cdots$$

Each J_{i+1} is obtained from J_i by adding a successor node. This will result in a contradiction because then $\bigcup J_i$ is an infinite path of T. Define $N(J_i)$ as the set of nodes of $T(J_i)$. We chose the successor nodes in such a manner that each $N(J_i)$ is infinite.

- Define $J_0 = \emptyset$. Because $T(\emptyset) = T$, the set $N(J_0)$ is infinite.
- In order to define J_{i+1} from J_i, let $n_1, \ldots, n_p$ be the successor nodes of J_i. It must be that p is finite. If all $N(J_i \cup \{n_j\})$ are finite (for $1 \leq j \leq p$), then $N(J_i) = \{n_1, \ldots, n_p\} \cup \bigcup_{j=1}^{p} N(J_i \cup \{n_j\})$ is finite. Because of this, one of the $N(J_i \cup \{n_j\})$ is not finite. Define $J_{i+1} = J_i \cup \{n_j\}$.

End of Proof

The following lemma can be seen as a generalisation of Lemma 2.3.2, because of the following:

Let T be a tree that satisfies the conditions of Königs lemma. If the tree is cut, (i.e. nodes are removed), then this process will end because there is only a finite number of nodes. Unfortunately this is not true for finitely branching trees with infinite paths, because they have infinitely many nodes. However we can speed up the cutting process by demanding that a cut is made in all branches simultaneously. In that case the process will end.

Definition 2.3.3 Let $T = (N, <)$ be a (possibly transfinite) tree. A list of sets of nodes $N_0, N_1, N_2, \ldots$, is a *fair cutting strategy* if:

- $N = N_0,\ N_0 \supset N_1,\ N_1 \supset N_2, \ldots$

- Each N_i is closed downward,
- For every path J of an N_i, there is an $j > i$, such that $(N_j, <)$ does not contain J anymore,

Lemma 2.3.4 Let $T = (N, <)$ be a (possibly transfinite) tree. Every fair cutting strategy $N_0, N_1, \ldots, N_i, \ldots$ is finite, and ends with $N_m = \emptyset$.

Proof
The proof is very strange. We construct a set J, such that

1. J is a path of one of the N_i, and
2. J is an initial segment of a path of all the N_j.

Let α be the height of T. We construct inductively a sequence $J_0, J_1, \ldots, J_\alpha$, such that each J_λ satisfies (2), and J_α, satisfies (1).

- Define $J_0 = \emptyset$. Clearly J_0 satisfies (2).
- For every successor ordinal λ, proceed as follows: Let $\{n_1, \ldots, n_p\}$ be the set of successor nodes of $J_{\lambda-1}$. This is a finite set because T is finitely branching. We distinguish 2 cases:
  1. For each n_j there is an N_{i_j}, where n_j is removed. Then there is an N_i in which all n_j are removed, because p is finite. Then the process ends with $J_\alpha = J_{\lambda-1}$.
  2. An n_j is never removed. Then put $J_\lambda = J_{\lambda-1} \cup \{n_j\}$. Because of the construction, J_λ satisfies (2).
- For every limit ordinal λ put

$$J_\lambda = \bigcup_{j \leq \lambda} J_j.$$

  Each J_j is in all of the N_i, and hence J_λ is in all of the N_i. So J_λ satisfies (2).

This construction will end with J_α satisfying (1) and (2). It must be the case that $J = \emptyset$, because of the assumption that every path is reduced. But then the sequence ends in a finite j.
End of Proof

2.4 Multisets

Multisets will be used in Chapter 6.

Definition 2.4.1 A multiset S is similar to a set, but it is able to distinguish how often an element occurs in it. The *characteristic function* χ_S is defined by:

- $\chi_S(x) = 0$ if x does not occur in S.
- $\chi_S(x) = n$ if x occurs n times in S.

We write [ ] for the empty multiset, $[1, 1, 2]$ is the multiset in which 1 occurs twice, 2 occurs once, and in which no other elements occur. The definition of maximal, and minimal element are the same as for normal (i.e. non multi-) sets. We define the following:

Definition 2.4.2 If S_1 and S_2 are multisets then

1. $S_1 \cup S_2$ is defined by
$$\chi_{S_1 \cup S_2}(x) = \chi_{S_1}(x) + \chi_{S_2}(x).$$
2. $S_1 \backslash S_2$ is defined by
$$\chi_{S_1 \backslash S_2}(x) = 0 \text{ iff } \chi_{S_1}(x) \leq \chi_{S_2}(x),$$
$$\chi_{S_1 \backslash S_2}(x) = \chi_{S_1}(x) - \chi_{S_2}(x), \text{ otherwise.}$$
3. $S_1 \subseteq S_2$ is defined by
$$\chi_{S_1}(x) \subseteq \chi_{S_2}(x) \text{ iff } \chi_{S_1}(x) \leq \chi_{S_2}(x), \text{ for all } x.$$
4. For any $n \in \mathcal{N}$, the set $n.S_1$ is defined by
$$\chi_{n.S_1}(x) = n.\chi_{S_1}(x).$$
5. S_1 and S_2 are disjoint if nowhere
$$\chi_{S_1}(x) \neq 0, \text{ and } \chi_{S_2}(x) \neq 0.$$

Chapter 3

Logic Preliminaries

In this chapter we introduce first order predicate logic. After that we prove some fundamental things about predicate logic, which are used in automated theorem proving. These are Herbrand theorem, and the possibility to Skolemise a formula. We end this chapter by defining unification, and resolution.

3.1 First order languages

Definition 3.1.1 We assume that we have an infinite set of *variables*. They will be written starting with uppercase letters. (e.g. $X, Y, Var1, X1, \dots$) There is a also a set of *constant symbols*. These will be written starting with a lowercase letter. We also consider numbers as constant symbols. Finally we assume that there is a set of *function symbols*. These will be written starting with a lowercase letter. We also consider the standard operator names, like $+$, and $\times$ as function symbols. Every function symbol has an integer attached to it, which is called the *arity* of the function symbol. The arity indicates the number of arguments that the function expects. We define the *set of terms* as the set of objects that can be constructed in finite time by the following rules:

- Each variable name is a term.
- Each constant symbol is a term.
- If $t_1, \dots, t_n$, with $n > 0$, are terms, and f is a function symbol, with arity n, then $f(t_1, \dots, t_n)$ is a term.

Terms will be used in first order logic, to indicate objects. Terms are the names of objects. If we have a proposition: 'The grass is green', 'the grass' functions as a term. Something is said about the grass, namely that it is green. We could have also said this about another term, for example 'the sky is green'. The last part of the sentence, '... is green' expects a term and will transform it into a sentence which is true, or not. Such a structure is called a *predicate*. There are also predicates which need two terms. For example: 'the grass is greener than the sky'. Here both 'the grass' and 'the sky' are terms, and '... is greener than ...' is a predicate which needs two terms. In the same manner as for a function, we will call the number of terms, that a predicate expects the *arity*.
Now if we have two propositions 'the grass is green', and 'the sky is blue', we might like to combine them: 'the grass is green and the sky is blue', or 'the grass is green or the sky is blue'. For a combination of the 'and'-type it is necessary that both components are true. For a proposition of the 'or'-type it is sufficient that one component is true. Whether or not such a combined proposition is true, can be obtained from the truth-values of the components. It is also possible to deny propositions: 'the grass is not green'. Such a denial is not true if the proposition is true. The following combinator is not so innocent: 'if the grass is green then the sky is blue'. This sentence somehow suggests a causal relation between the fact that the grass is green and the fact that the sky is blue. The problem with this causal relation is that it is extremely difficult to formalise, and that it doesn't fit well with the other operators. For this reason a large simplification has been made: A sentence of type 'if X then Y' is true if either X is not true, or Y is true. With this definition, the sentence above is true. Also true is the sentence 'if Paris is the capital of the Netherlands, then Amsterdam is the capital of France'. We do not claim that this last definition is the real meaning of 'if ... then ...'. It is only a convenient simplification.
However in some situations this definition catches the meaning of 'if ... then ...' quite well: The sentence 'if you want to live long you will have to eat healthy food' means that either you can accept a short life or healthy food.
There is another type of sentence that can be formed. If the grass green, then there exists a green thing. 'There is something which is green'. Its counterpart is: 'all things are green'. We will formalise this in the following:

Definition 3.1.2 We assume that we also have *predicate symbols*. These will be indicated by names, starting with a lowercase letter. Some will also be indicated by special symbols, like $=$, or $\leq$. We define formulae:

- If p is a predicate symbol, with arity $n \geq 0$, and $t_1, \ldots, t_n$ are terms, then $p(t_1, \ldots, t_n)$ is an atom.

Then:

- Every atom is a formula.
- If A is a formula then $\neg A$ is a formula.
- If A and B are formulae, then so are $A \vee B$, $A \wedge B$, $A \rightarrow B$.
- If A is a formula, and X is a variable, then $\forall X A$, and $\exists X A$, are formulae.

3.2 Semantics of first order logic

We define interpretations of first order formulae:

Definition 3.2.1 Let Γ be a set of first order formulae. An interpretation I of Γ is an ordered tuple $I = (D, [\,])$, where

- D is a nonempty domain,
- $[\,]$ is a function, which attaches
  - to every function symbol f in Γ, occurring with arity n, a total function $D^n \rightarrow D$. We write $[f]$ for this function.
  - to every predicate symbol p, occurring in Γ, with arity n a subset of D^n. These are the tuples for which the relation holds. We write $[p]$ for this subset.
  - to every variable V, which is free in an $F \in \Gamma$, an element of D. We will write $[V]$ for this element.

Now if we have an interpretation $I = (D, [\,])$ we can recursively attach a meaning to every term, when the variables are defined by $[\,]$:

- If the term is a variable V, then $[V]$ is already defined.
- For a constant symbol c, $[c]$ is already defined.
- If the term has form $f(t_1, \ldots, t_n)$, and f has arity $n > 0$, then $[f(t_1, \ldots, t_n)] = [f]([t_1], \ldots, [t_n])$.

We want to define when an interpretation makes a formula true. For this we sometimes need to change $[\,]$ a little: $[\,]_d^V$ is obtained from $[\,]$ by changing the value of V into d, if $[V]$ was defined, or adding it if $[V]$ was undefined:

1. $[\,]_d^V$ has the same value as $[\,]$, for all functions and constant symbols.

2. For all variables X different from V, $[X]_d^V$ is defined if and only if $[X]$ is defined. If $[X]_d^V$ and $[X]$ are both defined they have the same value.
3. $[V]_d^V = d$, independent of whether or not $[V]$ was defined.

Next we define when a formula is true or not in the interpretation. Let F be a formula, let I be an interpretation of F.

- If F is an atom, then F has form $F = p(t_1, \ldots, t_n)$. Here n is the arity of p. We define that $p(t_1, \ldots, t_n)$ is true in I if $([t_1], \ldots, [t_n]) \in [p]$. Otherwise $p(t_1, \ldots, t_n)$ is false in I.
- If F is of the form $\neg A$, then F is true in I if A is false in I. F is false in I if A is true in I.
- If F is of the form $A \vee B$ then F is true in I if either A is true, or B is true in I. Otherwise F is false in I.
- If F is of the form $A \wedge B$, then F is true in I if both A and B are true in I. Otherwise F is false in I.
- If F is of the form $A \rightarrow B$, then F is true if either A is not true in I, or B is true in I. Otherwise F is false in I.
- If F has form $\forall V A$, then F is true in $I = (D, [\ ])$ if A is true in $I' = (D, [\ ]_d^V)$, for every $d \in D$. Otherwise F is false in I.
- If F has form $\exists V A$, then F is true in $I = (D, [\ ])$ if A is true in $I' = (D, [\ ]_d^V)$, for a $d \in D$. Otherwise F is false in I.

We say that a formula (or a set of formulae) is *satisfiable* if there is an interpretation in which this formula, (or every formula in the set) is true.

3.3 A Sequent Calculus

In this section we give a *deduction system* for first order logic. The system that we define here is a sequent calculus. Sequent calculus is not completely standard. In most deduction systems formulae are added to a set, according to certain rules. If one wants to prove that Γ implies A, one starts with Γ, and tries to obtain the A, using the rules. For example there may be rules:

arrow If A and $A \rightarrow B$ are derived, then it is possible to derive B.

or From A derive $A \vee B$ and $B \vee A$.

and From A and B derive $A \wedge B$.

Sequent calculus works differently. Instead of deriving consequences, complete implications are derived. So the rule: From A and B the formula $A \wedge B$ can be derived, is replaced by the rule: If Γ implies A, and Γ implies B, then Γ implies $A \wedge B$. The first method has the disadvantage that it disturbs the symmetry of first order logic, by making a difference between assumption, and conclusion. In sequent calculus it is not necessary to make this distinction, and the symmetry of classical logic is preserved in the system. Before we can define sequent calculus, we need the following notions:

Definition 3.3.1 Let F_1 and F_2 be formulae. We call F_1 a *renaming* of F_2 if the following two things are the case:

1. F_1 and F_2 have the same structure. This means that if all variables in F_1 and F_2 are replaced by one variable, say X, then F_1 and F_2 are equal. This implies that F_1 and F_2 have the same structure, and that F_1 has a variable at a certain position if and only if F_2 has a variable at this position.
2. If X is a variable occurring in F_1 at a certain position, and Y occurs in F_2 at the same position, then either
   (a) X is free in F_1 and Y is free in F_2, and $X = Y$.
   (b) X and Y are both bound a quantifier. This quantifier must occur at the same position in both F_1 and F_2, and it must be universal or existential in both formulae.

This is a fairly complicated definition. We will consider formulae which are renamings of each other, equal.

Example 3.3.2 The following pairs of formulae are renamings of each other:

$p(X, Y)$	$p(X, Y)$
$\forall X \forall Y p(X, Y)$	$\forall Y \forall X p(Y, X)$
$\forall X \forall Y p(X)$	$\forall Y \forall Y p(Y)$

Definition 3.3.3 Let F be a formula, let X be a variable. A *free occurrence* of X in F, is a position in F which contains X, and which is not in the scope of a quantifier $\forall X$ or $\exists X$. Variable X is *free* in F if X has a free occurrence in F. These notions are well-defined because they are invariant under renaming. Let F be a formula, let X be a variable, let t be a term. The formula $F[X := t]$ is obtained by

1. selecting a renaming of F, for which each free occurrence of X is not in the scope of a quantifier which binds a variable occurring in t.
2. replacing all free occurrences of X by t in this renaming.

We will call this replacement *substitution*. The formula $F[X := t]$ is called a *substitution instance* of F, (or only instance)

It can be easily checked that when different renamings are chosen in (1), the results from the substitution will be renamings of each other.
Substitution will be used for the following: If one has proven a general thing, for example $\forall X \forall Y p(X, Y)$ then this implies $p(1, 2)$ and $p(s(s(X)), s(Y))$.

Definition 3.3.4 A *sequent* is a structure of the form

$$\Gamma \vdash \Delta.$$

The set Γ is a finite set of formulae, possibly empty. If $\Gamma = \emptyset$, we write $\vdash \Delta$. The set Δ is a finite set of formulae, possibly empty. If $\Delta = \emptyset$, we write $\Gamma \vdash$. If both Γ and Δ are empty, then we write $\vdash$.

We say that a sequent $\Gamma \vdash \Delta$ *holds* if every interpretation $I = (D, [\ ])$, of Γ, Δ, which makes all Γ true, makes one of the Δ true.

A sequent $\Gamma \vdash \Delta$ is true in an interpretation $I = (D, [\ ])$ if either

1. I does not make all Γ true, or
2. I makes one of the Δ true.

Next we give the replacement rules. The bars ——— mean that the sequent under the bar may be added if the sequents above the bar are present.

Definition 3.3.5 The rules for sequent calculus:

Axiom

$$\frac{}{A \vdash A}$$

Structural Rules

$$\text{Kl: } \frac{\Gamma \vdash \Delta}{\Gamma, A \vdash \Delta} \qquad \text{Kr: } \frac{\Gamma \vdash \Delta}{\Gamma \vdash \Delta, B}$$

Logical Rules

$$\wedge\text{l:}\ \frac{\Gamma, A, B \vdash \Delta}{\Gamma, A \wedge B \vdash \Delta} \qquad \wedge\text{r:}\ \frac{\Gamma \vdash \Delta, A \qquad \Gamma \vdash \Delta, B}{\Gamma \vdash \Delta, A \wedge B}$$

$$\vee\text{l:}\ \frac{\Gamma, A \vdash \Delta \qquad \Gamma, B \vdash \Delta}{\Gamma, A \vee B \vdash \Delta} \qquad \vee\text{r:}\ \frac{\Gamma \vdash \Delta, A, B}{\Gamma \vdash \Delta, A \vee B}$$

$$\rightarrow\text{l:}\ \frac{\Gamma \vdash \Delta, A \qquad \Gamma, B \vdash \Delta}{\Gamma, A \rightarrow B \vdash \Delta} \qquad \rightarrow\text{r:}\ \frac{\Gamma, A \vdash \Delta, B}{\Gamma \vdash \Delta, A \rightarrow B}$$

$$\neg\text{l:}\ \frac{\Gamma \vdash \Delta, A}{\Gamma, \neg\, A \vdash \Delta} \qquad \neg\text{r:}\ \frac{\Gamma, B \vdash \Delta}{\Gamma \vdash \Delta, \neg\, B}$$

Quantifier Rules

$$\forall\text{l:}\ \frac{\Gamma, A[X := t] \vdash \Delta}{\Gamma, \forall X A \vdash \Delta} \qquad \forall\text{r:}\ \frac{\Gamma \vdash \Delta, A}{\Gamma \vdash \Delta, \forall X A}$$

$$\exists\text{l:}\ \frac{\Gamma, A \vdash \Delta}{\Gamma, \exists X A \vdash \Delta} \qquad \exists\text{r:}\ \frac{\Gamma \vdash \Delta, A[X := t]}{\Gamma \vdash \Delta, \exists X A}$$

In the rules $\forall$r, and $\exists$l, there is the condition that X is not free in Γ or Δ.

Theorem 3.3.6 The sequent calculus given in Definition 3.3.5 is sound and complete. That means: A sequent $\Gamma \vdash \Delta$ is derivable if and only if it holds.

A proof can be found in ([Galllier86]).
In this sequent calculus the Γ and Δ of a sequent are sets, and this is very natural, because of the meaning of a sequent. There is no point in distinguishing how often a hypothesis is assumed, or how often a consequence is derived.
It is however possible to restrict this identification of the sequents $\Gamma \vdash \Delta, A, A$ and $\Gamma \vdash \Delta, A$. In a sequent $\Gamma \vdash \Delta$, it is possible to take Γ and Δ as multisets instead of sets, (or even lists). In that case some rules that are implicit when the Γ and Δ are sets, become explicit:

- From $\Gamma, A, A \vdash \Delta$ derive $\Gamma, A \vdash \Delta$,
- from $\Gamma \vdash \Delta, B, B$ derive $\Gamma \vdash \Delta, B$.

These rules are called *contraction rules*. Rules for the other direction are not necessary because they are captured by the rules Kl and Kr. In the case that Γ and Δ are lists, it is also meaningful to define a permutation rule.

These rules, which restructure sequents rather than derive them, are called *structural rules*. The weakening rule is also called a structural rule. Logic which have a sequent calculus in which these structural rules are in one or other form restricted are called *substructural logics*. An example of such a logic is *intuitionistic logic*, in which Γ is a set, and Δ contains at most one element. This makes contraction on the right impossible.
Another example is *linear logic*, the logic in which Γ and Δ are multisets, and where weakening and contraction are absent. It has two ands, and two ors. Unfortunately it is rather difficult to give a meaning to sequents of this logic, but it has many interesting proof theoretic properties.
There is one special rule which we did not mention here, the so called *cut rule*. It is the following rule:

Cut rule

$$\frac{\Gamma_1, A \vdash \Delta_1 \qquad \Gamma_2 \vdash \Delta_2, A}{\Gamma_1, \Gamma_2 \vdash \Delta_1, \Delta_2}$$

This rule is a redundant rule, because there exists a method to remove this rule from a proof if it is used. If this rule occurs in a proof it is possible to push it downward until it is no longer present. This process is called *cut elimination*. The cut rule has an undesirable property, which the other rules do not have, namely that the formula A completely dissapears. In all other rules, the formulae in the parent sequents are present as subformulae in the derived sequent. This property is called the *subformula* property.

3.4 Semantic Tableaux

Another proof method is the method of *semantic tableaux*. The method of semantic tableaux is strongly related to sequent calculus. There are the following 2 differences:

1. The Δ are written on the left side. So the sequent $\Gamma \vdash \Delta$ is replaced by $\Gamma, \neg\, \Delta \vdash \emptyset$.

2. The proof is written in a different way. Not as a tree of sequents, but as a tree of formulae.

Definition 3.4.1 *Semantic tableaux* are used as follows: If one wants to prove the sequent $\{A_1, \ldots, A_p\} \vdash \{B_1, \ldots, B_q\}$ then one will try to refute $\{A_1, \ldots, A_p, \neg\, B_1, \ldots, \neg\, B_q\}$.

This is done by constructing a tree, according to the following rules:

1. The construction starts with a linear tree, $(n_1, \ldots, n_{p+q})$, that is labelled with $A_1, \ldots, A_p, \neg B_1, \ldots, \neg B_q$.
2. It is possible to extend the tree as follows: Let J be a path of T.
   - If a node in J has label $A \wedge B$, then it is possible to extend J with a new node with label A.
   - If a node in J has label $A \wedge B$, then it is possible to extend J with a new node with label B.
   - If a node in J has label $A \vee B$, then it is possible to extend J in two directions. In one direction A is added as label, and in the other B is added as label. So, when this rule is applied, the path splits.
   - There are also the following rules: If a node in J contains the left hand side, then the path may be extended with a node, that is labelled with the right hand side.

     $$\begin{array}{lcl} A \rightarrow B & \Rightarrow & \neg A \vee B, \\ \neg \neg A & \Rightarrow & A, \\ \neg (A \vee B) & \Rightarrow & \neg A \wedge \neg B, \\ \neg (A \wedge B) & \Rightarrow & \neg A \vee \neg B, \\ \neg (A \rightarrow B) & \Rightarrow & A \wedge \neg B, \\ \neg \forall V A & \Rightarrow & \exists V \neg A, \\ \neg \exists V A & \Rightarrow & \forall V \neg A. \end{array}$$

   - If J contains a node that is labelled with $\forall X A$, then for every term t it is possible to extend J with a node with label $A[X := t]$.
   - If J contains a node that is labelled with $\exists X A$, then it is possible to extend J with a node with label A, on the condition that X is not free in a formula of J. Otherwise it is always possible to rename $\exists X A$.

   If a path J contains labels A and $\neg A$, then the path is *closed*. A semantic tree is *closed* if all its paths are closed. Otherwise it is *open*.

We then have the following:

Theorem 3.4.2 If $\{F_1, \ldots, F_n\}$ is a set of formulae, then there exists a closed semantic tableau, based on $\{F_1, \ldots, F_n\}$ if and only if $\{F_1, \ldots, F_n\}$ is unsatisfiable.

3.5 Skolemisation

In this section we define Skolemisation. This technique makes it possible to remove some quantifiers from a sequent that one wants to prove. Suppose that one wants to prove $\vdash \forall X p(X)$. Then it is sufficient to prove this sequent for an arbitrary X. It is possible to give this X a name c. In this manner the sequent can be replaced by $\vdash p(c)$. We will define Skolemisation in general. After that we prove that a sequent holds if and only if its Skolemisation holds.

Definition 3.5.1 We attach a *polarity* to all occurrences of (sub)formulae of a sequent. Let $\Gamma \vdash \Delta$ be a sequent. We say that all formulae in Γ occur *negatively* in the sequent, and that all formulae in Δ occur *positively* in the sequent. Furthermore:

- if a formula $A \vee B$, or $A \wedge B$ occurs positively in the sequent, then A and B occur positively in the sequent.
- if a formula $A \vee B$, or $A \wedge B$ occurs negatively in the sequent, then A and B occur negatively in the sequent.
- if a formula $A \rightarrow B$ occurs positively in the sequent then A occurs negatively in the sequent, and B occurs positively in the sequent.
- if a formula $A \rightarrow B$ occurs negatively in the sequent, then A occurs positively in the sequent, and B occurs negatively in the sequent.
- if a formula $\neg F$ occurs positively in the sequent, then F occurs negatively in the sequent.
- if a formula $\neg F$ occurs negatively in the sequent, then F occurs positively in the sequent.
- if a formula $\forall X F$, or $\exists X F$ occurs positively in the sequent, then F occurs positively in the sequent.
- if a formula $\forall X F$, or $\exists X F$ occurs negatively in the sequent, then F occurs negatively in the sequent.

This makes it possible to attach a polarity to all subformulae. (on a given position in a sequent)

Using this we can define Skolemisation.

Definition 3.5.2 Let $\Gamma \vdash \Delta$ be a sequent. A *Skolemisation* of $\Gamma \vdash \Delta$ is obtained by iteratively:

1. Selecting an outermost positive occurrence of a $\forall$-quantifier, or an outermost negative occurrence of an $\exists$-quantifier, (outermost means: Not in the scope of another positively occurring $\forall$, or negatively occurring $\exists$) and writing $F[QXA]$, for the formula in which the quantifier was found.
2. Collecting all variables in A that are bound by a quantifier (which must be a positively occurring $\exists$, or negatively occurring $\forall$) outside A. Write $V_1, \ldots, V_n$ for all these variables. It is possible that $n = 0$.
3. Replacing all occurrences of X by $f(V_1, \ldots, V_n)$, where f is a new function symbol which did not occur in $F[A]$ before,

This is done until no such replacements are possible. The functions f are called the *Skolem functions*. The terms $f(V_1, \ldots, V_n)$ are called *Skolem terms*.

Skolemisations are independent of renamings.

Theorem 3.5.3 Let $\Gamma \vdash \Delta$ be a sequent. Let $\Gamma' \vdash \Delta'$ be its Skolemisation. We have: $\Gamma \vdash \Delta$ holds iff $\Gamma' \vdash \Delta'$ holds.

The proof is far from easy. We give only a sketch.
If $\Gamma \vdash \Delta$ is a sequent then it is possible to find a renaming such that all quantifiers have a different variable. In the following it is assumed that this is done in a fixed manner. The variables that will be replaced by Skolem terms will be denoted as $W_1, \ldots, W_s$. The chosen Skolem terms will be $f_1(V_{1,1}, \ldots, V_{1,l_1}), \ldots, f_s(V_{s,1}, \ldots, V_{s,l_s})$. It is also assumed that these are fixed.
With $l_1, \ldots, l_s$ we will denote the arities of the Skolem functions. We will adopt some more notation: If $(D, [\,])$ is an interpretation of $\Gamma \vdash \Delta$, we write $[\,](d_{i,1}, \ldots d_{i,l_i})$ instead of $(\cdots([\,]_{d_{i,1}}^{V_{i,1}})_{d_{i,2}}^{V_{i,2}} \cdots)_{d_{i,l_i}}^{V_{l_i}}$, for obvious reasons. We write $[\,](d_{i,1}, \ldots, d_{i,l_i}; d)$ instead of $[\,](d_{i,1}, \ldots, d_{i,l_i})_d^{W_i}$.

Definition 3.5.4 Let $\Gamma \vdash \Delta$ be a sequent. Let $I = (D, [\,])$ be an interpretation of $\Gamma \vdash \Delta$. We define *Skolem relations* belonging to $\Gamma \vdash \Delta$ and I as relations $P_1, \ldots P_s$ on D with arities $l_1 + 1, \ldots, l_s + 1$. The i-th Skolem relation belongs to the i-th Skolem function. They must satisfy the following condition:

- If P_i belongs to a formula QW_iA, and the Skolem term belonging to W_i is $f_i(V_{i,1}, \ldots, V_{i,l_i})$, then: For every $d_{i,1}, \ldots, d_{l_i} \in D$, for every x and y, such that $P_i(d_{i,1}, \ldots, d_{l_i}, x)$ and $P_i(d_{i,1}, \ldots, d_{l_i}, y)$ it must be that

  - D and $[\ ](d_{i,1}, \ldots, d_{i,l_i}; x)$ makes A true iff
  - D and $[\ ](d_{i,1}, \ldots, d_{i,l_i}; y)$ makes A true.

  This definition is meaningful, because the truth value of A is determined by D and $[\ ](d_{i,1}, \ldots, d_{i,l_i}; x)$.

- For all $d_{i,1}, \ldots, d_{i,l_i} \in D$, there exists at least one $d \in D$, such that $P_i(d_{i,1}, \ldots, d_{i,l_i}, d)$.

Definition 3.5.5 Let $\Gamma \vdash \Delta$ be a sequent. Let $I = (D, [\ ])$ be an interpretation of $\Gamma \vdash \Delta$. Let $P_1, \ldots, P_s$ be Skolem relations. We define when *I makes a formula true with $P_1, \ldots, P_s$*. It is defined by cases, like in Definition 3.2.1. All cases are the same, except the following:

- if F has form $\forall W_i A$, and F is positively occurring in $\Gamma \vdash \Delta$, then F is true in I with $P_1, \ldots, P_s$ if A is true in every $I' = (D, [\ ]_d^V)$, such that $P_i(d_{i,1}, \ldots, d_{i,l_i}, d)$ Here the $d_{i,j}$ are the assignments to the $V_{i,j}$, so $d_{i,1} = [V_{i,1}], \ldots, d_{i,l_i} = [V_{i,l_i}]$. Otherwise F is false in I with $P_1, \ldots, P_s$.
- if F has form $\exists W_i A$, and F is negatively occurring in $\Gamma \vdash \Delta$, then F is true in I with $P_1, \ldots, P_s$ if there is a $d \in D$, such that $P_i(d_{i,1}, \ldots, d_{i,l_i}, d)$, and $I' = (D, [\ ]_d^{W_i})$ makes A true. Here $d_{i,1} = [V_{i,1}], \ldots, d_{i,l_i} = [V_{i,l_i}]$.
- If F is of the form $\forall V A$, and negatively occurring, or F is of the form $\exists V A$, and positively occurring, then nothing changes.

Definition 3.5.6 Let $\Gamma \vdash \Delta$ be a sequent. Let I be an interpretation of $\Gamma \vdash \Delta$. The *unrestricted Skolem relations* $P_1, \ldots, P_s$ are obtained as follows:

- For every F of the form $\forall W_i A$ that positively occurs in $\Gamma \vdash \Delta$,
  - If F is true in $(D, [\ ](d_{i,1}, \ldots, d_{i,l_i}))$, then for all d,

    $$P_i(d_{i,1}, \ldots, d_{i,l_i}, d).$$

  - If F is false in $(D, [\ ](d_{i,1}, \ldots, d_{i,l_i}))$, then

    $$P_i(d_{i,1}, \ldots, d_{i,l_i}, d),$$

    for all $d \in D$, such that A is false in $(D, [\ ](d_{i,1}, \ldots, d_{i,l_i}; d))$.

- Similarly for every F of the form $\exists W_i A$ that occurs negatively in $\Gamma \vdash \Delta$,
  - If F is true in $(D, [\,](d_{i,1}, \ldots, d_{i,l_i})$, then
    $$P_i(d_{i,1}, \ldots, d_{i,l_i}, d),$$
    for all $d \in D$, such that A is true in $(D, [\,](d_{i,1}, \ldots, d_{i,l_i}; d)$.
  - If F is false in $(D, [\,](d_{i,1}, \ldots, d_{i,l_i}; d)$, then, for all $d \in D$,
    $$P_i(d_{i,1}, \ldots, d_{i,l_i}, d).$$

Lemma 3.5.7

$$\Gamma \vdash \Delta \text{ is true in } I \text{ iff}$$

$$\Gamma \vdash \Delta \text{ is true in } I \text{ with } P_1, \ldots, P_s,$$

if $P_1, \ldots, P_s$ are the unrestricted Skolem relations based on the sequent $\Gamma \vdash \Delta$ and the interpretation I.

The proof can be obtained by induction following Definitions 3.2.1 and 3.5.5.

Lemma 3.5.8 Let $\Gamma \vdash \Delta$ be a sequent. Let $W_1, \ldots, W_s$ be the variables to be Skolemised. Let $I = (D, [\,])$ be an interpretation of $\Gamma \vdash \Delta$. Let $P_1, \ldots, P_s$ be a set of Skolem relations.
A *restriction* of D and the P_i is a domain $\overline{D}$, and Skolem relations $\overline{P}_i$ satisfying the following:

1. $\overline{D} \subseteq D$,
2. if $d_1, \ldots, d_{l_i} \in \overline{D}$, then
   $$\overline{P}_i(d_1, \ldots, d_{l_i}, x) \Rightarrow P_i(d_1, \ldots, d_{l_i}, x).$$

The following is the case:

$$\Gamma \vdash \Delta \text{ is true in } I \text{ with } P_1, \ldots, P_s \text{ iff}$$

$$\Gamma \vdash \Delta \text{ is true in every restriction } \overline{I} = (\overline{D}, [\,]) \text{ with } \overline{P}_1, \ldots, \overline{P}_s.$$

The proof can be obtained by induction on the structure of the formulae.
We can now combine 3.5.6 and 3.5.8 into:

Theorem 3.5.9 Let $\Gamma \vdash \Delta$ be a sequent, Then: $\Gamma \vdash \Delta$ is true in every interpretation I iff for every interpretation $I = (D, [\,])$, for every sequence of Skolem relations $P_1, \ldots, P_s$, $\Gamma \vdash \Delta$ is true in I with $P_1, \ldots, P_s$.

This is because every sequence of Skolem relations $P_1, \ldots, P_s$ is a restriction of the unrestricted Skolem relations.

Definition 3.5.10 Let $I = (D, [\,])$ be an interpretation of $\Gamma \vdash \Delta$, let $P_1, \ldots, P_s$ be a set of Skolem relations. $P_1, \ldots, P_s$ are called *functional* if

- Whenever for $d_1, \ldots, d_{l_i} \in D$, both $P_i(d_1, \ldots, d_{l_i}, x)$ and $P_i(d_1, \ldots, d_{l_i}, y)$ hold, then $x = y$.

Now the following is the case:

Lemma 3.5.11 Let $\Gamma \vdash \Delta$ be a sequent, then $\Gamma \vdash \Delta$ is true in every interpretation $I = (D, [\,])$ with functional $P_1, \ldots, P_s$, iff the skolemisation $\Gamma' \vdash \Delta'$ is true in every interpretation of $\Gamma' \vdash \Delta'$.

This is obtained by interpreting the Skolem functions as the functional Skolem relations.
We are almost finished now. It remains to show the following:

Lemma 3.5.12 Let $D = (I, [\,])$ be an interpretation with Skolem relations $P_1, \ldots, P_s$. There exists a restriction $\overline{D}$, with functional $\overline{P}_1, \ldots, \overline{P}_s$.

This completes the proof of Theorem 3.5.3.

3.6 Theorem of Herbrand

In this section we state what is usually called Herbrand's theorem. A sequent holds iff and only iff it possible to replace all formulae in the sequent by a finite number of instances, such that the resulting sequent holds.
According to ([Galllier86]), this is not the orginal theorem of Herbrand, because the original theorem was stated in terms of *provability* instead of validity. However we will stick to the tradition and prove Herbrand's theorem, which is not from Herbrand:
First we define what an instance is:

Definition 3.6.1 Let F be a formula. An *instance* of F is obtained by making substitutions for all quantifiers in F. So all QXA in F are replaced by an $A[X := t]$.

For example $p(a, b) \vee q(a)$ is an instance of $\forall X(\forall Y p(X, Y)) \vee q(X))$. Now we can state the theorem:

Theorem 3.6.2 Let $\Gamma \vdash \Delta$ be a sequent with no negatively occurring $\exists$-quantifiers and no positively occurring $\forall$-quantifiers (typically obtained by Skolemisation). The following two are equivalent:

1. $\Gamma \vdash \Delta$ holds.
2. There is a $\Gamma' \vdash \Delta'$, which is obtained by replacing all formulae in $\Gamma \vdash \Delta$ by a finite (possibly 0) number of instances and this $\Gamma' \vdash \Delta'$ holds.

Before we can give a proof we need some preparation.

Definition 3.6.3 Let $\Gamma \vdash \Delta$ be a sequent with no negatively occurring $\exists$-quantifiers, and no positively occurring $\forall$-quantifiers. A *Herbrand interpretation* of $\Gamma \vdash \Delta$ is an interpretation $I = (D, [\,])$, with

- D is the set of all terms that can be made from all variables, constants and function symbols in $\Gamma \vdash \Delta$.
- $[\,]$ is defined in such a manner that $[t] = t$, for all terms. So, for all variables that occur in $\Gamma \vdash \Delta$,

  $$[X] = X.$$

  For all constants c, that occur in $\Gamma \vdash \Delta$,

  $$[c] = c,$$

  For all functions f, that occur in $\Gamma \vdash \Delta$, (with arity n), $[f]$ is defined from:

  $$[f](t_1, \ldots, t_n) = f(t_1, \ldots, t_n).$$

- For the predicate symbols p in $\Gamma \vdash \Delta$, there are no conditions on $[p]$.

All Herbrand interpretations have the same domain. They only differ in the interpretations of the predicate symbols. As a consequence a Herbrand interpretation is completely determined by the interpretations of the predicate symbols and it is possible to describe a Herbrand interpretation by summing up the literals which are true in it:

Definition 3.6.4 Let $\Gamma \vdash \Delta$ be a sequent. The *Herbrand universe* of $\Gamma \vdash \Delta$ is the set of atoms that can be constructed from the predicate symbols, the function and constant symbols and the variables in $\Gamma \vdash \Delta$.

Let $I = (D, [\,])$ be a Herbrand interpretation of a sequent $\Gamma \vdash \Delta$. We define $\mathrm{HB}(I) =$

$$\{p(t_1, \ldots, t_n) \mid p \text{ occurs in } \Gamma \vdash \Delta \text{ with arity } n, \text{ and } (t_1, \ldots, t_n) \in [p]\} \cup$$
$$\{\neg\, p(t_1, \ldots, t_n) \mid p \text{ occurs in } \Gamma \vdash \Delta \text{ with arity } n, \text{ and } (t_1, \ldots, t_n) \notin [p]\}.$$

A Herbrand interpretation is completely determined by the set HB(I).

Lemma 3.6.5 Let $\Gamma \vdash \Delta$ be a sequent with no negatively occurring $\exists$-quantifiers, and no positively occurring $\forall$-quantifiers. Let I be a Herbrand interpretation of $\Gamma \vdash \Delta$. Then $\Gamma \vdash \Delta$ is true in I if there is a $\Gamma' \vdash \Delta'$ obtained by replacing each formula in Γ' or Δ', by *one* instance which holds in I.

Lemma 3.6.6 Let $\Gamma \vdash \Delta$ be a sequent with no negatively occurring $\exists$-quantifiers, and no positively occurring $\forall$-quantifiers. $\Gamma \vdash \Delta$ holds in an interpretation iff $\Gamma \vdash \Delta$ holds in a Herbrand interpretation.

3.7 Unification

In Definition 3.3.3 substitution was defined. Here we will look for minimal substitutions which make two terms equal.

If A is an atom then $A[X := t]$ is the result of replacing all free occurrences of X by t. In this notation the substitution is not considered as an independent object. The notation $A[X := t]$ is just a nice notation for a function $S(A, X, t)$. In this section, it is necessary to treat the substitution as an independent object, because, in order to define the minimal substitution, it is necessary to quantify over substitutions.
In the rest of this section we will not distinguish atoms and terms, because they have the same syntactical structure, and they behave the same under substitution. We will speak about terms, but we also mean atoms.

Definition 3.7.1 A substitution is a finite set of assignments of the following form:

$$\Theta = \{V_1 := t_1, \ldots, V_n := t_n\}.$$

It is not allowed that there are $V_i := t_i$, with $V_i = t_i$, because these are redundant. It must be the case that for no different $V_i := t_i$, and $V_j := t_j$, we have $V_i = V_j$. The intended meaning of a substitution is that in a term every occurrence of V_i will be replaced by t_i. We write $A\Theta$ for the application of Θ on A.

Definition 3.7.2 Let

$$\Sigma = \{V_1 := t_1, \ldots, V_n := t_n\}, \text{ and}$$

$$\Theta = \{W_1 := u_1, \ldots, W_m := u_m\}$$

be substitutions. The composition of Σ and Θ is defined as the set:

$$\Sigma \cdot \Theta = \{V := (V\Sigma)\Theta \mid V \neq (V\Sigma)\Theta\}.$$

The composition is always finite because it will contain not more than $n+m$ elements. It can be easily computed.

Lemma 3.7.3 For every term A,

$$A(\Sigma \cdot \Theta) = (A\Sigma)\Theta.$$

This is because it is true for every variable variable, by definition.
Composition of substitution is associative:

Lemma 3.7.4 For all substitutions Σ, Θ, and Ξ,

$$(\Sigma \cdot \Theta) \cdot \Xi = \Sigma \cdot (\Theta \cdot \Xi).$$

It is easily checked for each variable in Σ, Θ, or Ξ that this is true.
Now we define the most general unifier:

Definition 3.7.5 A substitution Θ which makes two terms equal is called a *unifier*. Let A and B be two terms. A unifier Θ of A and B is called a *most general unifier* of A and B if for every unifier Σ, there is a substitution Ξ, such that $\Sigma = \Theta \cdot \Xi$.

Let $A_1 = B_1, \ldots, A_n = B_n$ be a list of equations. A unifier of $A_1 = B_1, \ldots, A_n = B_n$ is a substitution Θ, for which $A_1\Theta = B_1\Theta, \ldots, A_n\Theta = B_n\Theta$. Θ is a most general unifier if for every unifier Σ of $A_1 = B_1, \ldots, A_n = B_n$, there is a Ξ, such that Σ can be written as $\Sigma = \Theta \cdot \Xi$. The term 'most general unifier' is often abbreviated as *mgu*.

We will also call the result $A\Theta$ an mgu, when Θ is an mgu of A and B.

We will call a substitution Θ a *renaming substitution* if

1. For every variable V, the result $V\Theta$ is a variable,
2. For no two different variables V_1 and V_2 is it possible that $V_1\Theta = V_2\Theta$.

It can be proven that the mgu is unique, up to renaming.
We will now prove the following important fact, due to ([Robins65]).

Theorem 3.7.6 Let $A_1 = B_1, \ldots, A_n = B_n$ be a set of equations. The following procedure determines whether or not the set of equations has an mgu. The procedure also determines the mgu in the case that it exists. The algorithm maintains a state vector $(A_1 = B_1, \ldots, A_n = B_n, \Theta)$, where $A_1 = B_1, \ldots, A_n = B_n$ is a list of equations, and Θ is a substitution.

Begin Conditions

Suppose that one wants to compute the mgu of the list of equations $A_1 = B_1, \ldots, A_n = B_n$. Begin with the state vector $(A_1 = B_1, \ldots, A_n = B_n, \emptyset)$.

Replacement Conditions:

Write the state vector as $(A_1 = B_1, \ldots, A_n = B_n, \Theta)$. Then:

1. If one of the equations in the state vector has form $V = V$, for a variable V, then this equation can be deleted without affecting the other equations or Θ.
2. Any equation of the form $f(t_1, \ldots, t_n) = f(u_1, \ldots, u_n)$ can be replaced by the equations $t_1 = u_1, \ldots, t_n = u_n$, without affecting Θ. Note that if $n = 0$, then the equation is replaced by nothing.
3. An equation of the form $V = t$, or $t = V$, where V does not occur in t and $V \neq t$, can be removed, but all other equations $A_i = B_i$ have to be replaced by $A_i\{V := t\} = B_i\{V := t\}$, and Θ has to be replaced by $\Theta \cdot \{V := t\}$.

End Conditions:

Write the state vector as $(A_1 = B_1, \ldots, A_n = B_n, \Theta)$. Then:

1. If one of the equations has form $f(t_1, \ldots, t_n) = g(u_1, \ldots, u_m)$, where either $f \neq g$, or $n \neq m$, then report that the attempt to find an mgu has failed.
2. If one of the equations has form $V = t$, or $t = V$, where V does occur in t, but $V \neq t$, then report that the attempt to find an mgu has failed.
3. If there are no equations left, then the attempt to construct an mgu is successfull. Θ equals the mgu.

We will prove that this algorithm is correct and terminating.
First we prove that it terminates. In order to do this we need the complexity of a term:

Definition 3.7.7 The *complexity* of a term t, which will be written as $\#t$, is recursively defined as follows:

1. For a variable V,
$$\#V = 1.$$
2. For a term of the form $f(t_1, \ldots, t_n)$, (possibly $n = 0$),
$$\#f(t_1, \ldots, t_n) = 1 + \#t_1 + \cdots + \#t_n.$$

The complexity of a list of equations $A_1 = B_1, \ldots, A_n = B_n$ equals $\#A_1 + \#B_1 + \cdots + \#A_n + \#B_n$.

In order to see that the algorithm terminates, note that every replacement condition does one of the following two things:

1. It decreases the number of variables in the equations.
2. It does not change the number of variables in the equations, but it reduces the total complexity of the equations.

(In case 3 the total number of variables decreases, because V does not occur in t. Checking the other replacement conditions is trivial). Because of this only a finite number of replacements will be made in the equations and the algorithm ends.
In order to prove the correctness of the algoritm we give the invariant of the algorithm:

INV1 Let (E, Θ), be a state vector that is constructed by the algorithm of Theorem 3.7.6. If the initial set of equations has a unifier, then whenever Σ is an mgu of E, the composition $\Theta \cdot \Sigma$ is an mgu of the initial set of equations. If Ξ is a unifier of the initial set of equations, then Ξ can be written as $\Theta \cdot \Sigma$.

INV2 If the initial set of equations does not have a unifier, then the algorithm will not enter a state (E, Θ), in which E has a unifier.

INV1 will ensure that the algorithm will give an mgu if one exists. INV2 ensures that if the algorithm gives an mgu, that then there exists a unifier of the initial set of equations.
We will prove the correctness of the procedure by proving that the replacements steps preserve the invariants. First some preparation:

Lemma 3.7.8 For a single equation $V = t$, the following holds:

1. If $V \neq t$, and V occurs in t, then for no substitution Σ it is possible that $V\Sigma = t\Sigma$.
2. If $V = t$, the equation $V = t$ has mgu $\emptyset$.

3. If $V \neq t$, and V does not occur in t, then $V = t$ has mgu $\{V := t\}$.

Proof

1. Because V occurs strictly in t, it will be the case that $V\Sigma$ occurs strictly in $t\Sigma$. No term can be equal to a strict subterm of itself.

2. If $V = t$, then obviously $V\emptyset = t\emptyset$. Because every substitution can be written as $\Sigma = \emptyset \cdot \Sigma$, the empty substitution is the mgu.

3. $V\{V := t\} = t$, and $t\{V := t\} = t$, because V does not occur in t. Hence $\{V := t\}$ is a unifier. Now assume that $V\Sigma = t\Sigma$. Then $\Sigma = \{V := t\} \cdot \Sigma$. This is because of the following: If $V \neq W$, then $W\{V := t\} = W$, and so $W\Sigma = W(\{V := t\} \cdot \Sigma)$. For V holds $V(\{V := t\} \cdot \Sigma) = t\Sigma = V\Sigma$.

End of Proof

Lemma 3.7.9 The equation $f(t_1, \ldots, t_n) = g(u_1, \ldots, u_m)$ with $f \neq g$, or $n \neq m$, has no unifier.

Proof
Let Σ be a substitution. Then:

$$f(t_1, \ldots, t_n)\Sigma = f(t_1\Sigma, \ldots, t_n\Sigma), \text{ and}$$

$$g(u_1, \ldots, u_m)\Sigma = g(u_1\Sigma, \ldots, u_m\Sigma).$$

Still $f \neq g$, or $n \neq m$, and the terms are not equal...
End of Proof

Lemma 3.7.10 Substitution Σ is a unifier of

$$f(t_1, \ldots, t_n) = f(u_1, \ldots, u_n) \text{ iff}$$

Σ is a unifier of

$$t_1 = u_1, \ldots, t_n = u_n.$$

Proof
Because

$$f(t_1, \ldots, t_n)\Sigma = f(t_1\Sigma, \ldots, t_n\Sigma),$$

$$f(u_1, \ldots, u_n)\Sigma = f(u_1\Sigma, \ldots, u_n\Sigma), \text{ and}$$

$$f(t_1\Sigma, \ldots, t_n\Sigma) = f(u_1\Sigma, \ldots, u_n\Sigma) \text{ iff}$$
$$t_1\Sigma = u_1\Sigma, \ldots, t_n\Sigma = u_n\Sigma.$$

End of Proof

Now we are ready to prove the correctness of the algorithm: We first prove that INV1 is preserved. During the proof the state vector is written as (E, Θ).

- We show that the invariant is produced by the initial state. Initially E is the initial set of equations, and $\Theta = \emptyset$. If the the initial set E has a unifier, and Θ is an mgu of E, then obviously $\emptyset \cdot \Theta$ equals the mgu.
- Let (E', Θ) be obtained from (E, Θ) by the first or second replacement rule. We prove that INV1$(E, \Theta) \Rightarrow$ INV1(E', Θ). In both cases we have Σ is unifier of E' iff Σ is a unifier of E. (In the second case by Lemma 3.7.10). As a consequence E and E' have the same set of mgu's.
- Let $(E\{V := t\}, \Theta \cdot \{V := t\})$ be obtained from $(E \cup \{V = t\}, \Theta)$ by an application of the third replacement rule. We prove INV1$(E\{V := t\}, \Theta \cdot \{V := t\})$ from INV1$(E \cup \{V = t\}, \Theta)$. Assume that the initial set of equations has a unifier. Let Σ be an mgu of $E\{V := t\}$. We have to show that
  - $(\Theta \cdot \{V := t\}) \cdot \Sigma$ is an mgu of the initial set of equations.

  By associativity of $\cdot$,

  $$(\Theta \cdot \{V := t\}) \cdot \Sigma = \Theta \cdot (\{V := t\} \cdot \Sigma).$$

  We are finished if we manage to prove that $\{V := t\} \cdot \Sigma$ is an mgu of $E \cup \{V = t\}$, because then we can use INV1$(E \cup \{V = t\}, \Theta)$.
  By Lemma 3.7.8, $\{V := t\}$ is an mgu of $V = t$. Then
  $E \cup \{V = t\}(\{V := t\} \cdot \Sigma) = (E\{V := t\} \cup \{t = t\})\Sigma$. As a consequence $\{V := t\} \cdot \Sigma$ unifies $E \cup \{V = t\}$.
  Now assume that Ξ is a unifier of $E \cup \{V = t\}$. We want to write $\Xi = (\{V := t\} \cdot \Sigma) \cdot \Xi'$. Because Ξ must unify $V = t$, it must be the case that Ξ can be written as $\Xi = \{V := t\} \cdot \Phi$. Φ must be a unifier of $E\{V := t\}$, and hance Φ can be written as $\Sigma \cdot \Xi'$. We are finished now: Write

  $$\Xi = \{V := t\} \cdot (\Sigma \cdot \Xi') = (\{V := t\} \cdot \Sigma) \cdot \Xi'.$$

- In the first and second end condition. There is an $A = B \in E$, that has no unifier. Assume that the initial set of equations has a unifier. Then it must be possible to write the mgu as $\Theta \cdot \Sigma$. This is impossible because E has no unifier. Therefore the initial set of equations cannot have a unifier.

- It remains to show that in the third end conditions, Θ is the mgu.

We now prove that if the initial set of equations has no unifier, the algorithm will not enter a state (E, Θ), where E has a unifier.

- If the initial set of equations does not have a unifier, then in the initial state (E, Θ), obviously E does not have a unifier.

- If (E', Θ) is obtained from (E, Θ) by the first or second replacement rule, then E has a unifier iff E' has a unifier. Therefore E' has no unifier.

- Let $(E\{V := t\}, \Theta \cdot \{V := t\})$ be obtained from $(E \cup \{V = t\}, \Theta)$. If $E\{V := t\}$ has a unifier Σ, then certainly $\{V := t\} \cdot \Sigma$ is a unifier of $E \cup \{V = t\}$, by Lemma 3.7.8.

We have now proven the correctness of the algorithm of Theorem 3.7.6.

Sometimes it is better to use another algorithm. The following algorithm is better suited for proving properties of unification than the algorithm of Theorem 3.7.6. It is not so efficient, but that is not what it is designed for.

Theorem 3.7.11 Let A and B be two terms, that have an mgu. The following procedure will obtain an mgu. The mgu will be obtained as a concatenation of the following form: $\Theta = \Sigma_1 \cdot \ldots \cdot \Sigma_n$. The procedure begins with $\Theta_0 = \{\}$. When Θ_i is defined as $\Sigma_1 \cdot \ldots \cdot \Sigma_i$, the procedure will construct another Σ_{i+1} as long as $A\Theta_i$ and $B\Theta_i$ are not equal.

This is done as follows:

- Put $T_1 = A\Theta_i$, and $T_2 = B\Theta_i$. Then as long as long as neither T_1, nor T_2 is a variable do:

  - If $T_1 = f(t_1, \ldots, t_n)$ $T_2 = f(u_1, \ldots, u_n)$, and $T_1 \neq T_2$, then one pair t_i, u_i must be unequal. Replace T_1 by t_i, and replace T_2 by u_i.

  Now, either T_1 or T_2 is a variable.

  1. If T_1 is a variable, then put $\Sigma_i = \{T_1 := T_2\}$.
  2. If T_2 is a variable, then put $\Sigma_i = \{T_2 := T_1\}$.

The Σ_i are called *mesh substituents*.

A computation of this algorithm can be transformed into a computation of the algorithm in Theorem 3.7.6. It is not necessary to include a test because the procedure assumes that there exists an mgu.

3.8 Resolution Theorem Proving

In this section we define resolution theorem proving. We begin by defining the normal form that is used, and then we will introduce the resolution rule. After that we prove the soundness and completeness of resolution.

Definition 3.8.1 A *clause* is a finite set of literals. The *meaning* of $\{A_1, \ldots, A_p\}$ is $\forall \overline{X}(A_1 \vee \cdots \vee A_p)$, where $\overline{X}$ are the variables that occur in the A_i. The meaning of $\emptyset$ is $\bot$, where $\bot$ is a special formula that cannot be true in an interpretation. A clause is *ground* if there are no variables in it. The effect of a substitution on a clause is defined memberwise.

For example, the meaning of $\{p(X), q(X,Y)\}$ is $\forall X \forall Y(p(X) \vee q(X,Y))$. The meaning of $\{P, R(0)\}$ is $P \vee R(0)$.

Theorem 3.8.2 Let $\Gamma \vdash \Delta$ be a sequent. There exist algorithms which transform $\Gamma \vdash \Delta$ into a finite set of clauses $\{c_1, \ldots, c_n\}$, such that $\Gamma \vdash \Delta$ holds if and only if $\{c_1, \ldots, c_n\}$ is unsatisfiable.

Proof
The proof proceeds in a number of steps.

1. If $\Gamma \vdash \Delta$ contains free variables, then add quantors: If a formula F, with free variable V occurs in Γ, then F is replaced by $\forall VF$. If a formula F, with free variable V occurs in Δ, then F is replaced by $\exists VF$. It is easily checked that $\Gamma \vdash \Delta$ holds iff the result of these replacements holds.

2. After that $\Gamma \vdash \Delta$ can be Skolemised to obtain a sequent $\Gamma' \vdash \Delta'$. By Theorem 3.5.3,

$$\Gamma \vdash \Delta \text{ holds iff } \Gamma' \vdash \Delta' \text{ holds.}$$

   There exists an algorithm which constructs the Skolemisation.

3. After this we have a sequent $\Gamma \vdash \Delta$ with no positively occurring $\exists$-quantifiers, and no negatively occurring $\forall$-quantifiers. Write $\Gamma' \vdash \Delta'$ as $\{A_1, \ldots, A_p\} \vdash \{B_1, \ldots, B_q\}$, and replace it by $\{A_1, \ldots, A_p, \neg\, B_1, \ldots, \neg\, B_q\} \vdash$. Replace variables in such a manner that no different quantifiers have the same variable. Then apply the following rewrite rules as long as possible:

$$\begin{array}{lcl} A \to B & \Rightarrow & \neg\, A \vee B, \\ \neg\, (A \to B) & \Rightarrow & A \wedge \neg\, B, \\ \neg\, (A \wedge B) & \Rightarrow & \neg\, A \vee \neg\, B, \\ \neg\, (A \vee B) & \Rightarrow & \neg\, A \wedge \neg\, B. \end{array}$$

$$\begin{array}{lcl} (\forall X P) \wedge Q & \Rightarrow & \forall X (P \wedge Q), \\ P \wedge \forall X Q & \Rightarrow & \forall X (P \wedge Q), \\ (\forall X P) \vee Q & \Rightarrow & \forall X (P \vee Q), \\ P \vee \forall X Q & \Rightarrow & \forall X (P \vee Q). \end{array}$$

$$\begin{array}{lcl} (A \wedge B) \vee C & \Rightarrow & (A \vee C) \wedge (B \vee C), \\ A \vee (B \wedge C) & \Rightarrow & (A \vee B) \wedge (B \wedge C). \end{array}$$

4. After that the following rules are applied:

$$\forall X (P \wedge Q) \quad \Rightarrow \quad \forall X P \wedge \forall X Q,$$

   As long as the sequent can be written as $\Gamma, A \wedge B \vdash$, this is replaced by $\Gamma, A, B \vdash$.

5. The result of the previous step will be of the form: $\{c_1, \ldots, c_n\} \vdash$, where each c_i is of the form

$$c_i = \forall X_{i,1} \cdots \forall X_{i,v_i} (L_1 \vee \cdots \vee L_{i,l_i}).$$

   Then the sequent $\{c_1, \ldots, c_m\} \vdash$ can be replaced by the clause set $\{\{L_{1,1}, \ldots, L_{1,l_1}\}, \ldots, \{L_{n,1}, \ldots, L_{n,l_n}\}\}$.

End of Proof

The algorithm given here has an exponential worst case complexity, which is caused by the factorisation rules:

$$(A \wedge B) \vee C \Rightarrow (A \vee C) \wedge (B \vee C), \text{ and } A \vee (B \wedge C) \Rightarrow (A \vee B) \wedge (B \vee C).$$

However in practice this algorithm behaves quite well because the formulae which would cause exponential growth, do not occur in real life problems.

If one however finds this unacceptable it is possible to introduce labels in the translation. (See [Mints88]).
It has turned out that making this translation is a rather subtle art. Small variations in Skolemisation, or in the manner of introducing the labels may cause hyperexponential differences in the resolution proof that is found ([Egly]).
We have now reduced the problem of proving a sequent $\Gamma \vdash \Delta$ to the problem of proving that a certain clause set $\{c_1, \ldots, c_n\}$ is unsatisfiable. This can be done by resolution.

Definition 3.8.3 We define resolution and factoring:

Resolution Let

$$c_1 = \{A_1, \ldots, A_p\}, \text{ and}$$

$$c_2 = \{B_1, \ldots, B_q\}.$$

be two clauses with no overlapping variables, such that A_1 and $\neg\, B_1$ are unifiable, with most general unifier Θ. Then the clause

$$\{A_2\Theta, \ldots, A_p\Theta, B_2\Theta, \ldots, B_q\Theta\}$$

is a *resolvent* of c_1 and c_2. The literals A_1 and $\neg\, B_1$ are called the literals *resolved upon*.

Factorisation Let

$$c = \{A_1, \ldots, A_p\}$$

be a clause, such that A_1 is unifiable with one of the A_i, with $2 \leq i < p$. Let Θ be the most general unifier. Then the clause

$$\{A_2\Theta, \ldots, A_{i-1}, A_{i+1}, \ldots, A_p\Theta\}$$

is a *factor* of c. The literals A_1 and A_2 are called the literals *factored* upon.

These rules will be used as follows. One starts with the initial clause set and tries to derive the empty clause by exhaustive search. It is the case that the empty clause will be derived if and only if the initial clause set is unsatisfiable.
We will now give an example, then warn for some subtleties in the definition of the resolution rule, and then prove that resolution is sound and complete.

Example 3.8.4 $\{p(0), q(0)\}$ is a resolvent of $\{p(X), q(X), r(X)\}$ and $\{\neg\, r(0)\}$.
$\{p(X,X,Y), q(X,X,Y)\}$ and $\{\neg\, p(X,Y,Y), r(X,Y,Y)\}$ resolve to
$\{q(X,X,X), r(X,X,X)\}$.
The clause $\{p(X,X)\}$ is a factor $\{p(X,Y), p(Y,X)\}$.

Some care has to be taken in the application of the resolution rule. (See [Leitsch88]).
In the definition above, the construction of a resolvent goes as follows:

1. A_1 and B_1 are deleted resulting in $\{A_2, \ldots, A_p\}$ and $\{B_2, \ldots, B_q\}$.
2. After that Θ is applied to obtain $\{A_2\Theta, \ldots, A_p\Theta\}$ and $\{B_2, \ldots, B_q\}$. These sets are joined.

One might be tempted to do it in different order:

1. First replace $\{A_1, \ldots, A_p\}$ by $\{A_1\Theta, \ldots, A_p\Theta\}$, and $\{B_1, \ldots, B_q\}$ by $\{B_1\Theta, \ldots, B_q\Theta\}$.
2. Then delete $A_1\Theta$ from $\{A_1\Theta, \ldots, A_p\Theta\}$, and $B_1\Theta$ from $\{B_1\Theta, \ldots, B_q\Theta\}$ and join the results.

This, however, is a different resolution rule, because it may be the case that $A_1\Theta = A_i\Theta$, for an $i \neq 1$, or $B_1\Theta = B_j\Theta$, for a $j \neq 1$. In the first case, in the second version of the resolution rule $A_i\Theta$ will be not present in the resolvent, whereas in the first version $A_i\Theta$ would occur in the resolvent.
When, for example $\{p(X,Y), p(Y,X)\}$ and $\{\neg\, p(0,0)\}$ resolve according to the first version, the result will be $\{p(0,0)\}$. When they resolve according to the second version, the result will be $\{\}$.
The first version of resolution is weaker than the second version, because the clause, derived with the second version is a subset of the clause, derived with the first version.
We will now prove the soundness of resolution by proving the following: If the meaning of a clause c is true in an interpretation, and c' is a factor of c, then the meaning of c' is true in the interpretation. Here we use the stronger notion of resolution. If the meanings of c_1 and c_2 are true in an interpretation, then the meaning of every possible resolvent of c_1 and c_2 is true in the interpretation.

Lemma 3.8.5 Let A be a literal. Let $A[X := t]$ be an instance of A, such that X does not occur in t. Let $I_1 = (D, [\ ]_1)$ be an interpretation of A, and let $I_2 = (D, [\ ]_2)$ be an interpretation of $A[X := t]$, such that

1. For all function and constant symbols f and c,

$$[f]_1 = [f]_2 \text{ and } [c]_1 = [c]_2.$$

2. For all variables V in A, except X,

$$[V]_1 = [V]_2.$$

3.

$$[X]_1 = [t]_2.$$

Then $(D, [\ ]_1)$ makes A true iff $(D, [\ ]_2)$ makes $A[X := t]$ true.

The proof is obtained by induction on the structure of A.

Lemma 3.8.6 Resolution and factoring are sound rules. We prove the following:

1. Let c be a clause, let c' be a factor of c. Let I be an interpretation of the meaning of c, such that the meaning of c is true in I. Then the meaning of c' is true in I.

2. Let c_1 and c_2 be clauses. Let c be a resolvent of c_1 and c_2. If I is an interpretation of the meanings of c_1, and c_2, such that both are true in I, then the meaning of c is true in I.

Proof
First we prove

- Let $c\Theta$ be an instance of a clause c, such that for no variable V,

$$V \text{ occurs in } V\Theta \text{ and } V \neq V\Theta.$$

  If the meaning of c is true in I, then the meaning of $c\Theta$ is true in I.

It is sufficient to prove this for simple Θ of the form $\Theta = \{X := t\}$. More complex substitutions can be obtained by iteration. Write the variables that occur in t as $Z_1, \ldots, Z_m$. Write the meanings of c and $c\Theta$ as

$$\forall X Y_1 \cdots Y_n (A_1 \vee \cdots \vee A_p), \text{ and}$$

$$\forall Z_1 \cdots Z_m Y_1 \cdots Y_n (A_1[X := t] \vee \cdots \vee A_p[X := t]).$$

(It is not excluded that some of the Z_i and Y_j are equal) Assume that the meaning of c is true in an interpretation I. This implies that in every interpretation $I_1 = (D, [\ ]_x^X\ {}_{y_1}^{Y_1} \cdots\ {}_{y_n}^{Y_n})$, the formula

$$A_1 \vee \cdots \vee A_p$$

is true.
We will prove from this that $A_1[X := t] \vee \cdots \vee A_p[X := t]$ is true in every interpretation $I_2 = (D, []_{z_1}^{Z_1} \cdots {}_{z_m}^{Z_m}\ {}_{y_1}^{Y_1} \cdots {}_{y_n}^{Y_n})$.
From an arbitrary I_2, an interpretation I_1 can be constructed by putting $[X]_1 = [t]_2$. Then by Lemma 3.8.5, A_i is true in I_1 iff $A_i[X := t]$ is true in I_2. It follows that one of the $A_i[X := t]$ must be true in I_2, and hence the meaning of $c\{X := t\}$ must be true in I.
From what we have now the soundness of factorisation follows. We prove the soundness of the strong version of resolution. The soundness of the weak version follows immediately from it.
Let $c_1 = \{A_1, \ldots, A_p\}$ and $c_2 = \{B_1, \ldots, B_q\}$ with $A_1 = \neg\ B_1$. Let $I = (D, [\,])$ be an interpretation of c_1 and c_2. Assume that both the meanings of c_1 and c_2 are true in I. Let $I' = (D, [\,]_{v_1}^{V_1} \cdots {}_{v_n}^{V_n})$ be an interpretation of $A_2 \vee \cdots \vee A_p \vee B_2 \vee \cdots \vee B_q$. I' is also an interpretation of $A_1 \vee \cdots \vee A_p$ and $B_1 \vee \cdots \vee B_q$. Since by assumption both must be true in I', and it is impossible that both B_1 and A_1 are true in I', either $A_2 \vee \cdots \vee A_p$, or $B_2 \vee \cdots \vee B_q$ must be true in I'. In both cases $A_2 \vee \cdots A_p \vee B_2 \vee \cdots \vee B_q$ is true in I'. Because of this the meaning of c is true in I.
End of Proof

We have now proven the soundness of resolution. It remains to prove the completeness. For this we prove first that every unsatisfiable set of ground clauses has a resolution refutation. After that we prove the so called *lifting lemma*.

Lemma 3.8.7 Let $C = \{c_1, \ldots, c_n\}$ be a finite set of propositional clauses. If C is unsatisfiable then C has a resolution refutation.

Proof
Let C_1 and C_2 be two finite clause sets. We write $C_1 \sqsubset C_2$ iff C_1 can be obtained from C_2 by deleting some literals from some clauses. This relation can be used for induction because there is only a finite number of literals to delete. If $C_1 \sqsubset C_2$, and C_2 is unsatisfiable, then C_1 is unsatisfiable. We prove Lemma 3.8.7 by induction on this relation.

Basis If for all $c_i \in C$, the length is at most one, then either C contains the empty clause, or a complementary pair $\{A\}$ and $\{\neg\ A\}$. In both case $\emptyset$ can be derived rather quickly.

Step If for some $c_i \in C$, the length is at least two, then let A be a literal in c_i. Define

- $C \backslash A$ is the result of deleting A from all clauses in C, in which A occurs.

- $C \rightarrow A$ is the result of replacing all clauses in C, with length at least two and in wich A occurs by the clause $\{A\}$.

Both $C \backslash A$ and $C \rightarrow A$ are unsatisfiable. By the induction hypothesis both have a derivation of $\emptyset$. Consider the derivation of $\emptyset$ from $C \backslash A$. There are two possibilities:

1. None of the clauses in C that contained A has been used. In that case the empty clause can also be derived from C.
2. One of the clauses that contained A in C has been used. Then the derivation of $\emptyset$ from $C \backslash A$ can be replaced by a derivation of $\{A\}$ from C. Because $\emptyset$ can be derived from $C \rightarrow A$, the empty clause can be derived from $C \cup \{\{A\}\}$, and hence from C.

End of Proof

Now we have proven the propositional completeness. We will now prepare for the proof of the Lifting Lemma.

Lemma 3.8.8 Let c be a clause with an instance $c\Theta$. Let A be a literal in $c\Theta$. If there is more than one literal $B \in c$, for which $A = B\Theta$, then c has a (possibly iterated) factor $c\Sigma$, such that there is only one literal in $c\Sigma$ of which A is an instance.

Proof
We will reduce the number n of literals $B \in c$ for which $B\Theta = A$ by computing a factor $c\Sigma$, until $n = 1$. If $n > 1$, then there are at least two literals B_1 and B_2, for which $B_1\Theta = B_2\Theta = A$. Because B_1 and B_2 are unifiable, they have an mgu Σ. Then Θ can be written as $\Theta = \Sigma \cdot \Xi$. Then $c\Theta = c(\Sigma \cdot \Xi) = (c\Sigma)\Xi$. Because of this $c\Theta$ is an instance of $c\Sigma$. There are strictly less literals $B \in c\Sigma$ for which $B\Xi = A$, because B_1 and B_2 are unified in $c\Sigma$.
End of Proof

Lemma 3.8.9 (Lifting Lemma) Let c_1 and c_2 be two clauses with instances $\overline{c}_1$ and $\overline{c}_2$. If $\overline{c}_1$ and $\overline{c}_2$ have a resolvent $\overline{c}$, then c_1 and c_2 have a resolvent c, such that $\overline{c}$ is an instance of c.

Proof
Write

$$\overline{c}_1 = \{\overline{A}_1, \ldots, \overline{A}_p\},$$
$$\overline{c}_2 = \{\overline{B}_1, \ldots, \overline{B}_q\}.$$

where $\overline{A}_1 = \neg\ \overline{B}_1$, and $\overline{A}_1$ and $\overline{B}_1$ are the literals resolved upon.
By Lemma 3.8.8 there exist (possibly iterated) factors of c_1 and c_2, such that $\overline{A}_1$ and $\overline{B}_1$ are an instance of only one literal in c_1 and c_2. We from now on assume that c_1 and c_2 are already factored. Then we have that $\overline{c}_1 \backslash \{\overline{A}_1\}$ is an instance of $c_1 \backslash \{A_1\}$, because no other literal than A_1 in c_1 will have $\overline{A}_1$ as instance. In the same way $\overline{c}_2 \backslash \{\overline{B}_1\}$ is an instance of $c_2 \backslash \{B_1\}$.
If we assume that c_1 and c_2 have no overlapping variables, then there exists one substitution Θ, such that

1. $A_1\Theta = \neg\ B_1\Theta$, and
2. $\overline{c} = \{A_2\Theta, \ldots, A_p\Theta, B_2\Theta, \ldots, B_q\Theta\}$.

Because of this A_1 and $\neg\ B_1$ are unifiable, and c_1 and c_2 have a resolvent. Define Σ as the mgu of A_1 and $\neg\ B_1$. By definition of mgu, Θ can be written as $\Theta = \Sigma \cdot \Xi$. The resolvent equals $\{A_2\Sigma, \ldots, A_p\Sigma, B_2\Sigma, \ldots, B_q\Sigma\}$, from which $\{\overline{A}_2, \ldots, \overline{A}_p, \overline{B}_2, \ldots, \overline{B}_q\}$ will be obtained by Ξ.
End of Proof

Theorem 3.8.10 Resolution is complete. If a clause set C is unsatisfiable, then the empty clause can be derived from it.

Proof
Assume that C is unsatifiable. Then:

1. By Herbrands theorem, (Theorem 3.6.2), there exists a finite set $\{c_1, \ldots, c_n\}$ of ground instances of elements of C, such that $\{c_1, \ldots, c_n\}$ is unsatisfiable.
2. By Lemma 3.8.7 this set has a resolution refutation.
3. By Lemma 3.8.9 this ground refutation can be lifted to a non-ground resolution refutation of C.

End of Proof

Example 3.8.11 Suppose one wants to prove the following sequent by resolution:

$$\forall X(p(X) \rightarrow q(Y)) \vdash \neg\ [\forall X(\neg\ p(X) \wedge \neg\ q(X))] \rightarrow \exists Y q(Y).$$

First an existential quantifier is added:

$$\exists Y \forall X(p(X) \rightarrow q(Y)) \vdash \neg\ [\forall X(\neg\ p(X) \wedge \neg\ q(X)) \rightarrow \exists Y q(Y).$$

Then the sequent can be Skolemised:

$$\forall X(p(X) \rightarrow q(c)) \vdash \neg\ (\neg\ p(d) \wedge \neg\ q(d)) \rightarrow \exists Y q(Y).$$

The right side is brought to the left:

$$\forall X(p(X) \rightarrow q(c)), \neg\ [(\neg\ (\neg\ p(d) \wedge \neg\ q(d)) \rightarrow \exists Y q(Y)] \vdash .$$

The rewrite rules give:

$$\forall X(\neg\ p(X) \vee q(c)), p(d) \vee q(d), \forall Y[\neg\ q(Y)] \vdash .$$

This results in the following set of clauses:

(1) $\{\neg\ p(X), q(c)\}$
(2) $\{p(d), q(d)\}$
(3) $\{\neg\ q(Y)\}$.

Then, there is the following resolution refutation: We give both the ground refutation, and the non-ground refutation:

(1)	$\{\neg\ p(d), q(c)\}$	$\{\neg p(X), q(c)\}$	(initial clause)
(2)	$\{p(d), q(d)\}$	$\{p(d), q(d)\}$	(initial clause)
(3)	$\{\neg\ q(d)\}$	$\{\neg\ q(Y)\}$	(initial clause)
(4)	$\{\neg\ q(c)\}$	$\{\neg\ q(Y)\}$	(initial clause)
(5)	$\{\neg\ p(d)\}$	$\{\neg\ p(Y)\}$	(from 1 and 4)
(6)	$\{q(d)\}$	{ q(d) }	(from 2 and 5)
(7)	$\{\}$	$\{\}$	(from 3 and 6)

3.9 Refinements of Resolution

In this section we present the standard ordering refinements of resolution. (See [ChangLee73], and [Lovelnd78]).

Definition 3.9.1

- An L-order $\prec$ is an order on literals with the following property:

  $$A \preceq B \text{ implies } A\Theta \preceq B\Theta, \text{ for all substitutions } \Theta.$$

  This property is called *liftability*.
- An A-order is an L-order with the following property:

  $$A \prec B \text{ implies } \pm A \prec \pm B.$$

- An *interpretation* is a set of literals I, with the following properties:
  - If $A \in I$, then for all instances $A\Theta$ of A, $A\Theta \in I$.
  - If $A \notin I$, then for all instances $A\Theta$ of A, $A\Theta \notin I$.
  - For each atom A exactly one of A and $\neg A$ is in I.

  A clause c is called *true* in I if $I \cap c \neq \emptyset$. Otherwise it is called *false*.
- Let C a clause set. An *indexing* of C is obtained by replacing each clause $\{A_1, \ldots, A_p\}$ by $\{A_1{:}n_1, \ldots, A_p{:}n_p\}$. Here for each clause the n_i can be chosen arbitrarily.

The notions of L-order and interpretation can be slightly generalised by considering only literals that are instances of literals in a certain clause set. The common definition for liftability is:

$$A \prec B \text{ implies } A\Theta \prec B\Theta.$$

The liftability given above is slightly more general, because the substitution is allowed to unify literals. There exist orderings which satisfy the weaker condition, but which do not satisfy the stronger condition, which can be lifted: An example is: $A \prec B$ if $A \neq p(0)$, and $B = p(0)$.
Using the previous we define the following refinements of resolution:

Definition 3.9.2

- **L-ordered resolution, A-ordered resolution**.
  L-ordered resolution is obtained from ordinary resolution, by taking an L-order $\prec$, and adding the following conditions:
  1. When a resolvent is constructed the literals resolved upon must be maximal.
  2. When a factor is constructed, one of the literals factored upon must be maximal.

  A-ordered resolution is obtained in the same manner, but taking an A-order instead of an L-order.
- **Semantic Resolution**.
  Semantic resolution is obtained from ordinary resolution, by fixing an interpretation I, and adding the following conditions:
  1. When a resolvent is constructed, one of the clauses resolved upon must be false.

2. Only factors of false clauses can be constructed.

- **Hyperresolution**.
  Hyperresolution is obtained by making leaps in semantic resolution. If a clause is true in I, then it can be written as: $\{A_1, \ldots, A_p, B_1, \ldots, B_q\}$, where $\{A_1, \ldots, A_p\} \subseteq I$, and $\{B_1, \ldots, B_q\} \cap I = \emptyset$. If this clause will be resolved with a false clause, the result will contain one true literal less. Then as long as the result contains true literals it will have to resolve with a false clause. This happens p times. After the p-th time the result is false. Hyperresolution is obtained from semantic resolution by only keeping the final false clauses. So, hyperresolution is obtained by imposing the following conditions:
  1. A true clause, containing q true literals has to be resolved q times with a false clause. Only the results of these iterated resolution steps are kept.
  2. Factorisation is only allowed on false clauses.
- **A-ordered hyperresolution, A-ordered semantic resolution**.
  A-ordered hyperresolution is obtained by combining the hyperresolution refinement with A-ordered resolution.
- **Lock Resolution**.
  Lock resolution is obtained as follows. The literals of the initial clauses are indexed. After that resolution and factoring are modified as follows:
  1. Only literals with maximal indices can be resolved upon. When a resolvent is constructed, the literals in the result inherit the indices from the original clauses.
  2. When a factor is constructed one of the literals factored upon must have a maximal index.

Then we have:

Theorem 3.9.3 All refinements, mentioned in Definition 3.9.2 are complete.

Proof
The proof consists of two parts.

1. First prove propositional completeness for all of the refinements.
2. Then prove that ground refutations in each refinement can be lifted.

We begin by proving propositional completeness. The completeness proof of L-ordered resolution can be obtained by making a small adaptation in the proof of Lemma 3.8.7. In the step the following will be changed: If one of the clauses has a length of at least two, then a minimal literal occurring in the clauses of length at least two will be chosen (as A). The rest of the proof can be the same because the clause $\{A\}$ can be derived.

The completeness proof of lock resolution can be obtained in the same manner: From the clauses of length at least two select a literal with minimal index.

We will prove the propositional completeness of the remaining refinements by proving the completeness of A-ordered semantic resolution. From this follows the completeness of hyperresolution, A-ordered hyperresolution, and semantic resolution. We have to be a bit careful because of the restriction in factoring on the non-ground level. In order to avoid problems in lifting we take all true clauses as multisets, and all false clauses as ordinary sets. Then the proof of Lemma 3.8.7 can be adapted as follows:

Basis If for all $c_i \in C$, the number of false literals in c_i equals at most 1, then every false clause will be a unit clause, or the empty clause.

Step If for some $c_i \in C$, there is more than one true literal in c_i, then choose A as a minimal true literal from the clauses that have more than one true literal. The rest of the proof is the same.

This results in an A-ordered semantic refutation which has no implicit factoring in true clauses.
In order to prove the possibility of lifting for all the refinements here it is sufficient to prove the following, for each refinement:
If $\{A_1, \ldots, A_p\}$ has instance $\{A_1\Theta, \ldots, A_p\Theta\}$, then

1. If $A_1\Theta$ is allowed to be resolved upon by the refinement, then if $A_1\Theta$ is an instance of more than one A_i, then one of the A_i is allowed to be factored upon by the refinement. Using this Lemma 3.8.8 can be proven.

2. If $A_1\Theta$ is allowed to be resolved upon by the refinement, and $A_1\Theta$ is an instance of only A_1, then A_1 is allowed to be resolved upon by the refinement.

For L-ordered resolution we note: If $A_1\Theta$ is an instance of $A_1, \ldots, A_n$, and $A_1\Theta$ is maximal in $c\Theta$, then one of the A_i is maximal in c. Suppose that none of the A_i is maximal. Then there is a B, such that $A_1 \prec B$, and

$A_1\Theta \neq B\Theta$, and $A_1\Theta \not\prec B\Theta$. This is a violation of the liftability property, as defined in Definition 3.9.1.
For lock resolution we note that if $A_1{:}\Theta$ is maximal in $c\Theta$, then all $A_i{:}n$ which have $A_1{:}\Theta$ as instance have the same index. As a consequence these are also maximal.
For A-ordered semantic resolution we note the following: If $A_1\Theta$ is a literal which is allowed to be resolved upon, and $A_1\Theta$ is an instance of more than one A_i, then $A_1\Theta$ occurs in a false clause, because true clauses are multi-sets. Because the A-order is liftable one of the A_i can be factored upon.
End of Proof

Chapter 4

How to Obtain Resolution Calculi

4.1 Introduction

In the previous chapter there were three proof methods, namely sequent calculus, semantic tableaux, and resolution. In this chapter we will try to look for generalisations of the resolution method.
Because resolution is successfull in classical logic it is natural to look for generalisations of resolution in non-standard logics. In order to to this it is necessary to know what are the essential features of resolution. We think that there are two:

1. Resolution is non-analytical.
2. Resolution is local.

Let us give a definition of analytical. It is not straightforward to give a definition of this notion, because deduction systems may have peculiarities which are difficult to catch. Therefore we give a rather non-strict definition:

Definition 4.1.1 A proof method is *analytical* if in order to prove a formula F, only subformulae of this formula are used. In the proofs of the subformulae again only subformulae can be used.

Sequent calculus without cut, as defined in Definition 3.3.4, is analytical. Addition of the cut rule makes sequent calculus non-analytical, because, it becomes possible to prove $\Gamma_1, \Gamma_2 \vdash \Delta_1, \Delta_2$ from $\Gamma_1, A \vdash \Delta_1$, and $\Gamma_2 \vdash \Delta_2, A$, with A not a subformula.

The semantic tableau calculus of Section 3.4 is not analytical because it contains rules $\neg(A \wedge B) \Rightarrow \neg A \vee \neg B$. However it is almost analytical, because in such rules the result has almost the same structure as the premisse. More important, there is no manner to introduce a quantifier in a semantic tableau, once it is deleted. Because of this the method of semantic tableaux is also analytical.
Resolution may appear to be analytical, but is not analytical. This is because a clause is implictly universally quantified. For example

$$\{p(X), q(X)\} \text{ and } \{\neg\, p(X)\} \text{ resolve to } \{q(X)\}.$$

Although $q(X)$ is a subformula of $p(X) \vee q(X)$, resolution is non-analytical because one should consider the meaning. The formula $\forall X q(X)$ is not a subformula of $\forall X(p(X) \vee q(X)\ )$.
A non-analytical proof search method has two advantages over an analytical proof search method. A deterministic and a non-deterministic advantage:

1. There is a deterministic advantage that the decision which X to take in $q(X)$ can be postponed until it is known which X is needed. As a consequence a deterministic computer will spend less time on wrong guesses for X.

2. There is a non-deterministic advantage that the derived clause $q(X)$ represents multiple instances. If one would have to choose X before computing the resolvent, the resolvent would have to be computed one time for each instance. Now it is possible to have one resolution step for infinitely many values for X. This may cause a speed up which will also be present on a non-deterministic computer, because it depends on the length of the proof constructed.

In general a non-analytical proof method has a disadvantage for automated theorem proving, namely that it needs creativity. If one for example in sequent calculus would like to add the cut rule, then it would be necessary to determine the A which is to be deleted in the cut. However this disadvantage is not present in resolution because the possibility to introduce formulae which are not a subformula is very limited.
Next we discuss the fact that resolution is local. Both semantic tableau and sequent calculus can be transformed into a non-analytical method using the so called method of *metavariables*. (See [Maslov86]) At the moment it is necessary to choose a substitution $A[X := t]$ for a formula $\forall X A$, this is not done, but instead X is replaced by a so called *metavariable*. In this manner it is possible to construct a proof tree as if there were a term instead of the metavariable. Only when axioms are reached one will try to fill in

values for the metavariables. The method that is thus obtained from sequent calculus is called a *global method* because different occurrences of the same metavariable may occur scattered anywhere through the proof.
Now when one makes a substitution for a metavariable there is risk that it is a wrong substitution. In the case of a wrong choice using a global method the whole proof structure is spoilt.
In resolution shared variables occur only within one clause. As a consequence resolution behaves much better in backtracking. Because a variable is only shared within one clause, in the case that a useless resolvent is derived, only this resolvent will be useless. The other clauses are not affected. Because of this resolution is very well adapted for automated proof search.
It can be concluded now that if one wants a generalisation of resolution, one has to look for a slightly non-analytical method, which needs only local variables.
In the next sections we discuss two manners to obtain such methods. The first is called the Maslov-Mints method. It is based on the observation that when one reverses the direction of proof search in sequent calculus, the result will be a local method. The second is a method to obtain resolution calculi from semantic tableau by restructuring a closed semantic tableau.

4.2 From Sequent Calculi to Resolution

In this section we discuss a method of obtaining resolution calculi from sequent calculi. It is developed in ([Mints88]), as a generalisation of a method introduced by S. Yu. Maslov, in ([Maslov71]). It is based on the observation that sequent calculus is almost resolution if one searches for the proof in bottom up direction, i.e. from the axioms towards the goal clause. There are some differences between sequent calculus and resolution but these can be removed by normalising a proof in sequent calculus. We think that the method can best be explained from an example. We will use a sequent calculus for classical propositional logic for this purpose, with only formulae on the right side. After that we give a general scheme for sequent calculi. This scheme is not the most general possible, because it does not include substructural logics as linear logic ([Girard87]) and intuitionistic logic. The scheme is a compromise between readability and absence of many technical problems on one side and being able to expose the generality on the other side. It is however possible to make adaptations for intuitionistic logic ([Mints88]) and linear logic ([Mints93]).

Using the general scheme we show that the proofs of sequent calculi can be transformed into resolution proofs by making some adaptations. The resolution proofs are proofs in the sequent calculus.

The idea is based on the fact that deriving a sequent $\vdash \Gamma$ can be seen as deriving a clause.
We now begin by giving a sequent calculus for classical propositional logic, in which only formulae on the right are allowed.

Example 4.2.1

Axiom

- Every sequent of the form $\vdash A, \neg A$ is an axiom and hence derivable.

Logical Rules

- If the sequent $\vdash \Gamma, A$ is derivable, then the sequent $\vdash \Gamma, A \vee B$ is derivable.
- If the sequent $\vdash \Gamma, B$ is derivable, then the sequent $\vdash \Gamma, A \vee B$ is derivable.
- If the sequents $\vdash \Gamma_1, A$ and $\vdash \Gamma_2, B$ are derivable, then the sequent $\vdash \Gamma_1, \Gamma_2, A \wedge B$ is derivable.
- If the sequent $\vdash \Gamma, \neg A$ is derivable, then the sequent $\vdash \Gamma, A \rightarrow B$ is derivable.
- If the sequent $\vdash \Gamma, B$ is derivable, then the sequent $\vdash \Gamma, A \rightarrow B$ is derivable.
- If the sequent $\vdash \Gamma, A$ is derivable, then the sequent $\vdash \Gamma, \neg \neg A$ is derivable.
- If the sequent $\vdash \Gamma, \neg A \vee \neg B$ is derivable, then the sequent $\vdash \Gamma, \neg (A \wedge B)$ is derivable.
- If the sequent $\vdash \Gamma, \neg A \wedge \neg B$ is derivable, then the sequent $\vdash \Gamma, \neg (A \vee B)$ is derivable.
- If the sequent $\vdash \Gamma, A \wedge \neg B$, is derivable, then the sequent $\vdash \Gamma, \neg (A \rightarrow B)$ is derivable.

Weakening

- If the sequent $\vdash \Gamma$ is derivable, then the sequent $\vdash \Gamma, A$ is derivable.

It can be seen easily that this sequent calculus is sound and complete for classical propositional logic. In this sequent calculus, sequents are treated as sets. This means in the sequent $\vdash \Gamma, A$ it is possible that $A \in \Gamma$.

Example 4.2.2 The tautological sequent $\vdash a \wedge b \wedge c, \neg\,(b \rightarrow a), \neg\,(b \wedge c)$ has the following derivation:

$$\dfrac{\dfrac{\dfrac{\dfrac{\vdash b, \neg\, b}{\vdash b, \neg\, b \vee \neg\, c} \qquad \vdash a, \neg\, a}{\vdash a, b \wedge \neg\, a, \neg\, b \vee \neg\, c,} \qquad \dfrac{\dfrac{\vdash b, \neg\, b}{\vdash b, \neg\, b \vee \neg\, c} \qquad \vdash c, \neg\, c,}{\vdash c, \neg\, b \vee \neg\, c.}}{\vdash b \wedge c, \neg\, b \vee \neg\, c,}}{\vdash a \wedge b \wedge c, b \wedge \neg\, a, \neg\, b \vee \neg\, c,}}{\vdash a \wedge b \wedge c, \neg\,(b \rightarrow a), \neg\, b \vee \neg\, c,}}{\vdash a \wedge b \wedge c, \neg\,(b \rightarrow a), \neg\,(b \wedge c).}$$

We will give a more general scheme of sequent calculi: It is general enough to be able to desribe the Mints method in general, but special enough to avoid some difficulties with substructural rules which we don't want here.

Definition 4.2.3 A sequent calculus is represented by a pair $S = (P, \mathcal{R})$.

- P is the set of formulae that can occur in the sequents. We will call the elements of P also literals, or propositional symbols, dependent on the context.
- $\mathcal{R}$ is the set of rules that can be used in the derivations. They have form $\{A_1, \ldots, A_p\} \vdash \{B_1, \ldots, B_q\}$. Here $p \geq 0$, and $q \geq 0$.

The sequents are treated as sets and there are the following rules:

Weakening If the sequent $\vdash \Gamma$ is derivable, then the sequent $\vdash \Gamma, A$ is derivable, for every $A \in P$.

Logical Rules If the rule $\{A_1, \ldots, A_p\} \vdash \{B_1, \ldots, B_q\} \in \mathcal{R}$, and the sequents $\vdash \Gamma_1, A_1$ up to $\vdash \Gamma_p, A_p$, are derivable, then the sequent $\vdash B_1, \ldots, B_q, \Gamma_1, \ldots, \Gamma_p$ is derivable.

In this formal definition the distinction between logical rules and axioms has dissappeared. The axioms are logical rules for which $p = 0$. This scheme of sequent calculi does not allow substructural logics, because the sequents are sets.

Example 4.2.4 We show how the sequent calculus given above fits into this scheme. Put $\mathcal{S} = (P, \mathcal{R})$ with P is the set of first order formulae, and $\mathcal{R}$ defined by the following schemata:

- $\{\} \vdash \{A, \neg\, A\}$,
- $\{A\} \vdash \{A \vee B\}$,
- $\{B\} \vdash \{A \vee B\}$,
- $\{A, B\} \vdash \{A \wedge B\}$,
- $\{\neg\, A\} \vdash \{A \rightarrow B\}$,
- $\{B\} \vdash \{A \rightarrow B\}$,
- $\{A\} \vdash \{\neg\, \neg\, A\}$,
- $\{\neg\, A, \neg\, B\} \vdash \{\neg\, (A \vee B)\}$,
- $\{\neg\, A \vee \neg\, B\} \vdash \{\neg\, (A \wedge B)\}$,
- $\{\neg\, A \wedge \neg\, B\} \vdash \{\, \neg\, (A \vee B)\}$,
- $\{A \wedge \neg\, B\} \vdash \{\neg\, (A \rightarrow B)\}$.

These rules are schemata. For example the rule schema $\{A\} \vdash \{A \vee B\}$ represents infinitely many rules. A possible instance is $\{p \wedge q\} \vdash \{(p \wedge q) \vee r\}$. We have now the tools to give the general idea in ([Mints88]). It is based on the observation that an application of a logical rule has a similar structure as an application of the hyperresolution rule: The sequent rule

- If there are sequents $\vdash \Gamma_1, A_1, \ldots, \vdash \Gamma_p, A_p$, then the sequent $\vdash \Gamma_1, \ldots, \Gamma_p, B_1, \ldots, B_q$, is derivable

is similar to the hyperresolution rule

- If there are clauses $c_1, \ldots, c_p$, such that $A_1 \in c_1, \ldots, A_p \in c_p$, then the following clause is a hyperresolvent of $c_1, \ldots, c_p$,

$$(c_1 \backslash \{A_1\}) \cup \cdots \cup (c_p \backslash \{A_p\}) \cup \{B_1, \ldots, B_q\}.$$

However there are some differences:

A1 There are no resolution rules which correspond to the sequent rules $\{A_1, \ldots, A_p\} \vdash \{B_1, \ldots, B_q\}$ with $p = 0$. For this reason the corresponding resolution rules have to be replaced by initial clauses: If there is a rule $\vdash \{B_1, \ldots, B_q\}$ then the clause

$$\{B_1, \ldots, B_q\},$$

is an initial clause.

A2 In the sequent rule above it is possible that $A_i \in \Gamma_i$, and that A_i is copied into the resulting sequent. This is impossible with resolution. Using resolution the literal A_i is always removed from the result. This difference is ignored in ([Mints88]). It is however not completely trivial. We use a form of proof normalization to remove it.

A3 In sequent calculus, there is a weakening rule, which has no equivalent in the resolution calculus. We use proof normalisation to remove this difference. Using proof normalisation it is possible to move all aplications of weakening to the last clause.

A4 In general, in sequent calculus one wants to prove a goal sequent $\vdash C_1, \ldots, C_r$ from the axioms, whereas with resolution one starts with the formula that one wants to prove, and tries to derive a special clause, the empty clause. This difference can be removed by changing the rules: Every rule $\{A_1, \ldots, A_p\} \vdash \{B_1, \ldots, B_q\}$ is replaced by

$$\{A_1, \ldots, A_p\} \vdash \{B_1, \ldots, B_q\} \backslash \{C_1, \ldots, C_r\}.$$

We prove in Theorem 4.2.7 that it is possible to take as goal sequent the empty clause in the calculus that is thus obtained. We use the notation $S(C_1, \ldots, C_r)$, for the resulting calculus. However despite the fact that the empty clause is derived instead of the goal clause, the resulting resolution calculus remains a bottom up calculus.

We will now discuss how the differences can be removed. We will first settle A2 and A3. This is done by proof normalisation.

Theorem 4.2.5 If $S = (P, \mathcal{R})$ is the representation of a sequent calculus, and a sequent $\vdash C_1, \ldots, C_r$ is derivable, then there is a derivation of $\vdash C_1, \ldots, C_r$, in which

1. When a logical rule $\{A_1, \ldots, A_p\} \vdash \{B_1, \ldots, B_q\}$ is applied on a sequent $\vdash \Gamma_i, A_i$ then $A_i \notin \Gamma_i$,
2. All applications of the weakening rule are at the end. So, first a sequent $\vdash \Gamma$, with $\Gamma \subseteq \{C_1, \ldots, C_r\}$ is derived without using the weakening rule, and next it is weakened to $\vdash \{C_1, \ldots, C_r\}$.

Proof
We first establish 1 by making replacements in the proof, and next we apply 'bubblesort' on the proof to obtain 2.

1. Suppose that somewhere in the proof the sequent

$$\vdash \Delta = \vdash \Gamma_1, \ldots, \Gamma_p, B_1, \ldots, B_q$$

is derived as

$$\begin{array}{l} \vdash \Gamma_1, A_1, \\ \cdots \\ \vdash \Gamma_p, A_p, \\ \hline \vdash \Gamma_1, \ldots, \Gamma_p, B_1, \ldots, B_q, \end{array}$$

and that an $A_i \in \Gamma_i$,
We replace Γ_i by $\Gamma_i \backslash \{A_i\}$. The result is the following proof step:

$$\begin{array}{l} \vdash \Gamma_1, A_1, \\ \cdots \\ \vdash \Gamma_i \backslash \{A_i\}, A_i \\ \cdots \\ \vdash \Gamma_p, A_p, \\ \hline \vdash \Gamma_1, \ldots, \Gamma_i \backslash \{A_i\}, \ldots, \Gamma_p, B_1, \ldots, B_q = \ \vdash \Delta'. \end{array}$$

Now either $\Delta' = \Delta$, in which case we do nothing, or $\Delta' = \Delta \backslash \{A_i\}$. In that case we apply weakening to obtain Δ. It is easily seen that this process of replacing proof steps is a terminating process.

2. Next we prove 2, by showing that all weakening steps can be moved upward. Suppose that we have the following proof step:

$$\begin{array}{l} \vdash \Gamma_1, A_1 \\ \cdots \\ \vdash \Gamma_p, A_p \\ \hline \vdash \Gamma_1, \ldots, \Gamma_p, B_1, \ldots, B_q, \end{array}$$

and that a $\vdash \Gamma_i, A_i$ is obtained by weakening. We distinguish:

- $\vdash \Gamma_i, A_i$ is obtained from Γ_i. In that case $\vdash \Gamma_1, \ldots, \Gamma_p, B_1, \ldots, B_q$ can be obtained from $\vdash \Gamma_i$ by weakening. So the application of the logical rule can be removed completely.
- $\vdash \Gamma_i, A_i$ is obtained from a sequent $\vdash \overline{\Gamma_i}, A_i$ by weakening. Because $A_i \notin \Gamma_i$ (as the result of step 1), $A_i \notin \overline{\Gamma_i}$. It is possible to make the following proof step:

$$\frac{\begin{array}{l}\vdash \Gamma_1, A_1, \\ \ldots \\ \vdash \overline{\Gamma_i}, A_i, \\ \ldots \\ \vdash \Gamma_p, A_p\end{array}}{\vdash \Gamma_1, \ldots, \overline{\Gamma_i}, \ldots, \Gamma_p, B_1, \ldots, B_q}$$

  The result can be weakened to $\vdash \Gamma_1, \ldots, \Gamma_p, B_1, \ldots, B_q$ in 0 or 1 step.

This is a terminating process because the number of applications of a logical rule after an application of weakening strictly decreases for each weakening application.

End of Proof

Now we consider A4. It is easily seen that adding the goal deletion rules makes it possible to derive the empty sequent instead of the goal sequent. We must also prove the converse: If it is posssible to derive the empty sequent with the goal deletion rules, then it is possible to derive the goal sequent, without the goal deletion rules.

Definition 4.2.6 Let $S = (P, \mathcal{R})$ define a sequent calculus. Let $\vdash C_1, \ldots, C_r$ be a sequent. We define $S(C_1, \ldots, C_r)$ as the sequent calculus, defined by

1. Replacing P by $P \backslash \{C_1, \ldots, C_r\}$.
2. Replacing all rules $\{A_1, \ldots, A_p\} \vdash \{B_1, \ldots, B_q\}$, by $\{A_1, \ldots, A_p\} \vdash \{B_1, \ldots, B_q\}) \backslash \{C_1, \ldots, C_r\}$.
3. Removing all rules $\{A_1, \ldots, A_p\} \vdash \{B_1, \ldots, B_q\}$, for which $\{A_1, \ldots, A_p\} \cap \{C_1, \ldots, C_r\} \neq \emptyset$.

The removal under (3) is possible because such a rule will never be applied, since the C_i will not be present in the sequent.

Theorem 4.2.7 Let $S = (P, \mathcal{R})$ define a sequent calculus. Let $\vdash C_1, \ldots, C_r$ be a sequent. Then $\vdash C_1, \ldots, C_r$ is provable in S iff the empty sequent $\vdash$ is provable in $S(C_1, \ldots, C_r)$.

Proof
Assume that $\vdash C_1, \ldots, C_r$ is provable in S. We prove the following induction hypothesis:

IH1 If a sequent $\vdash \Gamma$ is provable in S, then $\vdash \Gamma \backslash \{C_1, \ldots, C_r\}$ is provable in $S(C_1, \ldots, C_r)$.

- Suppose we have that a sequent is obtained by application of a logical rule.

$$\frac{\begin{array}{l}\vdash \Gamma_1, A_1, \\ \cdots \\ \vdash \Gamma_p, A_p,\end{array}}{\vdash \Gamma_1, \ldots, \Gamma_p, B_1, \ldots, B_q}$$

  By IH1 all

$$\begin{array}{l}\vdash (\Gamma_1, A_1) \backslash \{C_1, \ldots, C_r\}, \\ \cdots \\ \vdash (\Gamma_p, A_p) \backslash \{C_1, \ldots, C_r\}.\end{array}$$

  are provable. Now there are 2 possibilities:

  1. If an $A_i \in \{C_1, \ldots, C_r\}$, then $(\Gamma_i, A_i) \backslash \{C_1, \ldots, C_r\}$ can be weakened into $\vdash (\Gamma_1, \ldots, \Gamma_p, B_1, \ldots, B_q) \backslash \{C_1, \ldots, C_r\}$.
  2. If all $A_i \notin \{C_1, \ldots, C_r\}$, then we can introduce the following proof step.

$$\frac{\begin{array}{l}\vdash \Gamma_1 \backslash \{C_1, \ldots, C_r\}, A_1, \\ \cdots \\ \vdash \Gamma_p \backslash \{C_1, \ldots, C_r\}, A_p,\end{array}}{\begin{array}{l}\vdash (\Gamma_1 \backslash \{C_1, \ldots, C_r\}), \ldots, (\Gamma_p \backslash \{C_1, \ldots, C_r\}), \\ (\{B_1, \ldots, B_q\} \backslash \{C_1, \ldots, C_r\})\end{array}}$$

The conclusion is equal to

$$\vdash (\Gamma_1, \ldots, \Gamma_p, B_1, \ldots, B_q) \backslash \{C_1, \ldots, C_r\}.$$

- Now assume that $\vdash \Gamma, A$ is derived by weakening from $\vdash \Gamma$. Then by IH1, the sequent $\vdash \Gamma \backslash \{C_1, \ldots, C_r\}$ is derivable, which can be weakened in 0 or 1 step into $\vdash (\Gamma, A) \backslash \{C_1, \ldots, C_r\}$.

For the converse we use the following induction hypothesis:

IH2 If any sequent $\vdash \Gamma$ is provable in $S(C_1, \ldots, C_r)$, then $\vdash \Gamma, C_1, \ldots, C_r$ is provable in S.

- If $\vdash B_1, \ldots, B_q$ is an axiom of $S(C_1, \ldots, C_r)$, then there must be an axiom $\vdash B_1, \ldots, B_q, \Delta$ in S, for which $\Delta \subseteq \{C_1, \ldots, C_r\}$.
  Then $\vdash B_1, \ldots, B_q, C_1, \ldots, C_r$ can be obtained by weakening.

- Suppose that $\Gamma_1, \ldots, \Gamma_p, B_1, \ldots, B_q$ is obtained as follows in $S(C_1, \ldots, C_r)$:

$$\frac{\begin{array}{l}\vdash \Gamma_1, A_1, \\ \ldots \\ \vdash \Gamma_p, A_p,\end{array}}{\vdash \Gamma_1, \ldots, \Gamma_p, B_1, \ldots, B_q.}$$

  (Here $p > 0$). Then the rule $\{A_1, \ldots, A_p\} \vdash \{B_1, \ldots, B_q\}$ is used. There must be a rule $\{A_1, \ldots, A_p\} \vdash \{B_1, \ldots, B_q\} \cup \Delta$ in S, such that $\Delta \subseteq \{C_1, \ldots, C_r\}$. Then, using IH2 the following proof step is possible in S :

$$\frac{\begin{array}{l}\vdash \Gamma_1, A_1, C_1, \ldots, C_r \\ \ldots \\ \vdash \Gamma_p, A_p, C_1, \ldots, C_r\end{array}}{\vdash \Gamma_1, \ldots, \Gamma_p, B_1, \ldots, B_q, \Delta, C_1, \ldots, C_r.}$$

  Because $p > 0$, all $C_1, \ldots, C_r$ are present in one of the premisses, and the conclusion will contain all the $C_1, \ldots, C_r$.

- Suppose that $\vdash \Gamma, A$ is derived in $S(C_1, \ldots, C_r)$ from $\vdash \Gamma$ by weakening. Then $\vdash \Gamma, C_1, \ldots, C_r$ can be derived in S. This can be weakened into $\vdash \Gamma, A, C_1, \ldots, C_r$ in 0 or 1 step.

End of Proof

We have now obtained the following two things.

1. Every sequent calculus stands in a close relation to a resolution calculus in which the differences A1, A2 and A3 are removed.
2. For every sequent calculus S, and goal sequent $\vdash \Gamma$ there is a sequent calculus $S(\Gamma)$ in which the empty sequent can be derived if and only if $\vdash \Gamma$ can be derived with S.

This can be combined into the following:

Theorem 4.2.8 Let $S = (P, \mathcal{R})$ be a sequent calculus. Let $\vdash C_1, \ldots, C_r$ be a sequent. The following 4 are equivalent:

1. The sequent $\vdash C_1, \ldots, C_r$ is derivable in S.
2. The sequent $\vdash$ is derivable in $S(C_1, \ldots, C_r)$.
3. A clause $c \subseteq \{C_1, \ldots, C_r\}$ is derivable from the axioms with the resolution rules based on S.
4. $\emptyset$ is derivable from the axioms with the resolution rules based on $S(C_1, \ldots, C_r)$.

Proof
The equivalence (1) $\Leftrightarrow$ (2) is given in Theorem 4.2.7. The equivalences (1) $\Leftrightarrow$ (3) and (2) $\Leftrightarrow$ (4) follow from Theorem 4.2.5.
End of Proof

This shows how a resolution calculus can be obtained from a sequent calculus, by making a few small adaptations. We give an example:

Example 4.2.9 Consider the calculus in Example 4.2.1. The sequent $\vdash \neg\ (a \wedge \neg\ a) \vee b$ has the following proof: Left are the sequents, right are the rules that have been used.

$$\frac{\vdash \neg\ a, a}{\vdash \neg\ a, \neg\ \neg\ a,} \qquad \{\} \vdash \{a, \neg\ a\},$$

$$\{a\} \vdash \{\neg\ \neg\ a\},$$

$$\frac{}{\vdash \neg\, a \vee \neg\, \neg\, a, \neg\, \neg\, a,} \qquad \{\neg\, a\} \vdash \{\neg\, a \vee \neg\, \neg\, a\},$$

$$\frac{}{\vdash \neg\, a \vee \neg\, \neg\, a,} \qquad \{\neg\, \neg\, a\} \vdash \{\neg\, a \vee \neg\, \neg\, a\},$$

$$\frac{}{\vdash \neg\, (a \wedge \neg\, a),} \qquad \{\neg\, a \vee \neg\, \neg\, a\} \vdash \neg\, (a \wedge \neg\, a)\},$$

$$\frac{}{\vdash \neg\, (a \wedge \neg\, a) \vee b.} \qquad \{\neg\, (a \wedge \neg\, a)\} \vdash \{\neg\, (a \wedge \neg\, a) \vee b\}.$$

In order to obtain $S(\{\neg\, (a \wedge \neg\, a) \vee b\})$ the last rule has to be replaced by $\{\neg\, (a \wedge \neg\, a)\} \vdash \emptyset$. It is obvious that with this last rule, the empty sequent $\vdash$ can be derived. The resolution refutation, which corresponds to the proof given above, is:

(1)	$\{\neg\, a, a\}$	(axiom)
(2)	$\{\neg\, a, \neg\, \neg\, a\},$	(from 1)
(3)	$\{\neg\, a \vee \neg\, \neg\, a, \neg\, \neg\, a\},$	(from 2)
(4)	$\{\neg\, a \vee \neg\, \neg\, a\},$	(from 3)
(5)	$\{\neg\, (a \wedge \neg\, a)\},$	(from 4)
(6)	$\{\neg\, (a \wedge \neg\, a) \vee b\}$	(from 5).

With resolution it is also possible to derive the empty clause if the last rule is changed.

In the last example formulae have been used in the sequent. It is also possible to replace the formulae by labels. This is done in ([Mints88]). Then the goal sequent can be replaced by a set of labels, and all rules can be replaced by rules on labels.
There is some variation in how labels can be assigned to subformulae. It is possible to assign labels to subformulae, and it is possible to assign them to occurrences of subformulae. In the latter case different occurrences of the same subformulae may receive a different label.

Example 4.2.10 If one wants to prove $\vdash \neg\, (a \wedge \neg\, a) \vee b$, then the following formulae may play a rôle:

$L1 = \quad \neg\, (a \wedge \neg\, a) \vee b$
$L2 = \quad \neg\, (a \wedge \neg\, a)$
$L3 = \quad b$
$L4 = \quad \neg\, a \vee \neg\, \neg\, a$
$L5 = \quad \neg\, a$
$L6 = \quad \neg\, \neg\, a$
$L7 = \quad a.$

Then one looks for a proof of $\vdash L1$, with the following set of rules:

$\{L3\} \vdash \{L1\}, \quad \{L2\} \vdash \{L1\}, \quad \{L4\} \vdash \{L2\},$
$\{L5\} \vdash \{L4\}, \quad \{L6\} \vdash \{L4\}, \quad \{L7\} \vdash \{L6\},$
$\vdash \{L5, L7\}.$

A possible proof is:

$\vdash L5, L7$
$\vdash L5, L6$
$\vdash L4, L6$
$\vdash L4$
$\vdash L2$
$\vdash L1.$

This proof can be easily written as a resolution proof.

Finally we should say something about substructural logics. The main problem with substructural logics lies in Theorem 4.2.6, because its proof relies heavily on contraction. It is in general possible to fined acceptable solutions for A1, A2, A3 in substructural logics. (See [Mints88] and [Mints93]).

4.3 Maslov's inverse method

The method in ([Mints88]) is a generalisation of the inverse method of Maslov. ([Maslov86]). It is easy to explain Maslov method with the means of the previous section.
This method is called inverse method, because in the propositional case, the best manner of searching for a proof is top down. One starts with the sequent that one wants to prove, and tries to find a possible previous step in the proof until one has reached the axioms. This method is not optimal in the case of predicate logic because it may introduce global veriables, as described in the introduction of this chapter. This problem can be avoided by starting with the axioms and trying to derive the goal sequent. This is the inverse direction.
Let $\vdash \Gamma$ be a sequent in classical propositional logic. We call $\vdash \Gamma$ in *DCD normal form* if Γ consists of formulae (a sequent can be read as a disjunction (D)) which are conjunctions (C) of disjunctions(D) of literals. (A literal is an A or $\neg A$, for an atom A). So the sequent $\vdash (a \vee c) \wedge b, \neg a, \neg b$ is in DCD normal form. Any sequent can be transformed into DCD.

If one wants to prove a sequent in DCD-normal form, the sequent calculus can be quite a lot simpler because all rules for the connectives that are not allowed can be removed.

Example 4.3.1 (Sequent calculus for sequents in DCD-normal form).

Axiom1 Every sequent of the form $\vdash A \vee \neg A \vee C_1 \vee \cdots \vee C_r$, with $r \geq 0$, is an axiom.

Axiom2 Every sequent of the form $\vdash A \vee C_1 \vee \cdots \vee C_r, \neg A \vee D_1 \vee \cdots \vee D_s$, is an axiom. $(r \geq 0, s \geq 0)$

$\wedge$-r If the sequents $\vdash \Gamma_1, A_1$ up to $\vdash \Gamma_p, A_p$ are derivable, then the sequent $\vdash \Gamma_1, \cdots, \Gamma_p, A_1 \wedge \cdots \wedge A_p$ is derivable.

Weakening If the sequent $\vdash \Gamma$ is derivable then the sequent $\vdash \Gamma, A$ is derivable.

It is easily seen that this calculus is sound and complete for formulae in DCD-normal form. We give a derivation of the the tautology

$$\vdash (a \vee b) \wedge c, \neg b, \neg c \wedge (a \vee \neg a).$$

Example 4.3.2 Derivation of $\vdash (a \vee b) \wedge c, \neg b, \neg c \wedge (a \vee \neg a)$.

$$\dfrac{\vdash a \vee \neg a \qquad \dfrac{\vdash (a \vee b), \neg b \qquad \vdash c, \neg c}{\vdash (a \vee b) \wedge c, \neg b, \neg c}}{\vdash (a \vee b) \wedge c, \neg b, \neg c \wedge (a \vee \neg a)}$$

Now Maslov's inverse method is obtained by introducing labels, and applying Theorem 4.2.8 to the calculus defined in Example 4.3.1 with the goal deletion rules. The soundness and completeness follows from Theorem 4.2.8. It is also possible to present Maslov's method as a variant of resolution. This is done in ([Mints88]) and ([Lifschitz87]).

4.4 From Semantic Tableaux to Resolution

In this section we discuss a general method for obtaining resolution calculi from semantic tableaux calculi which was developed in ([Nivelle93]) and ([Nivelle94]). Because the existence of semantic tableau calculi depends strongly on structural rules, and the existence of interpretations, the method

is not as general as the method of Section 4.2. The method differs however from the method of Section 4.2, because it, despite the fact that it is a local method, preserves the top down direction of semantic tableaux. (This will be discussed in detail in Section 4.6). Another difference is that it possible to obtain ordering refinements immediately. This will be proven in Section 4.5. We will prove in this section that the semantic tableaux of every semantic tableau calculus can be transformed into resolution refutations. In this way every complete semantic tableau calculus automatically defines a complete resolution calculus.
We also prove the well-known fact that the completeness of a semantic tableau calculus follows from the fact that open paths in a semantic tree are models. More precisely, if the semantic tableaux rules are designed in such a manner that every path that is closed under the rules has an interpretation, then the semantic tableau calculus is complete.
We begin by giving an example: Semantic tableaux for classical logic. After that we give the general scheme of a semantic tableau calculus, and prove the second statement: If every complete, closed path defines a model of the logic, then the semantic tableau calculus is complete.
After that we define the resolution rules that belong to a semantic tableau calculus, and give the transformation from closed semantic tableaux to resolution proofs.

Example 4.4.1 The semantic tableau method for classical logic is a *refutation method*. It tries to prove that a set of formulae is unsatisfiable, instead of proving that it is universally valid. Semantic tableaux for classical logic works as follows: Suppose that one looks for a refutation of the set of classical formulae $\{A_1, \ldots, A_p\}$. Then one starts with the linear tree $(n_1, \ldots, n_p)$, in which each n_i is labelled with A_i. After that the following extension rules are applied:

If a path contains

$A \wedge B$	then add	A and B	in this path,
$A \vee B$,	then split and add	A	in one path,
	and add	B	in the other path,
$A \rightarrow B$,	then add	$\neg\, A \vee B$	in this path
$\neg\, \neg\, A$,	then add	A	in this path,
$\neg\, (A \vee B)$,	then add	$\neg\, A \wedge \neg\, B$,	in this path,
$\neg\, (A \wedge B)$,	then add	$\neg\, A \vee \neg\, B$	in this path,
$\neg\, (A \rightarrow B)$,	then add	$A \wedge \neg\, B$,	in this path
$\neg\, A$ and A,	then the path is closed		

There rules should be interpreted as schemata, i.e., A and B are meta-

variables which represent formulae.
If it is possible to construct a semantic tableau in which all paths are closed then the set of formulae is refuted.

The rules of Example 4.4.1 have the following form:

> $\{A_1, \ldots, A_p\} \vdash \{B_1, \ldots, B_q\}$, which means:
> If a path contains all of the A_i then continue in q paths. In each j-th path $(1 \leq j \leq q)$, B_j is added. It must be the case that $p > 0$ and $q \geq 0$.

The first rule-schema in Example 4.4.1 is in fact two rule-schemata: $\{A \wedge B\} \vdash \{A\}$, and $\{A \wedge B\} \vdash \{B\}$. The second corresponds to $\{A \vee B\} \vdash \{A, B\}$. The last rule schema corresponds to $\{A, \neg\ A\} \vdash \{\}$. The other rule-schemata are straightforward. For example the third rule corresponds to $\{A \rightarrow B\} \vdash \{\neg\ A \vee B\}$. We will use the general form above to define semantic tableaux:

Definition 4.4.2 A (propositional) *semantic tableau calculus* is a pair $S = (P, \mathcal{R})$, where P is a set of propositional symbols, and $\mathcal{R}$ is a set of rules of the form

$$\{A_1, \ldots, A_p\} \vdash \{B_1, \ldots, B_q\}.$$

(Here $p > 0$, and $q \geq 0$). A *formula of* S is an element of P. A *clause of* S is a finite set of elements of P.

The set of rules is intended to be designed in such a manner that a closed set defines an interpretation of the logic. For this reason we call a closed set an intepretation:

Definition 4.4.3 Let S be a semantic tableau calculus. A set I is *an interpretation of* S iff
whenever for a rule $\{A_1, \ldots, A_p\} \vdash \{B_1, \ldots, B_q\} \in \mathcal{R}$,

$$\{A_1, \ldots, A_p\} \subseteq I,$$

then

$$\{B_1, \ldots, B_q\} \cap I \neq \emptyset.$$

If c is a clause of S then an interpretation I of S is a *model* of c if $I \cap c \neq \emptyset$. An interpretation I is a *model of a formula* A if $A \in I$. An interpretation is a model of a set of clauses if it is a model of every clause in the set.

Definition 4.4.4 A set of clauses, or formulae is *consistent* if it has a model. It is *inconsistent* if it has no model.

We now define formally how to construct a semantic tableau using a given semantic tableau calculus and a set of clauses that one wants to refute. We use the definition of labelled tree of Definition 2.3.1.

Definition 4.4.5 Let $S = (P, \mathcal{R})$ be a semantic tableau calculus. Let C be a set of clauses. A *semantic tableau* is a labelled tree $(N, <, \Lambda)$, in which Λ labels the nodes of N with formulae of P. The tree has to be the result of the following process:

- The tree $(\emptyset, \emptyset, \emptyset)$ (the empty tree with the empty label function) is a semantic tableau.
- Every semantic tableau $(N, <, \Lambda)$ can be extended by extending a path J of $T = (N, <)$ according to one of the following rules:
  1. It is always possible, for any clause $\{A_1, \ldots, A_p\} \in C$, to extend J in p directions with new nodes $n_1, \ldots, n_p$, which are labelled with $\Lambda(n_i) = A_i$.
  2. If there is a rule $\{A_1, \ldots, A_p\} \vdash \{B_1, \ldots, B_q\}$, with $q > 0$, such that all A_i occur as labels of J, then J can be extended in q directions with new nodes $n_1, \ldots, n_q$, with labels $\Lambda(n_j) = B_j$.

Semantic tableaux are always finite. A semantic tableau is *closed* if for every path J of T there is a rule $\{A_1, \ldots, A_p\} \vdash \emptyset$, such that all A_i occur as labels of J. A semantic tableau that is not closed is called *open*. For a set of formulae $\{A_1, \ldots, A_p\}$, we define the semantic tableau as the semantic tableau of $\{\{A_1\}, \ldots, \{A_p\}\}$.

We will prove the soundness and completeness of semantic tableaux. First the soundness:

Lemma 4.4.6 Let $S = (P, \mathcal{R})$ be semantic tableau calculus. Let C be a set of clauses. C is inconsistent if there exists a closed semantic tableau of C.

Proof
We prove that if there exists a closed semantic tableau based on C, then C is inconsistent. Suppose there exists a closed semantic tableau $(N, <, \Lambda)$ of C. We prove that for every set $I \subseteq P$, which satisfies

1. For all $c \in C$, $c \cap I \neq \emptyset$.

2. If for a rule $\{A_1, \ldots, A_p\} \vdash \{B_1, \ldots, B_q\}$ with $q > 0$,

$$\{A_1, \ldots, A_p\} \subseteq I, \text{ then}$$

$$\{B_1, \ldots, B_q\} \cap I \neq \emptyset,$$

there is a path of J, such that $\Lambda(J) \subseteq J$. (Here we write $\Lambda(J) = \{\Lambda(n) \mid n \in I\}$) Then, because the semantic tableau is closed there must be a rule $\{A_1, \ldots, A_p\} \vdash \emptyset$, such that I contains all the A_i. Because of this I cannot be a model of C.
Let I be a set satisfying (1) and (2). The path J will be constructed inductively as $J_0, J_1, J_2, \ldots$, with all $\Lambda(J_i) \subseteq I$. Define $J_0 = \emptyset$. As long as J_{i+1} is not a complete path of the tree proceed as follows: Let $\{n_1, \ldots, n_q\}$ be the successor nodes of J_i.

- If they are added because of rule 1 in Definition 4.4.5, then let c be the clause that is used. Because $c \cap I \neq \emptyset$, one of the n_j must have a label $\Lambda(n_j) \in (c \cap I)$. Put $J_{i+1} = J_i \cup \{n_j\}$.
- If they are added because of rule 2 in Definition 4.4.5, then let $\{A_1, \ldots, A_p\} \vdash \{B_1, \ldots, B_q\}$ be the rule that is used. (with $q > 0$) All A_i must be present as labels of J_i, and hence be in I. Then one B_j must be in I. Put $J_{i+1} = J_i \cup \{n_j\}$.

In both cases $\Lambda(J_{i+1}) \subseteq I$.
End of Proof
The proof of the completeness is more complicated. For this we have to introduce the notion of semantic tree. A semantic tree is essentially a semantic tableau, but without the finiteness condition and with a fairness condition.

Definition 4.4.7 A *transfinite semantic tableau* can be obtained by adding a third rule in Definition 4.4.5. This third rule makes it possible to obtain transfinite semantic tableaux by taking limits of sequences of semantic tableaux. The rule is:

3. Let $(N_0, <_0, \Lambda_0), \ldots, (N_i, <_i, \Lambda_i), \ldots$ up to α, be an increasing sequence of transfinite semantic tableaux, i.e.

$$N_0 \subseteq N_1 \subseteq \cdots \subseteq N_i \subseteq \cdots,$$

$$<_0 \subseteq <_1 \subseteq \cdots \subseteq <_i \subseteq \cdots,$$

$$\Lambda_0 \subseteq \Lambda_1 \subseteq \cdots \subseteq \Lambda_i \subseteq \cdots.$$

Then $(\bigcup_{i<\alpha} N_i, \bigcup_{i<\alpha} <_i, \bigcup_{i<\alpha} \Lambda_i)$ is a transfinite semantic tableau.

A transfinite semantic tableau $(N, <, \Lambda)$ is called a *semantic tree* if it satisfies the following fairness conditions:

1. For every path J of $(N, <, \Lambda)$, for every clause $\{A_1, \ldots, A_p\}$, one of the A_i occurs as a label of J.
2. For every path J of $(N, <, \Lambda)$, for every rule $\{A_1, \ldots, A_p\} \vdash \{B_1, \ldots, B_q\}$, with $q > 0$, such that all A_i occur as a label of J, one of the B_j occurs as a label of J.

A semantic tree $(N, <, \Lambda)$ is *closed* if for every path of $(N, <)$ there is a rule of the form $\{A_1, \ldots, A_p\} \vdash$, such that all A_i occur as labels of the path. A semantic tree which is not closed is called *open*.

We give the completeness proof in 2 steps. The first step is to prove that there always exists a semantic tree. This tree is created by systematically checking all possibilities to apply a clause or a rule. For every inconsistent set of clauses, every semantic tree is closed. The second step is to show that a finite closed semantic tableau can be extracted from a closed semantic tree. First we construct a semantic tree:

Lemma 4.4.8 Let $S = (P, \mathcal{R})$ be a semantic tableau calculus. Let C be a set of clauses, not containing the empty clause. There exists a semantic tree of C.

Proof
Assume that the clauses of C, and the rules of $\mathcal{R}$ are well-ordered. Let α be the ordinality of C and let β be the ordinality of $\mathcal{R}$, under the given orderings.

We construct the labelled tree $T = (N, <, \Lambda)$, in length $\alpha + \omega \times \beta$.

- The construction starts with the labelled tree $(\emptyset, \emptyset, \emptyset)$.
- After that there will be an iteration up to α. For every $i, 0 \leq i < \alpha$, the $i+1$-th transfinite semantic tableau will be obtained from the i-th transfinite semantic tableau by branching on clause c_i in every path. For every limit ordinal i, with $0 \leq i \leq \alpha$, the i-th transfinite semantic tableau will be taken as the limit of its predecessors.
- After that the rules will be applied. The following iteration is repeated ω times:
  - For every i, with $0 \leq i < \beta$, the $i+1$-th semantic tree is obtained from the i-th semantic tree by branching on the i-th rule in every path that contains the premisses of this rule. The others paths are not affected.

– For every limit ordinal i, with $0 \leq i < \beta$, the i-th semantic tree is taken as the limit of all its predecessors.

It is not difficult to see that the result $(N, <, \Lambda)$ is as promised: In every path of the tree, an element of each clause has been added. If a path contains all $\{A_1, \ldots, A_p\}$ of a rule $\{A_1, \ldots, A_p\} \vdash \{B_1, \ldots, B_q\}$, then in one of the iterations up to ω a B_j will be added in that path.
End of Proof

If a clause set C is inconsistent, then every semantic tree, based on C will be closed, because an open path of a semantic tree would define a model of C.
Only a finite part of a closed semantic tree is actually used: If we throw away the rest, the remaining part is a closed semantic tableau. In order to be able to do this we need the following notion:

Definition 4.4.9 Let $T = (N, <)$ be a tree. We define the *kernel* of T (written as $\overline{T}$) as the tree that is the result of the following iteration:

1. $\overline{T}_0 = T$.
2. $\overline{T}_{i+1}$ is obtained from $\overline{T}_i$ by deleting the end node from every path J of $\overline{T}_i$ if it has an end node.
3. $\overline{T} = \bigcap \overline{T}_i$.

The kernel is the empty tree iff the tree is finite. Now this kernel has the following property:

Lemma 4.4.10 No path J of $\overline{T}$ has an end node. (Formally: For every $n_1 \in J$, there is an $n_2 \in J$, such that $n_1 < n_2$)

Proof
Suppose that a node n is an end node of a path J of $\overline{T}$. Then there is a moment i in the construction of the $\overline{T}$, where J is an end node of $\overline{T}_i$, because T is finitely branching. Then at stage $i + 1$ node n would have been removed from $\overline{T}_{i+1}$. This is a contradiction.
End of Proof

Lemma 4.4.11 If $(N, <, \Lambda)$ is a closed, transfinite semantic tableau, based on C and S, then there is a closed (finite) semantic tableau based on C and S.

Proof
We will construct a finite semantic tableau from a transfinite semantic tableau $(N, <, \Lambda)$, by iteratively reducing the kernel of $(N, <)$, using Lemma 2.3.3. Let $T = (N, <, \Lambda)$, be a transfinite semantic tableau. Let $\overline{T} = (\overline{N}, <)$ be the kernel of T. We will construct a new semantic tableau $T' = (N', <, \Lambda')$, with kernel $\overline{T}' = (\overline{N}', <)$, such that

1. $\overline{N}' \subseteq \overline{N}$,
2. $\overline{N}'$ is an initial subtree of $\overline{N}$,
3. Every path J of $\overline{T}$ is not a path of $\overline{T}'$. (because it is cut off)

This reduction clearly satisfies the conditions of Lemma 2.3.3, and hence it will terminate in a finite number of steps. The resulting semantic tableau will be finite.
Now let J be (non-empty) path of $\overline{T}$. We will define a method to obtain an inital segment $J' \subset$ from each J. After that $\overline{T}'$ will be constructed as the tree for which its paths are the minimal J'. So: If J' is a path of $\overline{T}'$, then J', is obtained as an initial segment from a path of $\overline{T}$, and there is no path J_1 of $\overline{T}$, for which $J'_1 \subset J'$.
We define how to obtain J' from J. We distinguish two cases:

- $T(J) = (\emptyset, <)$. Then J is a complete path of T. Then J has as labels all $A_1, \ldots, A_p$, of a rule $\{A_1, \ldots, A_p\} \vdash \emptyset$. We define J' as the smallest initial segment of J which contains all A_i. Because J has as length a limit ordinal, J' is a strict subset of J.

- $T(J) \neq (\emptyset, <)$. Then $T(J)$ must be finite. There are only a finite number of nodes in $T(J)$. Every node in $T(J)$ is derived by either an application of rule (1) or (2) in Definition 4.4.5. In case (1) the possibility to apply the rule is completely independent of J. In case (2) the possibility depends only on the fact that the labels $A_1, \ldots, A_p$ that are not present in $T(J)$, must be present in J. As a consequence the possibility to construct $T(J)$ depends only on a finite number of nodes of J. We define J' as the smallest initial segment of J which contains all these nodes. J' is a strict subset of J because J has as length a limit ordinal.

End of Proof

We have proven the completeness of semantic tableaux. So we state:

Lemma 4.4.12 If $S = (P, \mathcal{R})$ is a semantic tableau calculus, and C is an inconsistent set of clauses, then there exists a closed semantic tableau that refutes C.

We will now give the resolution rules which belong to a semantic tableaux calculus, and give the transformation from semantic tableaux to resolution.

Definition 4.4.13 Let $S = (P, \mathcal{R})$ be a semantic tableau calculus. The *resolution rules* belonging to this calculus are obtained as follows: If S contains a rule $\{A_1, \ldots, A_p\} \vdash \{B_1, \ldots, B_q\}$, and $c_1, \ldots, c_p$ are clauses of S, such that $A_1 \in c_1, \ldots, A_p \in c_p$, then the following clause is a resolvent of $c_1, \ldots, c_p$, making use of the rule r:

$$(c_1 \backslash \{A_1\}) \cup \cdots \cup (c_p \backslash \{A_p\}) \cup \{B_1, \ldots, B_q\}.$$

With the transformation we will prove the following:

Theorem 4.4.14 If $S = (P, \mathcal{R})$ is a semantic tableau calculus, and C is a set of clauses of S, then the following 3 are equivalent:

1. C is inconsistent,
2. C has a semantic tableau refutation, (based on S.)
3. C has a resolution refutation, (based on S.)

Proof
The equivalence **(1)** $\Leftrightarrow$ **(2)** is already stated in Lemma 4.4.6 and Lemma 4.4.12. It remains to prove the equivalence with **(3)**. We first prove that if C is consistent then C has no resolution refutation in Lemma 4.4.15. This is done by showing that if a set clauses C has a model I, then this model is also a model of all clauses that can be derived by resolution. As a consequence the empty clause cannot be derived. After that we prove that **(2)** $\Rightarrow$ **(3)** using the transformation from semantic tableaux to resolution.
End of Proof

Lemma 4.4.15 Let $S = (P, \mathcal{R})$ be a semantic tableau calculus. Let $\{A_1, \ldots, A_p\} \vdash \{B_1, \ldots, B_q\}$ be a rule, let $c_1, \ldots, c_p$ be clauses, such that $A_1 \in c_1, \ldots, A_p \in c_p$. Let I be an interpretation of S, such that for all c_i, $c_i \cap I \neq \emptyset$. Let c be the clause that can be obtained by resolution:

$$c = (c_1 \backslash \{A_1\}) \cup \cdots \cup (c_p \backslash \{A_p\}) \cup \{B_1, \ldots, B_q\}.$$

Then $c \cap I \neq \emptyset$.

Proof
We distinguish two cases:

- All $A_i \in I$: Then, because of the definition of interpretation $\{B_1, \ldots, B_q\} \cap I \neq \emptyset$, and hence $c \cap I \neq \emptyset$.
- One $A_i \notin I$: Then, because I is a model of c_i, it must be the case that $I \cap (c_i \backslash \{A_i\}) \neq \emptyset$. This implies $(c \cap I) \neq \emptyset$, because $c_i \backslash \{A_i\} \subseteq c$.

End of Proof

The proof of the other direction is more involved. We show that every closed semantic tableau can be transformed into a resolution proof.

Theorem 4.4.16 Let $T = (N, <, \Lambda)$ be a closed semantic tableau based on a set of clauses C and semantic tableau rules $S = (P, \mathcal{R})$. There is a general transformation from T into a resolution refutation of C using resolution rules based on S.

Proof
In the proof we will use a property of resolution which is called the *combination property*. It says that if the empty clause can be derived from a set $C \cup \{X\}$, and the empty clause can be derived from $C \cup \{Y\}$, then the empty clause can be derived from $C \cup \{X \cup Y\}$. It will be proven in Lemma 4.4.17. We use it to combine refutations of simpler clause sets into refutations of more complex clause sets.
Let $(n_1, \ldots, n_l)$ be the largest initial segment of a path that is common in all paths of T. If T does not branch, then $(n_1, \ldots, n_l)$ equals the complete tree. If T has multiple roots, then $l = 0$.
We use this fragment for our transformation. First we prove two things: The first is:

- For all labels $\Lambda(n_i)$, $(1 \leq i \leq l)$, the clause $\{\Lambda(n_i)\}$ can be derived.

If n_i is derived by rule 1 of Definition 4.4.5, then this is because of singleton clause $\{\Lambda(n_i)\} \in C$, and $\{\Lambda(n_i)\}$ is present as initial clause. If n_i is added because of rule 2 of Definition 4.4.5, then this is because of a rule $\{A_1, \ldots, A_p\} \vdash \{B\}$, and the fact that all A_i are present as labels in the tree. Then all singleton clauses $\{A_1\}, \ldots, \{A_p\}$ can be derived, and hence, by resolution $\{B\}$ can be derived.
The second thing to prove is:

- If $(n_1, \ldots, n_l)$ is not equal to the complete tree, and $m_1, \ldots, m_q$ are the immediate successors of n_l, then the clause $\{\Lambda(m_1), \ldots, \Lambda(m_q)\}$ can be derived.

The proof is essentially the same as the previous one. If the $m_1, \ldots, m_q$ are added in an application of rule (1), then the clause $\{\Lambda(m_1), \ldots, \Lambda(m_q)\}$ is present as initial clause. If the $m_1, \ldots, m_q$ are added in an application of rule (2), then there is a rule $\{A_1, \ldots, A_p\} \vdash \{B_1, \ldots, B_q\}$ such that all $\{A_1\}, \ldots, \{A_p\}$ are present as labels in the tree. Then $\{B_1, \ldots, B_q\}$ can be derived in one application of the resolution rule.
Using this we can now give the transformation:

Transformation

- If the tree is linear, then $T = (n_1, \ldots, n_l)$, and all the labels of the n_i can be derived as singleton clauses. Because T is closed, there is a rule $\{A_1, \ldots, A_p\} \vdash \emptyset$, such that all A_i are present als labels in T. Because the clauses $\{A_1\}, \ldots, \{A_p\}$ can be derived, $\emptyset$ can be derived.
- If the tree is not linear, let $m_1, \ldots, m_q$ be the successor nodes of n_l. We make the following resolution derivations. Let $m_1, \ldots, m_q$ be the successor nodes of n_l. We define, for all i, with $1 \leq i \leq q$, the clause sets $C_i = C \cup \{\Lambda(m_i)\}$. Then each semantic tableau $(n_1, \ldots, n_m, m_i)$ is a correct semantic tableau that refutes the clause set C_i, becuase m_i is obtained with an application of rule 1, and the other applications of rules are unchanged. By induction, we can construct the resolution refutations of the clause sets C_i with S. After that we apply the combination property q times to obtain a resolution refutation of the clause set $C \cup \{\Lambda(m_1), \ldots, \Lambda(m_q)\}$. Because $\{\Lambda(m_1), \ldots, \Lambda(m_q\}$ can be derived from C, we have constructed a resolution refutation from C.

End of Proof

It remains to prove combination.

Lemma 4.4.17 (Combination Lemma) Let $S = (P, \mathcal{R})$ be a semantic tableau rule calculus. Let C be a set of clauses. Let X_1 and Y_1 be clauses. If

1. it is possible to derive $\emptyset$ from $C \cup \{X_1\}$, and
2. it is possible to derive $\emptyset$ from $C \cup \{Y_1\}$, then

it is possible to derive $\emptyset$ from $C \cup \{X_1 \cup Y_1\}$. Moreover this derivation can be constructed from the derivations (1) and (2).

Proof
The idea is not difficult at all. Let $X_1, \ldots, X_n$ be a derivation of $\emptyset$ from C together with X_1. We have $X_n = \emptyset$, and each X_i is derived by one resolution step from $C, X_1, \ldots, X_{i-1}$. The obvious thing to do is to replace each X_i by $X_i \cup Y_1$. The resulting derivation would be:

$$X_1 \cup Y_1, X_2 \cup Y_1, \ldots, X_n \cup Y_1 = Y_1, Y_2, \ldots, Y_m = \emptyset.$$

When a rule is applied on clauses $c_1 \cup Y_1, \ldots, c_p \cup Y_1$ instead of $c_1, \ldots, c_p$, then the extra Y_1 can be copied into the resolvent. Unfortunately it may be possible that X_i and Y_1 overlap. In that case when a literal $A \in X_i \cap Y_1$ is used, A will be deleted and the result X_{i+1} cannot be written as $X_{i+1} \cup Y_1$. So all $X_i \cup Y_1$ have to be replaced by $X_i \cup Y_i'$ for a $Y_i' \subseteq Y_1$. We prove that this yields a correct resolution refutation:
Let $r = \{A_1, \ldots, A_p\} \vdash \{B_1, \ldots, B_q\}$ be a rule, let $c_1, \ldots, c_p$ be clauses, such that $c_1, \ldots, c_p$ can resolve to d, using r. Let $Y_1', \ldots, Y_p'$ be subsets of Y_1. Then $c_1 \cup Y_1', \ldots, c_p \cup Y_p'$ resolve into a subset of $d \cup Y_1$, using r. We have all $A_i \in c_i$,

$$d = (c_1 \backslash \{A_1\}) \cup \cdots \cup (c_p \backslash \{A_p\}) \cup \{B_1, \ldots, B_q\}.$$

Because of this we have all $A_i \in (c_i \cup Y_1)$, and the $c_i \cup Y_i'$ resolve into

$$(c_1 \cup Y_1') \backslash \{A_1\} \cup \cdots \cup (c_p \cup Y_p') \backslash \{A_p\} \cup \{B_1, \ldots, B_q\},$$

which is a subset of $d \cup Y_1$.

A possible consequence is that it may be necessary to derive $\emptyset$ using a subset of Y_1 instead of from Y_1 itself. We need a proof that this will not affect completeness, and that it is possible to construct the refutation of the subset of Y_1 from the refutation of Y_1. This property will be proven separately. It is called the subsumption property.
End of Proof

Definition 4.4.18 Let X and Y be two clauses. We say that X *subsumes* Y iff $X \subseteq Y$.

When we have proven the subsumption property we have the complete transformation. Some care has to be taken in the formulation of the subsumption property. The obvious is:

- If from $C \cup \{X\}$ the empty clause can be derived and Y subsumes X, then from $C \cup \{Y\}$ the empty clause can be derived.

This would be sufficient for the proof of Lemma 4.4.17, but we want to be able to use subsumption also in a different context. We want to be able to use subsumption in a refutation as often as possible. In order not to lose completeness in that case it is necessary to add the condition that the derivation of the empty clause from $C \cup \{Y\}$ will not be longer than the derivation of the empty clause from $C \cup \{X\}$. Otherwise it would be possible to postpone derivation of the empty clause every time subsumption is used. By using subsumption infinitely often it might be possible to postpone derivation of the empty clause forever. So the proof length will be limited:

Lemma 4.4.19 (Subsumption) If the empty clause can be derived from $C \cup \{X\}$ in n steps, and Y subsumes X, then the empty clause can be derived from $C \cup \{Y\}$ in not more than n steps.

Proof
It is sufficient to prove:

- If c can be derived from $c_1, \ldots, c_p$ with rule $\{A_1, \ldots, A_p\} \vdash \{B_1, \ldots, B_q\}$ in 1 step, and $d_1 \subseteq c_1, \ldots, d_p \subseteq c_p$, then a clause $d \subseteq c$ can be derived from $d_1, \ldots, d_p$ in 0 or 1 step.

It must be the case that $A_1 \in c_1, \ldots, A_p \in c_p$, and $c = (c_1 \backslash \{A_1\}) \cup \cdots \cup (c_p \backslash \{A_p\}) \cup \{B_1, \ldots, B_q\}$. We distinguish:

1. All $A_i \in d_i$. Then
$$d = [(d_1 \backslash \{A_1\}) \cup \cdots \cup (d_p \backslash \{A_p\}) \cup \{B_1, \ldots, B_q\}] \subseteq c.$$
2. One $A_i \notin d_i$. Then $d_i \subseteq (c_i \backslash \{A_i\}) \subseteq c$.

End of Proof

This completes the transformation from semantic tableaux to resolution.

Example 4.4.20 Let $S = (P, \mathcal{R})$ be defined as follows: $P = \{a, b, c, d\}$, and $\mathcal{R} = \{\ \{a\} \vdash \{b, c\},\ \{c\} \vdash \{\},\ \{b\} \vdash \{d\},\ \{b, d\} \vdash \emptyset\ \}$. The set $I = \{d\}$ is an interpretation of S. The clause $\{a, b\}$ is inconsistent. The following closed semantic tableau is possible.

$$\frac{b \qquad\qquad c}{d} \qquad\qquad d$$

This can be written as a labeled tree $T = (N, <, \Lambda)$ with

- $N = \{n_0, n_1, n_2, n_3, n_4\}$,
- $<$ is defined from

$$n_0 < n_1,\ n_0 < n_2,\ n_1 < n_3,\ n_4 < n_5$$

- $\Lambda(n_0) = a,\ \Lambda(n_1) = b,\ \Lambda(n_2) = c,\ \Lambda(n_3) = d,\ \Lambda(n_4) = b,\ \Lambda(n_5) = d.$

Note that the tree has multiple roots, and that the nodes are labelled instead of the branches. In order to construct a resolution refutation of the clause set $\{\{a, b\}\}$ it is necessary to combine resolution refutations of $\{\{a\}\}$ and $\{\{b\}\}$. In order to obtain a refutation of $\{\{a\}\}$ it is necessary to combine the resolution refutations of $\{\{a\}, \{b\}\}$ and $\{\{a\}, \{c\}\}$. We give the refutations of $\{\{a\}, \{b\}\}$ and $\{\{a\}, \{c\}\}$:

(1)	$\{a\}$	(initial)	(1)	$\{a\}$	(initial)
(2)	$\{b\}$	(initial)	(2)	$\{c\}$	(initial)
(3)	$\{d\}$	(from 2)	(3)	$\{\}$	(from 2)
(4)	$\{\}$	(from 2,3)			

This can be combined into a refutation of $\{\{a\}, \{b, c\}\}$. This refutation is given in the next table, together with a refutation of $\{\{b\}\}$.

(1)	$\{a\}$	(initial)	(1)	$\{b\}$	(initial)
(2)	$\{b, c\}$	(initial)	(2)	$\{d\}$	(from 1)
(3)	$\{d, c\}$	(from 2)	(3)	$\{\}$ (from 2)	
(4)	$\{c\}$	(from 2,3)			
(5)	$\{\}$	(from 4)			

These two can be combined into a refutation of $\{a, b\}$:

(1)	$\{a, b\}$	(initial)
(2)	$\{b, c\}$	(from 1)
(3)	$\{d, c\}$	(from 2)
(4)	$\{c\}$	(from 3)
(5)	$\{\}$	(from 4)

In the second clause b is already present. Therefore it is necessary to apply subsumption.

4.5 Refinements

In this section we will prove the completeness of 2 important refinements of resolution, namely subsumption and ordered resolution. It is also possible to combine ordered resolution and subsumption. We prove the completeness of ordered resolution by modifying the transformation of the previous section. The completeness of subsumption will be proven using Lemma 4.4.19.
Subsumption has already been used in the proof of Lemma 4.4.17. Here we discuss subsumption as a refinement. It is not necessary to limit the applications of subsumption to the places that are dictated by the proof of Lemma 4.4.17. It is possible to apply subsumption every time when there are two clauses c_1 and c_2, such that $c_1 \subseteq c_2$. In that case it is always possible to throw away c_2. This is safe because every derivation of the empty clause that uses c_2 can be replaced by a derivation, which is not longer, that uses c_1. In the literature, the following uses of subsumption are distinguished:

Forward Subsumption When a clause is derived, which is subsumed by a clause which is already present in the database, then the new clause is not kept.

Backward Subsumption When a new clause is derived, which subsumes a clause which is already present in the database, then the old clause is thrown away.

Full Subsumption When a clause c_1 is derived, which subsumes an older clause c_2, then it is possible to replace all clauses that are derived using c_2 by clauses that are derived from c_1. This process is called full subsumption.

Of course full subsumption is a more restrictive refinement than backward subsumption, but it is also more expensive, and more difficult to implement from the programmers point of view. It will be necessary to keep track, for all clauses, how they are derived. It is important to realise that subsumption is a very effective refinement. Computer experiments give tremendous improvement using subsumption.
Next we discuss ordered resolution. Let us first define it before discussing it:

Definition 4.5.1 Let $\prec$ be an order, let $S = (P, \mathcal{R})$ be semantic tableau rules. Let $c_1, \ldots, c_p$ be clauses. We call an element $A_i \in c_i$ *maximal* in c_i if for no $A \in c_i$, it occurs that $A_i \prec A$. Using this we call a clause c is an *ordered resolvent* of $c_1, \ldots, c_p$ if c is a resolvent, and all A_i are maximal in their respective c_i. (See Definition 4.4.13).

This is a restriction of the resolution rule, because sometimes the resolution steps are blocked, but when they are not blocked they are computed in the usual way.
Because of this the soundness of ordered resolution is immediate. For the completeness of ordered resolution we want to apply the construction of Section 4.4, so we need the following:

1. A proof that any immediate consequence of a set of singleton clauses can be computed. This is easy to see, because the elements of singleton clauses are maximal.

2. An ordered Combination lemma.

3. An ordered Subsumption lemma.

For the ordered subsumption lemma only a small modification in the proof is needed.

Lemma 4.5.2 (Ordered subsumption) If the empty clause can be derived from $C \cup \{X_1\}$ in n steps, and $Y_1 \subseteq X_1$, then the empty clause can be derived from $C \cup \{Y_1\}$ in at most n steps.

Proof
The proof of lemma 4.4.19, can be almost copied, but we have to prove that in case 1, the derivation of d is possible. For this it is sufficient to note that if A_i is maximal in $c_i, A_i \in d_i$, and $d_i \subseteq c_i$, then A_i is maximal in d_i.
End of Proof

As a consequence all of the abovementioned subsumption refinements will be compatible with ordered resolution.

Lemma 4.5.3 (Ordered Combination) Let $S = (P, \mathcal{R})$ be a semantic tableau calculus, and let $\prec$ be a total order on P. Let C be a set of clauses, and let X_1 and Y_1 be clauses. If

1. it is possible to derive $\emptyset$ from $C \cup \{X_1\}$ using $\prec$, and

2. it is possible to derive $\emptyset$ from $C \cup \{Y_1\}$ using $\prec$,

then it is possible to derive $\emptyset$ from $C \cup \{X_1 \cup Y_1\}$. Moreover this derivation can be constructed from the derivations (1) and (2).

Proof
We recursively define a function Σ which constructs the derivation.
$\Sigma(C; i; X_1, \ldots, X_n; Y_1, \ldots, Y_m)$ is defined when $C, i, X_1, \ldots, X_n, Y_1, \ldots, Y_m$ satisfy the following conditions:

1. $C, X_1, \ldots, X_n$ is a derivation of $\emptyset$ from X_1,
2. $C, Y_1, \ldots, Y_m$ is a derivation of $\emptyset$ from Y_1,
3. $1 \leq i \leq n$,

Then $\Sigma(C; i; X_1, \ldots, X_n; Y_1, \ldots, Y_m)$ denotes a derivation of $\emptyset$ from the following set of clauses:

1. The clauses of C,
2. For all j, with $1 \leq j < i$, the clause $X_j + Y_1$, where $X_j + Y_1$ is defined as $X_j \cup Y_1$ if the maximal element of X_j is the maximal element of $X_j \cup Y_1$, and as X_j, otherwise.
3. The clause $X_j \cup Y_1$.

The derivation $\Sigma(C; 1; X_1, \ldots, X_n; Y_1, \ldots, Y_m)$ is the derivation that we want.
Suppose now that we want to construct $\Sigma(C; i; X_1, \ldots, X_n; Y_1, \ldots, Y_m)$. If $i = n$, we can immediately take the derivation $C, Y_1, \ldots, Y_m$. Otherwise, if $i < n$, we distinguish two cases:

- If the maximum of X_i is the maximum of $X_i \cup Y_1$, then we can obtain $\Sigma(C; i{+}1; X_1, \ldots, X_n; Y_1, \ldots, Y_m)$ from $\Sigma(C; i{+}1; X_1, \ldots, X_n; Y_1, \ldots, Y_m)$ because $X_i \cup Y_1$ equals $X_i + Y_1$, and it is possible to derive a clause $X_{i+1} \cup Z$, where $Z \subseteq Y_1$, from $X_i \cup Y_1$. We can use subsumption to obtain the desired derivation.
- Otherwise we need two recursive calls. First we use

  $$\Sigma(C; i + 1; X_1, \ldots, X_n; Y_1, \ldots, Y_m),$$

  to obtain a refutation of $C, X_1 + Y_1, \ldots, X_{i-1} + Y_1, X_i$. This is possible because $X_i = X_i + Y_1$, and a clause $X_{i+1} \cup Z$, with $Z \subseteq Y_1$ can be derived. We write $D_1, \ldots, D_l$ for this derivation, with $D_1 = X_i$. This derivation cannot be used immediately because it will use $X_i + Y_1 = X_i$ instead of $X_i \cup Y_1$. However here Σ can be used to combine the derivations which use X_i and Y_1. So we put $\Sigma(C; i; X_1, \ldots, X_n; Y_1, \ldots, Y_m) = \Sigma(C \cup \{X_1 + Y_1, \ldots, X_{i-1} + Y_1\}; 1; Y_1, \ldots, Y_m, D_1; \ldots, D_l)$.

In order to see that this process terminates notice that it makes at most two recursive calls. In the first, the difference $n - i$ decreases, and Y_1 is unchanged. In the second $n - i$ may increase but Y_1 is replaced by another D_1 with lower (in $\prec$) maximal element. Because $\prec$ is a total order and only a finite number of literals occur in the original derivation $X_1, \ldots, X_n, Y_1, \ldots, Y_m$ and no new literals are invented $\prec$ is a well-order.
End of Proof
In this construction we have assumed that the order is total. However resolution with a partial order is also complete. This is because every partial order is included in a total order.

4.6 Comparison between the 2 methods

In this section we compare the bottom up and the top down method of obtaining a resolution calculus.
We first prove the well-known fact that there is a close relation between semantic tableaux and sequent calculus. Semantic tableau and sequent calculus are different manners of writing the same proof. There are some logics which have a sequent calculus and no semantic tableau calculus, for example linear logic and intuitionistic logic. All these logics are substructural. Because in our framework there is no room for substructurality all logics with a sequent calculus have a semantic tableau calculus.
After we have established this relation, we can apply both the transformation from sequent calculus and the transformation from semantic tableau on the same logic and compare the results. It turns out that we do not get the same. The Mints method starts with the axioms and tries to derive the clause that one wants to prove, whereas the method of Section 4.4 starts with the clauses that one wants to prove and ends in the axioms.
After that we show that it is possible to reverse a sequent calculus. That is: the axioms and the goal sequent can exchange their place. Finally we show that the transformation from semantic tableaux results in the same resolution calculus as the transformation from the reversed sequent calculus.
In order to compare the methods we will apply them to sequent calculus for classical logic, and semantic tableaux for classical logic. It is not immediately possible to compare the sequent calculus in Example 4.2.1 with the semantic tableau calculus in Example 4.4.1. This is because in the sequent calculus one tries to prove a formula, whereas in semantic tableau one tries to refute a formula. For this reason we modify the sequent calculus of Example 4.2.1 in such a manner that it will refute formulae instead of proving them. Because of the different meaning of a sequent we write the $\vdash$-symbol on the right instead of on the left: The sequent $\Gamma \vdash$ means: There

is no simultaneous model for all formulae in Γ.

Definition 4.6.1

- Every sequent of the form $A, \neg A \vdash$ is derivable.
- If the sequent $\Gamma, A \vdash$ is derivable then the sequent $\Gamma, A \wedge B \vdash$ is derivable.
- If the sequent $\Gamma, B \vdash$ is derivable then the sequent $\Gamma, A \wedge B \vdash$ is derivable.
- If the sequents $\Gamma_1, A \vdash$ and $\Gamma_2, B \vdash$ are derivable, then the sequent $\Gamma_1, \Gamma_2, A \vee B \vdash$ is derivable.
- If the sequent $\Gamma, \neg A \vee B \vdash$ is derivable, then the sequent $\Gamma, A \rightarrow B \vdash$ is derivable.
- If the sequent $\Gamma, A \vdash$ is derivable, then the sequent $\Gamma, \neg \neg A \vdash$ is derivable.
- If the sequent $\Gamma, \neg A \wedge \neg B \vdash$ is derivable, then the sequent $\Gamma, \neg (A \vee B) \vdash$ is derivable.
- If the sequent $\Gamma, \neg A \vee \neg B \vdash$ is derivable, then the sequent $\Gamma, \neg (A \wedge B) \vdash$ is derivable.
- If the sequent $\Gamma, A \wedge \neg B \vdash$ is derivable, then the sequent $\Gamma, \neg (A \rightarrow B) \vdash$ is derivable.
- If the sequent $\Gamma \vdash$ is derivable, then the sequent $\Gamma, A \vdash$ is derivable.

This sequent calculus can be written as $S = (P, \mathcal{R})$, with the following rules:

$\{A\} \vdash \{A \wedge B\}$ $\quad$ $\{B\} \vdash \{A \wedge B\}$
$\{A, B\} \vdash \{A \vee B\}$ $\quad$ $\{\neg A \vee B\} \vdash \{A \rightarrow B\}$,
$\{A\} \vdash \{\neg \neg A\}$, $\quad$ $\{\neg A \wedge \neg B\} \vdash \{\neg (A \vee B)\}$,
$\{\neg A \vee \neg B\} \vdash \{\neg (A \wedge B)\}$, $\quad$ $\{A \wedge \neg B\} \vdash \{\neg (A \rightarrow B)\}$.

We made some small modifications in the rules, to make the similarity with the semantic tableau calculus complete. If we had taken the negation of the calculus in Definition 4.2.1, then the rule for $\neg(A \rightarrow B)$ would have been: If one of the sequents $\Gamma, A \vdash$, or $\Gamma, \neg B \vdash$ is present then the sequent $\Gamma, \neg(A \rightarrow B) \vdash$ can be derived. The version given here is only a small modification.
We can now compare semantic tableaux and sequent calculus. If one wants to prove a sequent $\Gamma, A \vee B \vdash$ then the obvious thing to do is to look for a rule

which has $A \vee B$ among its consequents. There is only one such rule, which is: $\{A, B\} \vdash \{A \vee B\}$. So, if $\Gamma, A \vee B \vdash$ is not derived by weakening from $\Gamma \vdash$, then somewhere in the proof this rule has to be applied. So it is natural to try to prove $\Gamma_1, A \vdash$ and $\Gamma_2, B \vdash$, such that $\Gamma = \Gamma_1 \cup \Gamma_2$. Because the sequent calculus contains weakening it is safe to look for proofs of $\Gamma, A, A \wedge B \vdash$ and $\Gamma, B, A \wedge B \vdash$. This eliminates the problem of choosing Γ_1 and Γ_2 and the risk that we should have used $A \wedge B$ at another moment instead. This is essentially what is done when semantic tableaux are applied.

Definition 4.6.2 Let $S_{seq} = (P, \mathcal{R}_{seq})$ be a sequent calculus, with no rules of the form $\{A_1, \ldots, A_p\} \vdash$. Let $S_{tab} = (P, \mathcal{R}_{tab})$ be semantic tableau rules. If for every rule $\{A_1, \ldots, A_p\} \vdash \{B_1, \ldots, B_q\}$, there is a rule $\{B_1, \ldots, B_q\} \vdash \{A_1, \ldots, A_p\}$ in $\mathcal{R}_{tab}$, and there are no other rules in $\mathcal{R}_{tab}$, then we call S_{seq} and S_{tab} the *related* sequent calculus and semantic tableau calculus.

You see that like in real life both parties always want to go in opposite directions.

Theorem 4.6.3 Let $C_1, \ldots, C_r$ be a set of formulae. The sequent $\vdash C_1, \ldots, C_r$ is derivable with S_{seq} iff $C_1, \ldots, C_r$ is refutable with S_{tab}.

The proof is easy. Assume that $\vdash C_1, \ldots, C_r$ is provable in S_{seq}. We construct, by induction on the derivation of $\vdash C_1, \ldots, C_r$, a semantic tableau that refutes $C_1, \ldots, C_r$.
First note that it is always possible to have any C_i as root node.

We dinstinguish 2 cases:

1. $\vdash C_1, \ldots, C_r$ is obtained as an axiom. There is a rule $\vdash \{B_1, \ldots, B_q\}$ in S_{seq}, such that $\{B_1, \ldots, B_q\} \subseteq \{C_1, \ldots, C_r\}$. Then $(n_1, \ldots, n_q)$ with labels $(C_1, \ldots, C_n)$ is a closed semantic tableau, because the rule $\{B_1, \ldots, B_q\} \vdash \in \mathcal{R}_{tab}$, and all nodes are obtained as an application of rule 1 in Definition 4.4.5.

2. $\vdash C_1, \ldots, C_r$ is obtained in the following proof step:

$$\frac{\begin{array}{l}\vdash \Gamma_1, A_1, \\ \cdots \\ \vdash \Gamma_p, A_p,\end{array}}{\vdash \Gamma_1, \ldots, \Gamma_p, B_1, \ldots, B_q.}$$

By induction there are semantic tableaux T_i that refute $\vdash \Gamma_i, A_i$. Every T_i can be replaced by a semantic tableau T_i', in which A_i is the label of the root. Construct the following semantic tableau
$T = (N, <, \Lambda)$: T starts with a sequence $(n_1, \ldots, n_q)$, with labels $B_1, \ldots, B_q$. After this sequence the T_i' are attached as immediate descendents. This a correct semantic tableau because:

- Every n_i is obtained by application of rule 1 in Definition 4.4.5.
- Every node in a T_i' unequal to the root is obtained by the same rule as in T_i.
- Every root of a T_i' is obtained as an application of rule 2 in Definition 4.4.5, with $\{B_1, \ldots, B_q\} \vdash \{A_1, \ldots, A_p\} \in \mathcal{R}_{tab}$,

Suppose that $C_1, \ldots, C_r$ can be refuted with a semantic tableau T. Let J be an initial segment of a path of T. We show that for every such $J = \{n_1, \ldots, n_k\}$, the sequent $\vdash \Lambda(n_1), \ldots, \Lambda(n_k), C_1, \ldots, C_r$ is provable.

1. Let J be a complete path of T. Because J is closed, there is a rule $\{B_1, \ldots, B_q\} \vdash \emptyset$ such that all B_j occur as labels in J. Then $\vdash B_1, \ldots, B_q$ is an axiom of S_{seq} and this can be weakened to a sequent containing all the labels of J.

2. Let J be an initial segment of a path, which is not a complete path. Let $m_1, \ldots, m_p$ be the immediate successors of J, let Γ be the labels on J. They are obtained either in an application of rule 1 or rule 2. If it is rule 1, then $p = 1$, and the label of m_1 equals one of the C_j. By induction the sequent $\vdash \Gamma, C_1, \ldots, C_r, C_j$ is provable, which equals the sequent $\vdash \Gamma, C_1, \ldots, C_r$.

   If they are obtained in application of rule 2, then there is a rule $r = \{A_1, \ldots, A_p\} \vdash \{B_1, \ldots, B_q\} \in \mathcal{R}_{seq}$, such that J contains all the B_j as labels. All the sequents $\vdash \Gamma, C_1, \ldots, C_r, A_i$ are provable by induction. Then $\vdash \Gamma, C_1, \ldots, C_r, B_1, \ldots, B_q$ can be obtained by an application of r. Because Γ contains all the B_j as labels, this is equal to $\vdash \Gamma, C_1, \ldots, C_r$.

End of Proof

So semantic tableau is essentially a proof search strategy in sequent calculus, which tries to search a proof *top down*. That is: it starts with the formulae which one wants to prove and tries to reach the axioms.
The resolution method of Mints has precisely the opposite strategy. One starts with the axioms and tries to reach the formula that one wants to prove. For this reason it is a *bottom up* method.

When the methods of Sections 4.4 and 4.5 are applied to a semantic tableau the direction is preserved: The resolution proof will also be a top down proof. This is one important difference between the 2 methods. In many cases it is not known in advance which axioms will be used in the proof. In that case, with the method of Section 4.4, all axioms have to be taken as initial clauses. This is practically not feasible. Because of this the method has to be limited to sequent calculi where the set of potential axioms is finite, for example because of the subformula property.
However the methods of section 4.2 can be generalised to substructural logics, e.g. intuitionistic logic and linear logic. ([Mints88] and [Mints93]).
We finally prove an interesting thing: That it is possible to reverse a sequent calculus. In this manner bottom up and top down change position. The same is possible with a semantic tableau calculus. The resolution calculus obtained from the reversed sequent calculus, is the same as the resolution calculus obtained form the semantic tableau calculus. We prove it for semantic tableau, because it is easiest.

Theorem 4.6.4 Let $S = (P, \mathcal{R})$ be a semantic tableaux calculus, let C be a set of clauses of S. We define $\overline{S} = (\overline{P}, \overline{\mathcal{R}})$, and $\overline{C}$ from:

- $\overline{P} = \{\overline{p} \mid p \in P\}$,
- For every rule

  $$\{A_1, \ldots, A_p\} \vdash \{B_1, \ldots, B_q\} \in \mathcal{R}, \text{ with } q > 0,$$

  $\overline{\mathcal{R}}$ contains the rule

  $$\{\overline{B}_1, \ldots, \overline{B}_q\} \vdash \{\overline{A}_1, \ldots, \overline{A}_p\}.$$

  For every clause

  $$\{C_1, \ldots, C_r\} \in C,$$

  $\overline{\mathcal{R}}$ contains the rule

  $$\{\overline{C}_1, \ldots, \overline{C}_r\} \vdash \emptyset.$$

- $\overline{C} = \{\{\overline{A}_1, \ldots, \overline{A}_p\} \mid \{A_1, \ldots, A_p\} \vdash \emptyset \in \mathcal{R}\}$.

Then C has an S-model iff $\overline{C}$ has an $\overline{S}$-model.

Proof
It is sufficient to prove the theorem in one direction.
First we observe that it is possible to combine the clauses and rules into one set. Let

$$\mathcal{R}C = \mathcal{R} \cup \{\emptyset \vdash \{c\} \mid c \in C\}.$$

Then a set I is a model of $\mathcal{R}C$ if for all $\{A_1, \ldots, A_p\} \vdash \{B_1, \ldots, B_q\}$ in $\mathcal{R}C$, with $p \geq 0$, and $q \geq 0$, whenever $\{A_1, \ldots, A_p\} \subseteq I$,

$$\{B_1, \ldots, B_q\} \cap I \neq \emptyset.$$

Similarly it is possible to define a set $\overline{\mathcal{R}C}$, and $\overline{I}$ is a model of $\overline{\mathcal{R}C}$ if for all $\{\overline{A}_1, \ldots, \overline{A}_p\} \vdash \{\overline{B}_1, \ldots, \overline{B}_q\}$ in $\overline{RC}$, whenever $\{\overline{A}_1, \ldots, \overline{A}_p\} \subseteq \overline{I}$,

$$\{\overline{B}_1, \ldots, \overline{B}_q\} \cap \overline{I} \neq \emptyset.$$

Because of the construction of $\overline{\mathcal{R}C}$,

$$\{A_1, \ldots, A_p\} \vdash \{B_1, \ldots, B_q\} \in RC \text{ iff } \{\overline{B}_1, \ldots, \overline{B}_q\} \vdash \{\overline{A}_1, \ldots, \overline{A}_p\} \in \overline{RC}.$$

Now let I be a model of C, with S. We show that the set $\overline{I} = \{\overline{A} \mid A \notin I\}$ is a model of $\overline{C}$ with $\overline{S}$. Assume that $\{\overline{A}_1, \ldots, \overline{A}_p\} \subseteq \overline{I}$. Then $\{A_1, \ldots, A_p\} \cap I = \emptyset$. Because I is a model of $\mathcal{R}C$, it is not possible that $\{B_1, \ldots, B_q\} \subseteq I$. Hence it must be that $\{\overline{B}_1, \ldots, \overline{B}_q\} \cap I \neq \emptyset$.
End of Proof

4.7 Why do ordering refinements work?

In this section we try to explain why ordering refinements of resolution improve the process of finding a proof. This is not so easy as it seems at first sight. The problem is that, although an ordering refinement obviously reduces the number of clauses that will be derived in the i-th generation, it may also block the shortest derivation of the empty clause, and replace it by a longer one. It is possible that without the refinement the empty clause would be derived in the n-th generation, and that with the refinement the empty clause will be derived in the m-th generation with $m > n$. If the m-th generation with the refinement is larger than the n-th generation without the refinement, then nothing is gained.
There is no theoretical guarantee that ordering refinements improve the resolution process, but in practice they do.

Before we explain this on the predicate level, we explain it on the propositional level. On the propositional level a refinement can have two positive effects:

1. If a clause set C is satisfiable, then less clauses will be derived with a refinement. There is no problem in understanding this.

2. If a clause set C is unsatisfiable, then the empty clause will (in general) be derived more efficiently. This is more difficult to understand, because of the reasons above.

We think that (2) should be explained with (1): Let C be a minimal unsatisfiable clause set. All $C' \subset C$ are satisfiable. In a resolution refutation of C it will also be tried to refute all $C' \subset C$. Because all these failed attempts will be shorter the total search space will be smaller.

Example 4.7.1 Let C be the following clause set. C contains

- The clause $\{p_0\}$,
- For each i, with $0 \leq i < n$, the clause

$$\{\neg\, p_i, p_{i+1}\}$$

- The clause $\{\neg\, p_n\}$.

C is unsatisfiable. Let $\prec$ be the A-order defined from $\pm p_i \leq \pm p_j$ iff $i < j$. Without $\prec$, the empty clause can be derived from C in the $O(\log N) - th$ generation. (In the first generation, there will be all $\{\neg\, p_i, p_{i+2}\}$, the second generation will contain all $\{\neg\, p_i, p_{i+4}\}$, etc) With $\prec$, the empty clause will be derived from C in the $O(n)$-th generation.

Let $\overline{C}$ be the clause set obtained by deleting $\{p_0\}$ from C. $\overline{C}$ is satisfiable. Using $\prec$ only $O(n)$ clauses will be derived from both C and $\overline{C}$. Without $\prec$ $O(n^2)$ clauses will be derived from $\overline{C}$. These clauses will be derived in the $O(\log n)$-th generation.

So, in this example the behaviour of $\overline{C}$ completely compensates for the longer proof that is found with the refinement.

Chapter 5

Resolution in modal logic

In this chapter we will apply the results of Sections 4.4, and 4.5 to modal logic. Modal logics are a useful field for the application of these results because there are many modal logics. A unified treatment is useful. We will define a generic resolution method for a large class of modal logics. This class is the class of modal logics that are defined by a so called *conservative* restriction on the accessibility relation. We will call these modal logics the conservative modal logics. Consistently with Section 4.4 we will obtain resolution calculi from semantic tableaux calculi.

Many modal logics have interpretations which are based on the so called *possible world* semantics. A (possible world) interpretation consists of a set of possible worlds, connections between these possible worlds (the *accessibility relation*), and a function which defines the truth values of the propositional symbols in the different worlds. The different modal logics can be characterized by imposing restrictions on the accessibility relation. The standard modal logics can be obtained by natural restrictions on the accessibility relation.

It is possible to define a method to obtain in a general way a complete semantic tableau calculus from the restrictions on the accessibility relation of a modal logic. Examples are in ([Catach91]) and ([Fitting88]). Semantic tableaux for modal logics work essentially the same as for classical logic: One tries to construct an interpretation of the initial formula. If the semantic tableau closes, then this has failed. If the semantic tableau is open, then the open path corresponds to a model of the formula.

Unfortunately there is an important difference: In classical logic it is possible to define an interpretation by a set of literals. This is not possible in general in modal logic. Because of this, semantic tableaux in modal logic do not contain formulae, but graph structures, labelled with formulae. If we

would use this type of semantic tableaux for producing resolution calculi, the result would be a resolution calculus which does not use modal formulae, but a certain type of graph structures. This is not what we want. For this reason we define a new type of semantic tableau calculi, which use only formulae of modal logic. This is not possible for all modal logics. Because of this the method will be restricted to the conservative modal logics.
There exist many proposals for resolution calculi in modal logics. See, for example ([EnjFar89], [FarHerz88], [Fitting91], [Ohlbach88a], [Ohlbach88b]) None of these tries to obtain a general method. Also in ([Mints88]), it is not attempted to obtain a general method, although the author describes a general method for obtaining resolution calculi. This is because of the reason described in Section 4.6, namely that the method is bottom up, and the axioms are not known in advance.
We will define our resolution method for propositional logic only. In the Section 4.1 it is explained that the justification of resolution lies in the fact that it is non-analytical, and local on the predicate level. So if one defines a resolution calculus for propositional modal logic only, one misses the justification. The problem is that modal predicate logic is technically extremely complicated, because in general different worlds do not have the same domain. Also, in general, the same object has different names in different worlds. This makes ordinary unification and Skolemisation impossible. These problems deter us as they have deterred most authors, except for one: ([Somm92]).
In the next section we will define modal logic. After that we define the possible world semantics for modal logics, and give our method for representing restrictions on the accessibility relation. With this method we can define the conservative modal logics.
After that we will define the normal forms that we will use for resolution. Then there is a section filled with examples on how resolution works. Then finally we give the formal treatment.

5.1 Modal Logic

In this section we define modal logics. In general modal logics are defined by specifying axioms of an axiomatic deduction system. The modal logics have the same language, but they differ in the formulae that can be derived. We will first define the language of modal logics, and then define the axiomatic deduction systems for modal logics.
Modal logic is obtained from classical logic by adding two additional operators: $\Box A$, which has meaning: A is necessary, and $\Diamond A$, which has meaning: A is possible. It is assumed that $\Box A \equiv \neg \Diamond \neg A$, and $\Diamond A \equiv \neg \Box \neg A$.

Definition 5.1.1 We define the set of modal formulae as the set of things that can be made by applying finitely often the following rules:

- Every propositional symbol P is a modal formula.
- $\bot$ and $\top$ are modal formulae.
- If A is a modal formula, then $\neg A$ is a modal formula.
- If A and B are modal formulae, then $A \vee B$, $A \wedge B$ and $A \to B$ are modal formulae.
- If A is a modal formula, then $\Diamond A$ is a modal formula.
- If A is a modal formula, then $\Box A$ is a modal formula.

Next we define the axiomatic deduction systems for modal logics. The axiomatic form is the standard form of a deduction system for modal logic. It is defined by a set of axioms, and a set of deduction rules. The deduction rules are the same for all modal logics. There are some axioms which are common for all modal logics, and there are some axioms which depend on the logic.

Definition 5.1.2 We first define the rules and axioms which are common to all logics:

- Every propositional tautology A is derivable. Here modal subformulae are treated as propositional symbols.
- Every rule of the form

  $$\Box(A \to B) \to (\Box A \to \Box B)$$

  is derivable. This rule is called the *K-rule*.
- If A is derivable, then $\Box A$ is derivable. This rule is called the *necessitation* rule.
- If A and $A \to B$ are derivable, then B is derivable. This rule is called *modus ponens*.
- If A is derivable, then every $A[P := B]$ is derivable. Here $A[P := B]$ is the result of replacing all occurrences of a propositional symbol P by a modal formula B. This rule is called the *substitution* rule.

For each modal logic $\mathcal{L}$ there is the following rule:

- If A is an axiom belonging to the logic $\mathcal{L}$, then A is derivable.

Using the derived formulae, the $\vdash$-relation can be defined: $A_1, \ldots, A_p \vdash B$ iff $(A_1 \wedge \cdots \wedge A_p) \rightarrow B$ is derivable.

We give some of the more well known logics, and the axioms that define their deduction systems. For the philosophical motivations behind these logics we refer to ([HuCre84]).

Definition 5.1.3 The following axiom-schemata define the following modal logics:

K			
D	$\Diamond \top$		
T	$\Box A \rightarrow A$		
$S4$	$\Box A \rightarrow A$	$\Box A \rightarrow \Box \Box A$	
$S5$	$\Box A \rightarrow A$	$\Box A \rightarrow \Box \Box A$	$\Diamond \Box A \rightarrow A$
B	$\Box A \rightarrow A$		$\Diamond \Box A \rightarrow A$

In ([Catach91]) a very large overview of modal logics and corresponding axioms can be found. K is the weakest modal logic, because it has no axioms, except the K-rule. $S5$ is the strongest modal logic. We end with an example of a deduction in an axiomatic system:

Example 5.1.4 We will prove the following formula in the system B :

$$\Box(\neg\, P \wedge \Diamond \Box(\Box\, P \vee Q) \rightarrow Q).$$

(The $\wedge$ has priority over the $\rightarrow$) System B has the following axioms:

$\Box\, A \rightarrow A$,
$\Diamond \Box\, A \rightarrow A$.

A possible proof is:

(1)	$\Box\, P \rightarrow P$	(instance of an axiom)
(2)	$\Diamond \Box(\Box\, P \vee Q) \rightarrow (\Box\, P \vee Q)$	(instance of an axiom)
(3)	$[\Box\, P \rightarrow P] \rightarrow$ $([\Diamond \Box(\Box\, P \vee Q) \rightarrow (\Box\, P \vee Q)] \rightarrow [\neg\, P \wedge \Diamond \Box(\Box\, P \vee Q) \rightarrow Q])$	(prop. tautology)
(4)	$[\Diamond \Box(\Box\, P \vee Q) \rightarrow (\Box\, P \vee Q)] \rightarrow [\neg\, P \wedge \Diamond \Box(\Box\, P \vee Q) \rightarrow Q]$	(modus ponens on 1,3)
(5)	$\neg\, P \wedge \Diamond \Box(\Box\, P \vee Q) \rightarrow Q$	(modus ponens on 2,4)
(6)	$\Box(\neg\, P \wedge \Diamond \Box(\Box\, P \vee Q) \rightarrow Q)$	(necessitation rule on 5)

5.2 Semantics of Modal logic

In this section we will formally define the possible world semantics for modal logics. After that we define our method of defining restrictions on the accessibility relation. This method is based on what we call *tree rules*. With these tree rules, we can define the class of conservative modal logics.
The possible world semantics was introduced in ([Kripke59]). In the possible world semantics, the modal statement 'A is possible' is interpreted as: There is a world which is an alternative to the present world, where A holds. The notation that we use is from ([Goldbl87]).

Definition 5.2.1 Let F be a modal formula. An *interpretation* I of F is a triple $I = (S, R, v)$ with

- S is a non-empty set of possible worlds, or states, or moments.
- R is the accessibility relation. It is a binary relation on S.
- v is a function which interprets every propositional symbol in F in every world. It is a function from $P \times S$ to $\{true, false\}$. Here P is the set of propositional symbols in P.

We will say that $s' \in S$ *can be reached* from $s \in S$ if $R(s, s')$. We define when an interpretation *makes a modal formula true* in a world: Let F be a modal formula, let $I = (S, R, v)$ be an interpretation of F. Let s be a world of S.

- If F is a propositional symbol, then I makes F true in s if $v(F, s) = true$. Otherwise I makes F false in s.
- If $F = \bot$, then I makes F false in s.
- If $F = \top$, then I makes F true in s.
- If F has form $A \wedge B$, then I makes F true in s if I makes both A and B true in s. Otherwise I makes F false in s.
- If F has form $A \vee B$, then I makes F true in s if I makes at least one of A and B true in s. Otherwise I makes F false in s.
- If F has form $A \rightarrow B$, then I makes F true in s if I makes A false in s, or I makes B true in s. If this is not the case then I makes F false in s.
- If F has form $\Box A$, then I makes F true in s if for all worlds s' that can be reached from s, I makes A true in s'. Otherwise I makes F false in s.

- If F has form $\Diamond A$, then I makes F true in s if there exists a world s' that can be reached from s, such that I makes A true in s'.

An interpretation I makes a formula A true (without mentioning any world) if I makes A true in every world.

Definition 5.2.2 Let $\mathcal{L}$ be a modal logic, which is defined by a set of axioms. Let $\mathcal{I}$ be a set of interpretations that is characterized by a certain condition on the accessibility relation. We say that $\mathcal{L}$ is *characterized* by $\mathcal{I}$ if the set of formulae that can be derived in $\mathcal{L}$ equals the set of formulae that are true in all interpretations of $\mathcal{I}$.

A sequent $\Gamma \vdash B$ *holds* in $\mathcal{I}$ if, for every interpretation $I \in \mathcal{I}$, such that there is a world s of I, such that I makes the formulae in Γ true in s, I makes the formula B true in I.

A formula F is *satisfiable* in $\mathcal{I}$ if there is an interpretation $I \in \mathcal{I}$, such that I makes F true in a world s.

Similar to classical logic, a sequent $\Gamma \vdash B$ holds iff $\Gamma, \neg B$ is unsatisfiable in a class of interpretations.
As said before the various modal logics, as defined in Definition 5.1.3, are characterized by certain subclasses of the possible interpretations. The subclasses are obtained by imposing natural conditions on the accessibility relation. Some of the possible conditions are:

Seriality $\forall X \exists Y R(X, Y)$.

Reflexivity $\forall X R(X, X)$.

Transitivity $\forall X \forall Y \forall Z (R(X, Y) \wedge R(Y, Z) \rightarrow R(X, Z))$.

Symmetry $\forall X \forall Y (R(X, Y) \rightarrow R(Y, X))$.

The modal logics that are defined in Definition 5.1.3. are sound and complete for the classes of modal interpretations that satisfy the following conditions on the accessibility relation:

K			
D	seriality		
T	reflexivity		
$S4$	reflexivity	transitivity	
$S5$	reflexivity	transitivity	symmetry
B	reflexivity		symmetry

There are proofs in ([Goldbl87]) and ([HuCre84]). It is possible to define the axioms for modal deduction systems from the conditions on the accessibility relation with general methods in the same manner as we will do this for resolution calculi (See [Nivelle92]).
In order to be able to define a general method of obtaining resolution calculi, it is necessary to have a representation for possible conditions on the accessibility relation. The representation that we will give here is based on tree rules. The class of conditions for which our method works is the class of conditions which can be defined by these tree rules. A tree rule is a pair consisting of a tree and a branch, with the meaning: If the structure (S, R) of an interpretation (S, R, v) contains the tree, then it must also contain the branch.

Definition 5.2.3 A *tree rule* is a quadruple $(N, <, n_f, n_t)$ where $(N, <)$ is a finite tree, and $n_f, n_t \in N$. The K-rule is the rule $(w_0(w_1), w_0, w_1)$. In the sequel we write just rule instead of tree rule.

We define its formalised meaning:

Definition 5.2.4 Let $T = (N, <)$ be a tree. Let $I = (S, R, v)$ be an interpretation of a modal formula F.

An *embedding* of T into I is a total function, (not necessarily surjective, not necessarily injective), such that:

- If n_2 is a direct descendant of n_1 in T, then $(\varphi(n_1), \varphi(n_2)) \in R$.

An interpretation $I = (S, R, v)$ satisfies a rule (T, n_f, n_t) if for every embedding φ of T into I,

$$(\varphi(n_f), \varphi(n_t)) \in R.$$

An interpretation I satisfies a set of rules $\mathcal{R}$ if it satisfies every rule in $\mathcal{R}$. A set of modal interpretations is *conservative* if it can be defined as the set of modal interpretations that satisfies a certain set of tree rules $\mathcal{R}$. A modal logic is conservative if it is characterized by a conservative set of interpretations.

The K-rule is tautologeous, because it states that every branch that is there, is really there. We give some examples of tree rules:

reflexivity	(w, w, w)
transitivity	$(w_1(w_2(w_3)), w_1, w_3)$
symmetry	$(w_1(w_2), w_2, w_1)$

The logic B is characterized by the set of rules $\mathcal{R} = \{(w, w, w), (w_0(w_1), w_1, w_0)\}$. The property of seriality cannot be expressed by a tree rule, because it is not possible to require the existence of worlds, only of connections in the accessibility relation.

The frame properties that can be defined by tree rules are called conservative because tree rules are not able to postulate the existence of worlds. Tree rules can only postulate the existence of branches in the accessibility relation. Progressive people want new worlds, and conservative people want to stick to the old one.

5.3 Normal Forms

In the same manner as for resolution in classical logic, resolution in modal logic uses normal forms. The following normal form has as only function to remove redundancy in notation. The $\rightarrow$ is removed and negations are moved inwards.

Definition 5.3.1 We define the *negation normal form* (which will be abbreviated as NNF). A formula is in NNF if it can be constructed finitely by the following rules:

- Every propositional symbol P is in NNF.
- The negation of a propositional symbol $\neg P$ is in NNF.
- $\bot$ and $\top$ are in NNF.
- If A is in NNF, then $\Box A$ and $\Diamond A$ are in NNF.
- If A and B are in NNF, then $A \vee B$ and $A \wedge B$ are in NNF.

Theorem 5.3.2 The following replacements transform an arbitrary formula into an equivalent formula in NNF.

$$
\begin{array}{lcl}
A \to B & \Rightarrow & \neg\, A \vee B, \\
\neg\,(A \vee B) & \Rightarrow & \neg\, A \wedge \neg\, B, \\
\neg\,(A \wedge B) & \Rightarrow & \neg\, A \vee \neg\, B, \\
\neg\,(A \to B) & \Rightarrow & A \wedge \neg\, B, \\
\neg\,\neg\, A & \Rightarrow & A, \\
\Box\top & \Rightarrow & \top, \\
\Diamond\bot & \Rightarrow & \bot.
\end{array}
\qquad
\begin{array}{lcl}
A \vee \bot & \Rightarrow & A, \\
A \vee \top & \Rightarrow & \top, \\
A \wedge \bot & \Rightarrow & \bot, \\
A \wedge \top & \Rightarrow & A, \\
\bot \vee A & \Rightarrow & A, \\
\top \vee A & \Rightarrow & \top, \\
\bot \wedge A & \Rightarrow & \bot, \\
\top \wedge A & \Rightarrow & A.
\end{array}
$$

Resolution in modal logic is used in the same way as in classical logic. If one wants to prove a sequent $A_1 \wedge \cdots \wedge A_p \vdash B$, this is done by trying to refute the formula $A_1 \wedge \cdots \wedge A_p \wedge \neg\, B$.
The normal form that we will use is based on the following:

- We do not want a $\vee$ in the scope of a $\Diamond$, because they are very difficult to handle. This means that in the scope of a $\Diamond$ we will try to factor the $\vee$'s out.
- For the rest we want the normal form to be as close as possible to the clausal normal form in classical logic. This means that we will try obtain a formula of the form $(L_{1,1} \vee \cdots \vee L_{1,l_1}) \wedge \cdots \wedge (L_{n,1} \vee \cdots \vee L_{n,l_n})$, where the $L_{i,j}$ are a sort of literals.

This will be made formal in the next definition:

Definition 5.3.3 A *modal literal* is a modal formula in NNF in which every $\vee$ is in the scope of a $\Box$. A *modal clause* is a finite set of modal literals.

Theorem 5.3.4 There is an algorithm which transforms every modal formula F in NNF into a finite set of modal clauses. We write $< F >$ for this set. The algorithm is the following:

1. Every $\vee$ in the scope of a $\Box$ is not affected.
2. On every $\vee$ in the scope of a $\Diamond$, but not in the scope of a $\Box$, the following rules will be applied:

$$
\begin{array}{lcl}
(A \vee B) \wedge C & \Rightarrow & (A \wedge C) \vee (B \wedge C), \\
A \wedge (B \vee C) & \Rightarrow & (A \wedge B) \vee (A \wedge C), \\
\Diamond(A \vee B) & \Rightarrow & \Diamond A \vee \Diamond B.
\end{array}
$$

3. On $\vee$'s that are not in the scope of a $\Diamond$ and not in the scope of a $\Box$, the following rules will be applied:

$$\begin{array}{lcl}(A \wedge B) \vee C & \Rightarrow & (A \vee C) \wedge (B \vee C), \\ A \vee (B \wedge C) & \Rightarrow & (A \vee B) \wedge (A \vee C).\end{array}$$

When this rewriting is finished, the outcome has form:

$$(L_{1,1} \vee \cdots \vee L_{1,l_1}) \wedge \cdots \wedge (L_{n,1} \vee \cdots \vee L_{n,l_n}),$$

in which the $L_{i,j}$ are modal literals. This will be replaced by

$$\{\{L_{1,1}, \ldots, L_{1,l_1}\}, \ldots, \{L_{n,1}, \ldots, L_{n,l_n}\}\}.$$

Let us give some examples:

Example 5.3.5

- For every modal formula C in NNF, except for $\top$, the formula $\Diamond(P \wedge \Box C)$ is a modal literal.
- $\Diamond(P \vee Q)$ is not a modal literal, because it can be factored into $\Diamond P \vee \Diamond Q$.
- $\Diamond(P \wedge \Diamond(Q \wedge R))$ is a modal literal.
- $\Diamond(\Box(\Diamond(P \vee Q)) \wedge R)$ is a modal literal.

The formula $F = \Diamond(\Box(P \vee Q) \vee (R \wedge S))$ will be transformed into the following set of clauses:

$$\{\{\Diamond(\Box(P \vee Q)), \Diamond(R \wedge S)\}\}.$$

The formula $F = \Diamond(P \vee Q) \vee (\Box(R \wedge S) \wedge \Diamond(T \vee \Diamond U))$ will be transformed into

$$\begin{array}{l}\{ \\ \quad \{\Diamond P, \Diamond Q, \Box(R \wedge S)\}, \\ \quad \{\Diamond P, \Diamond Q, \Diamond T, \Diamond\Diamond U\} \\ \}.\end{array}$$

The formula $F = \Diamond(P \vee Q) \vee \Diamond(\Box(R \wedge S) \wedge \Diamond(T \vee \Diamond U))$ will be transformed into

$$\{\{\Diamond P, \Diamond Q, \Diamond(\Box(R \wedge S) \wedge \Diamond T), \Diamond(\Box(R \wedge S) \wedge \Diamond\Diamond U))\}\}.$$

5.4 Examples to Resolution

In the next section we will formally define the semantic tableau rules, from which the resolution calculi will be obtained. In this section we will explain the rules, using many examples. We do this by starting with no rules at all, and then repeat the following seven times:

- Show with an example that something goes wrong.
- Add rules in order to prevent this.

For a modal logic $\mathcal{L}$, we define a semantic tableau calculus $S_T = (P_T, \mathcal{R}_T)$, where P_T is the set of modal literals. The rules $\mathcal{R}_T$ have to be defined in such a manner that every set $M \subseteq P_T$, which is closed under the rules $\mathcal{R}T$, belonging to modal logic $\mathcal{L}$ is satisfiable in $\mathcal{L}$. The rules must also be sound. Start with $S_T = (P_T, \mathcal{R}_T)$, where $\mathcal{R}_T = \emptyset$.

1. The set $M = \{P, Q\}$ is an interpretation of $(P_T, \emptyset)$. It has a modal interpretation $I = (S, R, v)$ with $S = \{\overline{s}\}$, $R = \emptyset$, and $v(P, \overline{s}) = v(Q, \overline{s}) = true$. This will be the general strategy: To construct an $I = (S, R, v)$ with a special world $\overline{s}$, where all modal literals in M will be true. It works for $M = \{P, Q, R\}$, and for $M = \{\neg\, P, Q, \neg\, R\}$.

2. It does not work for $M = \{P, Q, R, \neg\, P\}$. The reason is that it is not possible to have $v(P, \overline{s}) = true$, and $v(\neg\, P, \overline{s}) = true$. As a consequence, for every propositional symbol P, a rule $\{P, \neg\, P\} \vdash \emptyset$ is added to $\mathcal{R}_T$. It is easily seen that this is a sound rule.

3. We are really making progress now. We already have a complete resolution calculus for modal logic without the $\Box$ and the $\Diamond$. But now consider $M = \{\Diamond P, \Diamond \neg\, P, Q, R\}$, and the system K, with no conditions on R. The obvious solution here is to take $I = (S, R, v)$, with $S = \{\overline{s}, s_1, s_2\}$, $R(\overline{s}, s_1)$, and $R(\overline{s}, s_2)$, and

   $v(Q, \overline{s}) = true$,
   $v(R, \overline{s}) = true$,
   $v(P, s_1) = true$,
   $v(P, s_2) = false$.

   This strategy also works for

   $$M = \{\Diamond(P \wedge \Diamond Q \wedge R \wedge \Diamond(\Diamond S \wedge \Diamond P))\}, \text{ and}$$

   $$M = \{\Diamond(\Diamond P \wedge \Diamond \neg\, P), \Diamond(\Diamond Q \wedge \Diamond \neg\, Q)\}.$$

   This last formula will have $I = (S, R, v)$, with $S = \{\overline{s}, s_1, s_2, s_3, s_4, s_5, s_6\}$ and $R(\overline{s}, s_1), R(s_1, s_2), R(s_1, s_3), R(\overline{s}, s_4), R(s_4, s_5), R(s_4, S_6)$.

In general we will try to obtain a modal interpretation by copying the tree structure which is formed by the $\Diamond$'s in the modal literals in M.

4. This will not work, unfortunately for $M = \{\Diamond(P \wedge Q \wedge \neg P)\}$, because it is not possible to have $v(P, s_1) = true$, and $v(P, s_1) = false$. It is necessary to add rules of the form: $\{F\} \vdash \emptyset$, when construction of an interpretation of F would force us to make both P and $\neg P$ true in one world. This happens when P and $\neg P$ are in the scope of the same $\Diamond$, and this $\Diamond$ is not in the scope of a $\Box$.

5. Now we must handle the $\Box$. Consider $M = \{\Box P, \Diamond(Q \wedge \neg P)\}$, and the system K. The constructed interpretation will be $I = (S, R, v)$ with $S = \{\overline{s}, s_1\}$, with $R(\overline{s}, s_1)$, and $v(P, s_1) = false,\ v(Q, s_1) = true$. Unfortunately, now $\Box P$ is false in $\overline{s}$, because of $R(\overline{s}, s_1)$, and $v(P, s_1) = false$. There is no modal interpretation, in which M is true. What is necessary here is a rule, which ensures that when there are two modal literals $\Box A$, and $\Diamond B$, that then A will be true in the world constructed from $\Diamond B$. The obvious rule is:

$$\{\Box A, \Diamond B\} \vdash \{\Diamond(A \wedge B)\}.$$

It will be also necessary to apply this rule in the scope of a $\Diamond$. Like:

$$\{\Diamond \cdots \Diamond(\Box A \wedge \cdots \wedge \Diamond B) \cdots)\} \vdash \{\Diamond \cdots \Diamond(\Box A \wedge \cdots \wedge \Diamond(A \wedge B) \cdots)\}.$$

These rules introduce a new problem: The construction of the modal interpretation, as was given in Step 3 does not work anymore. Consider

$$M = \{\Box P, \Diamond Q, \Diamond(Q \wedge P), \Diamond(Q \wedge P \wedge P), \Diamond(Q \wedge P \wedge P \wedge P), \ldots\}.$$

In the constructed interpretation there will still be a world constructed from $\Diamond Q$ where possibly P is false. The world constructed from $\Diamond(Q \wedge P)$ is intended to be the same world as the world constructed from $\Diamond Q$, but the construction method of Step 3 ignores this fact. The situation can be more complex. For example in

$$M = \{\Box P, \Box Q, \Box R, \Diamond A, \Diamond(A \wedge P), \Diamond(A \wedge P \wedge Q), \Diamond(A \wedge P \wedge Q \wedge R)\},$$

all the worlds that will be constructed are intended to be the same world. In order to solve this problem we have to introduce a relation $\sqsubset$ on positions in modal literals in M. Its meaning is: $p_1 \sqsubset p_2$ if the formula in which p_2 occurs follows from the formula in which p_1 occurs, by a rule, and then p_2 is obtained from p_1. Then the set of

possible worlds S will be constructed by using $\sqsubset$ sequences instead of the positions themselves. In the last M there is the sequence:

[the position in $\Diamond A$ containing A] $\sqsubset$
[the position in $\Diamond(A \wedge P)$ containing $A \wedge P$] $\sqsubset$
[the position in $\Diamond(A \wedge P \wedge Q)$ containing $A \wedge P \wedge Q$)] $\sqsubset$
[the position in $\Diamond(A \wedge P \wedge Q \wedge R)$ containing $A \wedge P \wedge Q \wedge R$].

Then the accessibility relation R will be defined from: If somewhere in M the formula $\Diamond \cdots \Diamond A$ occurs, then

$R($ [Every sequence containing the position of A],

[every sequence containing the position of $\Diamond A$])

finΦΦΦFinally v is defined from the literals that occur in the positions occurring in the sequences.

6. Until now we have only considered the system K. Suppose that the set of tree rules $\mathcal{R}$ is not empty. Then in general the construction of Step 5 will not work, because the constructed interpretation will not satisfy $\mathcal{R}$. Another stage has to be added to the construction: When the interpretation $I = (S, R, v)$ does not satisfy one of the rules, the missing connections are added to R. The result will be called the *closure* of R. We write $\overline{R}$ for the result, and define $\overline{I} = (S, \overline{R}, v)$. If, for example, $\mathcal{R} = \{(n, n, n)\}$ (reflexivity) and

$$M = \{\Box A, \neg A\},$$

then the construction of Step 6 would result in $I = (S, R, v)$ with $S = \{\overline{s}\}, R = \emptyset$, and $v(\overline{s}, A) = false$. This interpretation does not satisfy $\mathcal{R}$. The connection $(\overline{s}, \overline{s})$ has to be added. Then $\overline{I} = (S, \overline{R}, v)$ with $\overline{R} = (\overline{s}, \overline{s})$ satisfies $\mathcal{R}$.

Now, unfortunately $\Box A$ is not true anymore in $\overline{s}$ with this new $\overline{I}$. In order to avoid this, extra semantic tableau rules in $\mathcal{R}_T$, dependent on $\mathcal{R}$ are necessary. It is necessary to anticipate the fact that the cycles will be added. This is possible with the rules:

$$\{\Box A\} \vdash \{A\}, \text{ and}$$

$$\{\Diamond \cdots \Diamond(\Box A \wedge B)\} \vdash \{\Diamond \cdots \Diamond(\Box A \wedge B \wedge A)\}.$$

To $(n_1(n_2), n_2, n_1)$ correspond the rules:

$$\{\Diamond(\Box A \wedge B)\} \vdash \{A\},$$

$$\{\Diamond \cdots (\Diamond(\Box\, A \wedge B) \wedge C)\} \vdash \{\Diamond \cdots (\Diamond(\Box\, A \wedge B) \wedge C \wedge A)\}.$$

The general method to obtain these rules is the following: It is possible to predict from a set of literals $\{A_1, \ldots, A_p\}$ the tree structure that it will generate. If for a rule (T, n_f, n_t) the tree T occurs in this tree structure, and a formula $\Box\, A$ occurs at a place corresponding to n_f, then A has to be added in the place corresponding to n_t. The rules, mentioned above can be obtained in this manner.

7. Now there is one remaining problem. For Step 6 it must be possible to determine from the tree structure that will be generated by the formulae, the branches that will be added. Sometimes it is not possible to do this in one step. For example, let $\mathcal{R}$, contain $(n_0(n_1(n_2)), n_0, n_2)$. This rule expresses transitivity.

$$M = \{\Box\, A \wedge \Diamond \Diamond \Diamond \Diamond \neg\, A\}$$

has no modal interpretation satisfying $\mathcal{R}$. There is the following modal rule, obtained from $\mathcal{R}$.

$$\{\Box\, A \wedge \Diamond(B \wedge \Diamond C)\} \vdash \{\Box\, A \wedge \Diamond(B \wedge \Diamond(C \wedge A))\}.$$

Now M is not an interpretation of this rule, but

$$M' = \{\Box\, A \wedge \Diamond \Diamond(A \wedge \Diamond \Diamond \neg\, A)\}.$$

is. From M' it is not possible to construct an interpretation, satisfying $\mathcal{R}$. It goes wrong because closure is not reached in one iteration. In order to solve this, the set of rules $\mathcal{R}$ has to be replaced by another set of rules $\overline{\mathcal{R}}$, with the property that the closure is always reached in one step. This set $\overline{\mathcal{R}}$ will be called the *closure* of $\mathcal{R}$.

5.5 Technical Preparation

We will now prepare for the formal treatment. Before we can give the formal description, we will make explicit how the tree structure can be extracted from a modal literal, as was done in Step 3. This structure will be used in defining the semantic tableau rules, and in constructing the interpretation. After that we will formally define the closure of Step 6, and of Step 7.

Definition 5.5.1 Let F be a modal formula in NNF. The *tree representation* of F is a labelled tree (T, Λ). The labels are sets of modal formulae. It is the result of the following process: Start with a tree T consisting of one node n with label $\{F\}$. Then extend T as long as possible in accordance with the following rules:

1. If there is a node n in T, that is labelled with a set that contains a formula $A \wedge B$, then $A \wedge B$ is replaced by A and B in this label set.
2. If there is a node n in T, that is labelled with a set that contains $\top$, then $\top$ is removed from this label set.
3. If there is a node n in T, that is labelled with a set that contains a formula $\Diamond A$, then a new node n' is added to T as a direct descendant of n and this node receives label $\{A\}$.

Every node n in T can be traced back to a position in F from which it originates.
The tree representation can be easily extracted from a modal formula in NNF. It will not be actually constructed during the resolution process, but be used in order to precisely define the modal resolution rules. In a computer program it is quite easy to avoid it, but it makes the explanation clearer. The tree representation tree has to be present in every modal interpretation which makes F true.

Lemma 5.5.2 Let F be a modal formula in NNF. Let $(N, <, \Lambda)$ be the tree representation of F. Define $T = (N, <)$. Let t be the root of T. Let $I = (S, R, v)$ be an interpretation of F. Let s be a world of S. The following 2 are equivalent:

1. I makes F true in s.
2. There exists an embedding φ of T in (S, R), such that $\varphi(t) = s$, and for all $n \in N$, I makes all formulae in $\Lambda(n)$ true in $\varphi(n)$.

Proof
The proof is by induction on the construction of the tree representation in Definition 5.5.1. We call the iterations of the construction

$$(N_0, <_0, \Lambda_0), \ldots, (N_m, <_m, \Lambda_m).$$

Then $(N_m, <_m, \Lambda_m) = (N, <, \Lambda)$. It is sufficient to prove the following:

- For $(N_0, <_0, \Lambda)$ it is the case that (1) $\Leftrightarrow$ (2).
- If (2) holds for $(N_i, <_i, \Lambda_i)$, then (2) holds for $(N_{i+1}, <_{i+1}, \Lambda_{i+1})$.
- If (1) holds for $(N_{i+1}, <_{i+1}, \Lambda_{i+1})$, then (1) holds for $(N_i, <_i, \Lambda_i)$.

Then, for $(N_i, <_i, \Lambda_i)$ the equivalence (1) $\Leftrightarrow$ (2) can be obtained by i times applying the second statement, one time the first statement, and i times applying the third statement.
We begin by proving that (1) $\Leftrightarrow$ (2) for $(N_0, <_0, \Lambda_0)$. The initial tree representation equals $(N_0, <_0, \Lambda_0)$ where

- N_0 consists of one node t. This node is the root of $(N_0, <_0)$.
- $<_0$ is empty.
- $\Lambda_0(n)$ equals $\{F\}$.

Suppose that I makes F true in s. Define φ_0 from $\varphi_0(t) = s$. φ_0 is clearly an embedding of $T_0 = (N_0, <_0)$ into I. We have $\Lambda(t) = \{F\}$, and F is true in $\varphi_0(t)$.
Suppose, for the other direction, that there exists an embedding φ_0 of $(N_0, <_0)$ into (S, R), with $\varphi_0(t) = s$, such that I makes all elements of $\Lambda(n)$ true in $\varphi_0(t)$. Then obviously I makes F true in s, because $\Lambda(n) = \{F\}$.
We now prove that if (2) holds for $(N_i, <_i, \Lambda_i)$ with embedding φ_i, then there exists an embedding φ_{i+1}, such that (2) holds for $(N_{i+1}, <_{i+1}, \Lambda_{i+1})$ with φ_{i+1}. We distinguish the cases how $(N_{i+1}, <_{i+1}, \Lambda_{i+1})$ is obtained from $(N_i, <_i, \Lambda_i)$. It will always be that $\varphi_i \subseteq \varphi_{i+1}$, and so always $\varphi(s) = t$.

- Assume that $(N_{i+1}, <_{i+1}, \Lambda_{i+1})$ is obtained by an application of case 1. Then $(N_{i+1}, <_{i+1}) = (N_i, <_i)$. We can take $\varphi_{i+1} = \varphi_i$. Let $n \in N_i$ be the node for which $A \wedge B$ is replaced by A and B. The other nodes are not problematic because their labels do not change. Because the other elements of $\Lambda_{i+1}(n)$ were already present in $\Lambda_i(n)$, it is sufficient to show that A and B are true in $\varphi_{i+1}(n)$. This follows from the fact that $A \wedge B$ is true in $\varphi_i(n)$.
- Assume that $(N_{i+1}, <_{i+1}, \Lambda_{i+1})$ is obtained by an application of case 2. Then $(N_{i+1}, <_{i+1}) = (N_i, <_i)$. It is possible to take $\varphi_{n+1} = \varphi_n$. Because for every $n \in N_i$, $\Lambda_{i+1}(n) \subseteq \Lambda_i(n)$, it follows, for every $n \in N$, that all elements in $\Lambda_{i+1}(n)$ are true in $\varphi_{n+1}(n)$.
- Assume that $(N_{i+1}, <_{i+1}, \Lambda_{i+1})$ is obtained by an application of case 3. Then N_{i+1} is obtained from N_i by adding one node n_2 (as a successor of n_1). $<_{i+1}$ is obtained by adding (n_1, n_2) to $<_i$. Using the induction hypothesis, we have

  $$\Diamond A \text{ is true in } \varphi_(n_1).$$

  Write $s_1 = \varphi_i(n_1)$. There must be world s_2, such that $R(s_1, s_2)$, and I makes A true in s_2. Then if we put $\varphi_{i+1} = \varphi_i \cup \{(n_2, s_2)\}$, φ_{i+1} will make all elements of $\Lambda_{i+1}(n)$ true in $\varphi_{i+1}(n)$, for all $n \in N_{i+1}$.

It remains to prove that if (2) holds for $(N_{i+1}, <_{i+1}, \Lambda_{i+1})$, then (2) holds for $(N_i, <_i, \Lambda_i)$. Call the embeddings $\varphi_0, \ldots, \varphi_i$. We distinguish how $(N_{i+1}, <_{i+1}, \Lambda_{i+1})$ is obtained from $(N_i, <_i, \Lambda_i)$.

- In case 1, $(N_i, <_i) = (N_{i+1}, <_{i+1})$. For all nodes $n \in N_i$, we have $\Lambda_{i+1}(n) = \Lambda_i(n)$, with one exception: In the label set of one node n, the formula $A \wedge B$ has been replaced by A and B. Because I makes $A \wedge B$ true in $\varphi_{i+1}(n)$, I makes both A and B true in $\varphi_i(n)$.

- In case 2, $(N_i, <_i) = (N_{i+1}, <_{i+1})$, and for all nodes $n \in N_i$, $\Lambda_i(n) = \Lambda_{i+1}(n)$, with one exception: In the label set of one node n, $\top$ has been added to $\Lambda_{i+1}(n)$. Certainly I will make $\top$ true in $\varphi_i(n)$.

- In case 3, call the node where $\Diamond A$ occurs n_1, and the new created node n_2. Then $N_i = N_{i+1} \backslash \{n_2\}$, and $<_i = <_{i+1} \backslash \{(n_1, n_2)\}$. For all nodes $n \in N_i$, $\Lambda_i(n) = \Lambda_{i+1}(n)$, with one exception: For node n_1 the formula $\Diamond A$ has been added to the label set $\Lambda_i(n_1)$. If we take φ_i by removing (n_2, s_2) from φ_{i+1}, then it is sufficient to prove that

  $$I \text{ makes } \Diamond\ A \text{ true in } \varphi_i(n_1).$$

  Because φ_{i+1} is an embedding, and n_2 is a successor of n_1 it must be that $R(\varphi_{i+1}(n_1), \varphi_{i+1}(n_2))$. Because I makes A true in $\varphi_{i+1}(n_2)$, I makes $\Diamond A$ true in $\varphi_i(n_1)$.

End of Proof

In the rules described in Step 5 and Step 6 in Section 5.4, it is necessary to add a formula at a certain position in a formula. For this we define:

Definition 5.5.3 Let (T, Λ) be the tree representation of a modal literal F. Let n be a node of T. We define the result of *adding* A at n, as the result of replacing the formula B at the position from which n originates by $A \wedge B$.

It is possible that the result is not a modal literal.

Example 5.5.4 Let

$$F = \Box A \wedge \Diamond(\Diamond \Diamond \top \wedge \Box \Diamond A \wedge \Diamond(A \wedge B)).$$

The tree representation has $T = n_0(n_1(n_2(n_3)), n_4)$ and Λ defined from:

$$\begin{array}{lcl} \Lambda(n_0) & = & \{\Box\, A\}, \\ \Lambda(n_1) & = & \{\Box \Diamond A\}, \\ \Lambda(n_2) & = & \{\}, \\ \Lambda(n_3) & = & \{\}, \\ \Lambda(n_4) & = & \{A, B\}. \end{array}$$

The result of adding P at n_1 equals

$$\Box\, A \wedge \Diamond(\Diamond(\Diamond \top) \wedge P \wedge \Box \Diamond A \wedge \Diamond(A \wedge B)\ ).$$

The result of adding $\Diamond P$ at n_0 equals

$$\Box\, A \wedge \Diamond(\Diamond \Diamond \top \wedge \Box \Diamond A \wedge \Diamond(A \wedge B)) \wedge \Diamond P.$$

We define the closure of Step 7 in Section 5.4.

Definition 5.5.5 Let $\mathcal{R}$ be a set of rules. Let $I = (S, R, v)$ be an interpretation of a formula F. The $\mathcal{R}$-*closure* of I, (denoted as $\overline{I}$), is obtained by replacing R by the smallest $\overline{R}$, such that $R \subseteq \overline{R}$, and $\overline{I} = (S, \overline{R}, v)$ satisfies $\mathcal{R}$.

This is a well-defined notion because it can be obtained by the standard iteration:

- $$R_0 = R,$$
- R_{i+1} is obtained from R_i as follows: If R_i violates a rule $(N, <, n_f, n_t)$ in $\mathcal{R}$, then there exists an embedding φ of $(N, <)$ in (S, R, v), for which $(\varphi(n_f), \varphi(n_t))$ is not present in R_{i+1}. R_{i+1} is obtained by adding all missing $(\varphi(n_f), \varphi(n_t))$ to R_i, for all possible rules, for all possible embeddings φ.
- $$\overline{R} = \bigcup_{i<\omega} R_i.$$

Definition 5.5.6 Let $\mathcal{R}$ be a set of rules. We call $\mathcal{R}$ *closed* if for every interpretation I, the closure $\overline{I}$ can be obtained in one iteration. So, for every interpretation I, it must be the case that $\overline{I} = (S, R_1, v)$.

We will now give the transformation into closed sets of rules, described in Step 7:

Theorem 5.5.7 Let $\mathcal{R}$ be a set of rules. There exists an equivalent closed set of rules $\overline{\mathcal{R}}$. Equivalent means: For every interpretation I the $\mathcal{R}$-closure equals the $\overline{\mathcal{R}}$-closure.

Proof
Let $\mathcal{R}$ be a set of rules. Start with $\overline{\mathcal{R}} = \mathcal{R}$. We make a list of all effects of $\mathcal{R}$ on trees. Let (S, R) be an arbitrary tree. Let $(S, \overline{R})$ be its closure. All branches $(s_1, s_2) \in \overline{R} \backslash R$ are added because of $\mathcal{R}$. If all rules $((S, R), s_1, s_2)$ are added to $\mathcal{R}$, then $\overline{\mathcal{R}}$ will have the same effect on T, as $\mathcal{R}$. This can be done for all trees. Then the resulting $\overline{\mathcal{R}}$ is closed, and for every interpretation I the $\mathcal{R}$-closure equals the $\overline{\mathcal{R}}$-closure.
End of Proof

In general it is not so sensible to construct this closed set of rules, because it is infinite. Fortunately it is not necessary to construct it. It is only necessary to know what branches will be added in the closure.

5.6 Resolution

We will now define how the semantic tableau rules can be obtained from a closed set of tree rules.

Definition 5.6.1 Let C be a set of modal clauses. Let $\mathcal{R}$ be a set of rules. We define the following semantic tableaux calculus $S_T = (P_T, \mathcal{R}_T)$ from $\mathcal{R}$, where P_T is the set of modal literals that can be built using the propositional symbols that occur in C and $\mathcal{R}_T$ contains the following rules:

Classical Rules

- For every propositional symbol, $\mathcal{R}_T$ contains the rule $\{P, \neg P\} \vdash \emptyset$.
- For every modal literal L, such that in the tree representation of L, there occurs a pair $P, \neg P$ in the label set of one node, $\mathcal{R}_T$ contains the rule $\{L\} \vdash \emptyset$.

Modal Rules The modal rules are defined in 4 steps:

1. We first define the so called *generating sequents*

$$A_1 \wedge \cdots \wedge A_p \vdash F,$$

with the A_i modal literals, and F a modal formula. $A_1, \ldots, A_p$, and F must satisfy the following condition:

There must be a rule $(T, w_f, w_t) \in \mathcal{R}$ and an embedding φ of T into the tree representation of $A_1 \wedge \cdots \wedge A_p$. There must be in $\Lambda(\varphi(w_f))$ a modal literal of the form $\Box X$, and F must be the result of adding X at $\varphi(w_t)$ in $A_1 \wedge \cdots \wedge A_p$. If F is not in NNF, then F is brought in NNF.

2. The second step consists of some cleaning: Let $A_1 \wedge \cdots \wedge A_p \vdash F$ be a generating sequent. Write $F = B_1 \wedge \cdots \wedge B_q$, with $q \geq 1$. In all generating sequents $A_1 \wedge \cdots \wedge A_p \vdash B_1 \wedge \cdots \wedge B_q$, the B_j that equal one of the A_i are deleted.
3. The third step consists of deleting the redundant generating sequents: All generating sequents $A_1 \wedge \cdots \wedge A_{p-1} \wedge A_p \vdash F$, for which there is a generating sequent $A_1 \wedge \cdots \wedge A_{p-1} \vdash F$, (with possible permutation of the A_i) are deleted.
4. The fourth step is the transformation into rules: For every generating sequent

$$A_1 \wedge \cdots \wedge A_p \vdash F,$$

and for every clause C in $< F >$, $\mathcal{R}_T$ contains the rules

$$\{A_1, \ldots, A_p\} \vdash C,$$

where $< F >$ is the clausal normal form of F that is defined in Theorem 5.3.4.

The classical rules are explained in Step 2 and Step 4 of Section 5.4.
It is possible to see the resolution rules, defined in Step 5 in Section 5.4, as a special case of the rules, defined in Step 6, using the K-rule $r = (w_1(w_2), w_1, w_2)$. So we will treat Step 5 and Step 6 not separately.

Example 5.6.2 Some instances of the 2nd classical rule are:

$$\{\Diamond(P \wedge \Box(Q \vee R) \wedge \neg P)\} \vdash \emptyset,$$

$$\{\Box P \wedge (Q \wedge R \wedge \neg Q)\} \vdash \emptyset.$$

The following are not instances:

$$\{\Box(P \wedge \neg P)\} \vdash \emptyset,$$

$$\{\Box \Diamond(P \wedge \neg P)\} \vdash \emptyset,$$

$$\{\Diamond(P \wedge \Diamond \neg P)\} \vdash \emptyset.$$

Assume that $\mathcal{R}$ contains $(w_0(w_f, w_t), w_f, w_t)$. Let

$$F = \Diamond(\Diamond \Box A \wedge \Diamond B) \wedge C.$$

The tree representation equals (T, Λ), with $T = n_0(n_1(n_2, n_3))$, and

$$\begin{array}{lcl} \Lambda(n_0) & = & \{C\}, \\ \Lambda(n_1) & = & \emptyset, \\ \Lambda(n_2) & = & \{\Box A\}, \\ \Lambda(n_3) & = & \{B\}. \end{array}$$

There are 5 embeddings possible:

$$\begin{array}{lllll} \varphi_0(w_0) = n_0 & \varphi_1(w_0) = n_1 & \varphi_2(w_0) = n_1 & \varphi_3(w_0) = n_1 & \varphi_4(w_0) = n_1 \\ \varphi_0(w_f) = n_1, & \varphi_1(w_f) = n_2, & \varphi_2(w_f) = n_2, & \varphi_3(w_f) = n_3, & \varphi_4(w_f) = n_3, \\ \varphi_0(w_t) = n_1, & \varphi_1(w_t) = n_2, & \varphi_2(w_t) = n_3, & \varphi_3(w_t) = n_2, & \varphi_4(w_t) = n_3, \end{array}$$

φ_1 and φ_2 allow construction of a generating sequent:

S_1 $\Diamond(\Diamond \Box A \wedge \Diamond B) \wedge C \vdash \Diamond(\Diamond(A \wedge \Box A) \wedge \Diamond B) \wedge C.$

S_2 $\Diamond(\Diamond \Box A \wedge \Diamond B) \wedge C \vdash \Diamond(\Diamond \Box A \wedge \Diamond(A \wedge B)) \wedge C,$

In both cases A has been added to $\varphi(w_t)$. In the second step, S_1 and S_2 are reduced to:

S_1' $\Diamond(\Diamond \Box A \wedge \Diamond B) \wedge C \vdash \Diamond(\Diamond(A \wedge \Box A) \wedge \Diamond B).$

S_2' $\Diamond(\Diamond \Box A \wedge \Diamond B) \wedge C \vdash \Diamond(\Diamond \Box A \wedge \Diamond(A \wedge B)),$

Unfortunately for S_1' and S_2', they will not survive the cleaning process of Step 3, because the C is redundant. The following generating sequent would survive:

S_1'' $\Diamond(\Diamond \Box A \wedge \Diamond B) \vdash \Diamond(\Diamond \Box A \wedge \Diamond(A \wedge B)),$

S_2'' $\Diamond(\Diamond \Box A \wedge \Diamond B) \vdash \Diamond(\Diamond(A \wedge \Box A) \wedge \Diamond B).$

Some more about the factoring in Step 4: The following generating sequents, which are all based on the K-rule, factor as follows:

$$\Diamond \top \wedge \Box(A \vee B) \vdash \Diamond(A \vee B),$$

becomes

$$\{\Diamond \top, \Box(A \vee B)\} \vdash \{\Diamond A, \Diamond B\}.$$

The following sequent, based on reflexivity, factors as follows:

$$\Box((A \vee B) \wedge (C \vee D)) \vdash (A \vee B) \wedge (C \vee D).$$

becomes

$$\{\Box((A \vee B) \wedge (C \vee D))\} \vdash \{A, B\}, \text{ and}$$
$$\{\Box((A \vee B) \wedge (C \vee D))\} \vdash \{C, D\}.$$

The sequent

$$\Box((A \wedge B) \vee (C \wedge D))\} \vdash (A \wedge B) \vee (C \wedge D),$$

factors into the rules:

$$\{\Box((A \wedge B) \vee (C \wedge D))\} \vdash \{A, C\},$$
$$\{\Box((A \wedge B) \vee (C \wedge D))\} \vdash \{A, D\},$$
$$\{\Box((A \wedge B) \vee (C \wedge D))\} \vdash \{B, C\},$$
$$\{\Box((A \wedge B) \vee (C \wedge D))\} \vdash \{B, D\}.$$

Theorem 5.6.3 The rules, defined in Definition 5.6.1 are sound: Let $\mathcal{R}_T$ be a set of semantic tableau rules obtained from a set of tree rules $\mathcal{R}$. If $\{A_1, \ldots, A_p\} \vdash \{B_1, \ldots, B_q\} \in \mathcal{R}_T$, and $I = (S, R, v)$ is an interpretation that satisfies $\mathcal{R}$, and I makes all A_i true in one world $s \in S$, then I makes one of the B_j true in s.

Proof
The soundness of the first classical rule is immediate. For the second we use Lemma 5.5.2. If L were true in a world in I, then P and $\neg P$, would have to be true in one world of I. This is not possible.
For the modal rules it is sufficient to prove the following: If $A_1 \wedge \cdots \wedge A_p \vdash B_1 \wedge \cdots \wedge B_q$ is a generating sequent, based on a rule r, and $I = (S, R, v)$ is a modal interpretation that satisfies r and which makes all A_i true in a world $s \in S$, then I makes all the B_j true in s. Let $I = (S, R, v)$ be a modal interpretation, satisfying $r = (T, w_f, w_t)$, (possibly the K-rule) Let $A_1 \wedge \cdots \wedge A_p \vdash B_1 \wedge \cdots \wedge B_q$ be a generating sequent, based on r. Suppose that $A_1 \wedge \cdots \wedge A_p$ is made true by I in a world $s \in S$. Let $(N, <, \Lambda)$ be the tree representation of $A_1 \wedge \cdots \wedge A_p$, let t be the root of $(N, <)$. There is the following:

1. By Lemma 5.5.2, there is an embedding ρ of $(N, <)$ into (S, R), such that $\rho(t) = s$, and for all nodes $n \in N$, the formulae in $\Lambda(n)$ are true in $\rho(n)$.

2. Also there is an embedding φ of T, into $(N, <, \Lambda)$, such that $\Lambda(\varphi(w_f))$ contains the $\Box X$ from which the sequent is obtained.

The combination $(\rho \cdot \varphi)$ is an embedding of T into (S, R), with the property that $\Box X$ is true in $(\rho \cdot \varphi)(w_f)$. Because ρ is an embedding, and I satisfies r, it must be the case that $((\rho \cdot \varphi)(w_f), (\rho \cdot \varphi)(w_t)) \in R$, and hence X must be true in $(\rho \cdot \varphi)(w_t)$). Then, using Lemma 5.5.2, in direction (2) $\Rightarrow$ (1), it follows that $B_1 \wedge \cdots \wedge B_q$ is true in s.
End of Proof

Next we prove the completeness. This is more involved.

Theorem 5.6.4 Let $\mathcal{R}$ be a closed set of rules, which contains the K-rule. Let $S_T = (P_T, \mathcal{R}_T)$ be obtained from $\mathcal{R}$ as in Definition 5.6.1. Let M be a set of modal literals, which is closed. There exists a modal interpretation $I = (S, R, v)$, such that I satisfies all rules in $\mathcal{R}$, and there is a world $\overline{s} \in S$, such that I makes all formulae $F \in M$ true in $\overline{s}$.

The rest of this section is devoted to the proof. We use the construction of Step 5. Let M be an interpretation of S_T. First we construct the following set:

$$\mathcal{M} = \{A_1 \wedge \cdots \wedge A_p \mid A_1, \ldots, A_p \in M, p > 0\}.$$

It is the set of finite conjunctions of modal literals in M. This does not fundamentally change the construction, but it removes some technical problems. This set makes it possible to use the generating sequents, as they are formed in Step one of Definition 5.6.1.

Definition 5.6.5 Let $A_1 \wedge \cdots \wedge A_p \vdash F$, be a generating sequent. A *choice* for F is obtained by applying the following rules as much as possible, but never inside the scope of a $\Box$.

$$\begin{array}{lcl} & & A \vee \bot \Rightarrow A, \\ & & A \vee \top \Rightarrow \top, \\ & & A \wedge \bot \Rightarrow \bot, \\ A \vee B \Rightarrow A, & & A \wedge \top \Rightarrow A, \\ A \vee B \Rightarrow B. & & \bot \vee A \Rightarrow A, \\ & & \top \vee A \Rightarrow \top, \\ & & \bot \wedge A \Rightarrow \bot, \\ & & \top \wedge A \Rightarrow A. \end{array}$$

So a choice is obtained by making a choice for every $\vee$, and doing some normalisation, to make sure that the result will be in NNF.

Lemma 5.6.6 Let $A_1 \wedge \cdots \wedge A_p \vdash F$ be a generating sequent. Let $\overline{F}$ be a choice for F. The tree representation of $\overline{F}$ can be obtained from the tree representation $(N, <, \Lambda)$ of F, by

1. making a choice for every formula in a $\Lambda(n)$, and
2. Continuing the process of Definition 5.5.1, if this is necessary.

As a consequence there is the following:

Lemma 5.6.7 Let $A_1 \wedge \cdots \wedge A_p \vdash F$, be a generating sequent. Let $\overline{F}$, be a choice for F. There exists a natural embedding ψ from the tree representation $(N, <, \Lambda)$ of F into the tree representation $(\overline{N}, \overline{<}, \overline{\Lambda})$ of $\overline{F}$, with the following additional property: If $I = (S, R, v)$, is a modal interpretation, and $n \in N$ is a node, and there is a world $s \in S$, such that I makes the formulae in the label set $\overline{\Lambda}(\psi(n))$ true in s, then I makes the formulae in the label set $\Lambda(n)$ true in s.

This follows from Lemma 5.5.2.

$\mathcal{M}$ has the following good property:

Lemma 5.6.8 If $A_1 \wedge \cdots \wedge A_p \vdash B_1 \wedge \cdots \wedge B_q$ is a generating sequent, and $A_1 \wedge \cdots \wedge A_p \in \mathcal{M}$, then there is a choice $\overline{B}_1 \wedge \cdots \wedge \overline{B}_q$ of $B_1 \wedge \cdots \wedge B_q$ in $\mathcal{M}$.

We will now prepare for the definition of $\sqsubset$ of Step 5. The formulae in $\mathcal{M}$ have tree representations. We assume that there are no overlapping nodes between tree representations of different conjunctions in $\mathcal{M}$.

Definition 5.6.9 Let $A_1 \wedge \cdots \wedge A_p$ and $\overline{B}_1 \wedge \cdots \wedge \overline{B}_q$ be conjunctions in $\mathcal{M}$, for which there is a generating sequent

$$A_1 \wedge \cdots \wedge A_p \vdash B_1 \wedge \cdots \wedge B_q,$$

such $\overline{B}_1 \wedge \cdots \wedge \overline{B}_q$ is a choice of $B_1 \wedge \cdots \wedge B_q$. Let ψ the embedding of $A_1 \wedge \cdots \wedge A_p$ into $\overline{B}_1 \wedge \cdots \wedge \overline{B}_q$, which is obtained as the concatenation of the embedding used in the construction of the generating sequent, and the embedding defined by Lemma 5.6.7. Then we define, for all nodes n in the tree representation of $A_1 \wedge \cdots \wedge A_p$,

$$n \sqsubset \psi(n).$$

Using this we define the sequences of Step 5: A $\sqsubset$-*sequence* is a (possibly finite, possibly infinite) sequence, $n_1 \sqsubset n_2 \sqsubset n_3 \sqsubset \cdots$.

Definition 5.6.10 We define the interpretation of Step 5. First an interpretation $I = (S, R, v)$ is constructed. After that $\overline{I}$ is obtained by replacing R by its closure.

- S is the set of all $\sqsubset$-sequences together with the set $\{r_0, r_1, r_2, \ldots\}$, consisting of the roots of the tree representations of the conjunctions in $\mathcal{M}$. The world obtained from the roots of the tree representations will be called $\overline{s}$.
- R is defined from $(s_1, s_2) \in R$ if there are two nodes $n_1 \in s_1$ and $n_2 \in s_2$ which occur in the tree representation of the same conjunction, and in this tree representation n_2 is a direct descendant of n_1.
- $\overline{R}$ is defined as the $\mathcal{R}$-closure of R.
- v is defined from: If P is a propositional symbol, that occurs in one of the conjunctions, then $v(P, s) = \mathit{true}$ iff s contains a node n and $P \in \Lambda(n)$.

Now 2 things must be proven:

1. $\overline{I} = (S, \overline{R}, v)$ satisfies $\mathcal{R}$, and
2. I makes all conjunctions $A_1 \wedge \cdots \wedge A_p \in \mathcal{M}$ true in $\overline{s}$.

(1) is trivial because $\overline{R}$ is defined as the the $\mathcal{R}$-closure of R. (2) is more difficult. (2) is proven by means of the following:

A If $n \in s$, and $s \in S$, and $F \in \Lambda(n)$, then $\overline{I}$ makes F true in s.

We will first prove **A**, making use of an assumption called B, by means of a simple induction argument. A implies (2), because of Lemma 5.5.2. The assumption **B** is:

B If $\overline{R}(s_1, s_2)$, $n_1 \in s_1$, $\Box A \in \Lambda(n_1)$, then there is an $n_2 \in s_2$, and a choice $A' \in \Lambda(n)$, such that This A' contains not more $\Box$'s than A.

We now prove **A** by induction on the number of $\Box$'s in F. Assume that $s \in S,\ n \in s$, and that $F \in \Lambda(n)$. Three cases are distinguished:

- F is a propositional symbol. By construction of v, it must be the case that $v(F, s) = \mathit{true}$ iff $F \in \Lambda(n)$.
- F is the negation of a propositional symbol P. By construction of v, it must the case that:

$$v(P, s) = \mathit{true} \text{ iff } P \in \Lambda(n).$$

Because $\neg P \in \Lambda(n)$, it is not possible that $P \in \Lambda(n)$, because otherwise the 2nd classical rule would be applicable, and M would be no interpretation of S_T. It follows that $v(P, s) = false$.

- F is of the form $\Box A$. It is here that **B** is used. By **B**, for all s_2, such that $R(s, s_2)$, there is an $n \in s_2$, and $A' \in \Lambda(n)$, such that A' propositionally implies A. Because of the induction hypothesis, it must be that A' is true in s_2.

This completes the proof of **A**.
It remains to prove **B**. Property **B** is enforced by the modal rules. First it should be noted that the whole process is monotonic:

- If $A_1 \wedge \cdots \wedge A_p \vdash B_1 \wedge \cdots \wedge B_q$ is a generating sequent, then the tree structure (T_A, Λ_A) of $A_1 \wedge \cdots \wedge A_p$ can be embedded, with embedding φ, into the tree structure (T_B, Λ_B) of $B_1 \wedge \cdots \wedge B_q$, and, for each node n in T_A,
$$\Lambda(n) \subseteq \Lambda(\varphi(n)).$$

This gives the following compactness property:

Lemma 5.6.11 Let T be a (finite) tree, which can be embedded into S. (with embedding φ). Let n be a node of T, and $w \in \varphi(n)$, and $A \in \Lambda(w)$. Then there exist a conjunction $A_1 \wedge \cdots \wedge A_p \in \mathcal{M}$, and an embedding ψ of T in the tree structure of $A_1 \wedge \cdots \wedge A_p$, such that $A \in \Lambda(\psi(n))$.

Proof
Because of the monotonicity, and the fact that only a finite number of events are needed.
Now it remains to prove **B**. Assume that $\overline{R}(s_1, s_2)$, $n_1 \in s_1$, and $\Box A \in \Lambda(n_1)$. Two cases can be distinguished:

1. Also $R(s_1, s_2)$. Then the tree $T = w_0(w_1)$ can be embedded into (S, R), with an embedding φ, and $\varphi(w_0) = s_1$ and $\varphi(w_1) = s_2$

2. Not $R(s_1, s_2)$. There is a rule $(T, n_f, n_t) \in \mathcal{R}$, such that T can be embedded in (S, R), with φ, and $\varphi(n_f) = s_1$, $\varphi(n_t) = s_2$.

In both cases there is a rule (T, n_f, n_t), such that

1. there is an embedding φ of T in (S, R) and

2. $n_1 \in \varphi(n_f)$, and $\Box A \in \Lambda(n_1)$.

Then there exist, by Lemma 5.6.11, a conjunction $A_1 \wedge \cdots \wedge A_p$ and an embedding ψ of T into the tree representation of $A_1 \wedge \cdots \wedge A_p$, such that $\Box A \in \Lambda(\psi(n_f))$.
There is a generating sequent

$$A_1 \wedge \cdots \wedge A_p \vdash B_1 \wedge \cdots \wedge B_q,$$

with $B_1 \wedge \cdots \wedge B_q$ obtained by adding A. This completes the proof.

Now we can combine Theorem 4.4.6, 5.6.3 and 5.6.4:

Theorem 5.6.12 If $\mathcal{R}$ is a closed set of rules, then the resolution calculus based on the semantic tableau rules described in Definition 5.6.1, is sound and complete for the set of interpretations that satisfy $\mathcal{R}$.

Ordering refinements and subsumption refinements are immediately obtained by the results in Section 4.5.
We will end with an example.

Example 5.6.13 Suppose we want to refute the following formula in the system B.

$$[\Diamond(\Box\, Q \wedge P) \wedge \Box(Q \wedge P)] \vee [\neg\, Q \vee \Box(\neg\, P \vee \neg\, Q)].$$

It can be transformed into the following two modal clauses:

$\{\Diamond(\Box\, Q \wedge P), \Diamond(Q \wedge P)\}$,
$\{\neg\, Q, \Box(\neg\, P \vee \neg\, Q)\}$.

System B is defined by the following closed set of tree rules:

$$\overline{R} = \{(w_0(w_1), w_1, w_0), (w, w, w)\}.$$

In order to obtain a complete semantic tableau system, it is necessary to add the K-rule: $(w_0(w_1), w_0, w_1)$.

We will construct a closed semantic tableau. In order to make the tableau not too large we construct four tableaux, one for combination of literals from the initial clause set.

(1)	$\Diamond(\Box\, Q \wedge P)$	(from 1st clause)
(2)	$\neg\, Q$	(from 2nd clause).
(3)	Q	(from 1)

The third clause is derived, using the generating sequent

$$\Diamond(\Box\, Q \wedge P) \vdash Q.$$

(1)	$\Diamond(\Box\, Q \wedge P)$	(from 1st clause)			
(2)	$\Box(\neg\, P \vee \neg\, Q)$	(from 2nd clause)			
(3)	$\Diamond(\Box\, Q \wedge P \wedge \neg\, P)$	(from 1 and 2)	(3)	$\Diamond(\Box\, Q \wedge P \wedge \neg\, Q)$	(from 2)
			(4)	$\Diamond(\Box\, Q \wedge P \wedge \neg\, Q \wedge Q$	(from 1)

The third clauses in both the left path and the right path are derived with the generating sequent

$$\Diamond(\Box\, Q \wedge P) \wedge \Box(\neg\, P \vee \neg\, Q) \vdash \Diamond(\Box\, Q \wedge P) \wedge (\neg P \vee \neg\, Q).$$

The fourth clause in the right path is derived with the generating sequent

$$\Diamond(\Box\, Q \wedge P \wedge \neg Q) \vdash \Diamond(\Box\, Q \wedge Q \wedge P \wedge \neg Q).$$

(1)	$\Box(Q \wedge P)$	(from 1st clause)
(2)	$\neg\, Q$	(from 2nd clause)
(3)	Q	(from 1)

The third clause is derived with the generating sequent

$$\Box(Q \wedge P) \vdash Q.$$

(1)	$\Diamond(Q \wedge P)$	(from 1st clause)			
(2)	$\Box(\neg\, P \vee \neg\, Q)$	(from 2nd clause)			
(3)	$\Diamond(Q \wedge P \wedge \neg\, P)$	(from 1 and 2)	(3)	$\Diamond(Q \wedge P \wedge \neg\, Q$	(from 1 and 2)

Both third clauses are derived with the generating sequent

$$\Diamond(Q \wedge P) \wedge \Box(\neg\, Q \vee \neg\, P) \vdash \Diamond(Q \wedge P \wedge (\neg\, Q \vee \neg\, P).$$

5.7 Conclusions

We have defined a general resolution calculus for propositional modal systems. This resolution calculus is weaker than axiomatic deduction systems for modal logics, because of two reasons:

- It is possible to define deduction systems for many non-conservative properties of the accessibility relation. An example are the diamond properties: For each i_1, i_2, j_1, j_2 there is a diamond property:

  $$\forall XY_1Y_2(R^{i_1}(X, Y_1) \wedge R^{i_2}(X, Y_2) \rightarrow \exists Z(R^{j_1}(Y_1, Z) \wedge R^{j_2}(Y_2, Z)).$$

A complete deduction system can be obtained by taking the corresponding axioms:

$$\Diamond^{i_1}\Box^{j_1}A \rightarrow \Box^{i_2}\Diamond^{j_2}A.$$

- For conservative properties of the accessibility relation, it is not necessary to use closed sets of rules in deduction systems. (See ([Nivelle92])).

We do not consider it likely that any of these weaknesses can be removed, (on the contrary to what is optimistically hoped for in ([Nivelle93a])), but we are not completely certain.

Chapter 6

Resolution Games

Resolution games have been introduced in ([Nivelle94b]) to model non-liftable orderings. Resolution games are non-deterministic ordering refinements. Non-deterministic refinements can be seen as a game, because the unpredictable element of the refinement can be seen as a counterplayer. As a consequence it is necessary, if one wants to prove the completeness of such a refinement, to prove that there exists a winning strategy against the counterplayer.
Resolution games have been constructed for the modelling of non-liftable orderings. These orderings may change when the literals are instantiated. This changing of a non-liftable ordering happens in a complicated manner and is almost unpredictable. It is therefore better to see the non-liftable ordering as non-deterministic ordering and to try to prove completeness for non-deterministic orderings.
There are essentially two types of the resolution game. The *blue* and the *yellow* resolution game. (Don't try to understand the colours. They are meaningless) There is another one, the *simultaneous* resolution game, but it is a variant of the blue resolution game.
In the next section we will begin by discussing the blue resolution game. In order to make the result as general as possible we will construct a transformation from semantic tableaux, using the framework of Section 4.4. It is not possible to transform a closed semantic tableau into a resolution game refutation, as in Section 4.4, because the actual refutation will depend on the counterplayer. However it is possible to transform a closed semantic tableau into a winning strategy of the resolution game. This transformation can be obtained in essentially the same manner as in Section 4.4, by proving a Subsumption Lemma, and a Combination Lemma for winning strategies. (See Lemma 4.4.17, and Lemma 4.4.19).

Because proofs of high complexity are not more than statistical evidence we will prove the completeness of the blue resolution game also in another manner, namely by using an adaptation of a proof in ([Bezem90], and [BG90]). This is done in Section 6.3.
After that we prove the completeness of the simultaneous resolution game. The simultaneous resolution game is a small adaptation of the blue resolution game, which is more adapted to some applications in the next chapter, (namely Section 7.4).
Finally we consider the yellow resolution game. The yellow resolution game is rather abstract and has no applications, at this moment. Nevertheless we think it is interesting. The general completeness question of the yellow resolution game is still open, but an important subclass is complete.

6.1 Blue Resolution Games

The blue resolution game is played by two players. One player will try to derive the empty clause, and the other will try to prevent this. The player who wants to derive the empty clause will be called the *opponent*. The player who wants to prevent this will be called *the defender*.
It is not possible to allow the defender to change the ordering in an unrestricted manner, because in that case completeness will be lost. (There will be an example later). For this reason the possibilities of the defender have to be restricted. In the blue resolution game the restriction is the following: Instead of allowing the defender to change the ordering, we attach indices to the literals and allow the defender to change the indices. There is one global ordering on the indexed literals, which is used all throughout the game. This ordering has to be well-founded, and the defender is allowed to make only replacements which make the literal decrease in the ordering.
We will begin by giving a formal definition of the blue resolution game. In order to obtain a more general completeness result the clauses will be replaced by multisets.
After that we will repeat some definitions of Section 4.4, but making some small adaptations, because of the indices, and because clauses are multisets instead of sets now. We define interpretations, resolvents and factors. There is no factorisation in Section 4.4 because there clauses have been defined as sets. After that we define how the defender can make replacements, and how the game is played.
In order to define a resolution game the following is necessary: A semantic tableaux calculus $S = (P, \mathcal{R})$, a set of potential indices, and a well-founded order on the indexed literals. This gives the following:

Definition 6.1.1 A *blue resolution game* is an ordered quadruple $\mathcal{G} = (P, \mathcal{R}, \mathcal{A}, \prec)$, where

- P is a set of propositional symbols,
- $\mathcal{A}$ is a set of attributes,
- $\mathcal{R}$ is a set of rules. The rules are of the form

$$[a_1, \ldots, a_p] \vdash [b_1{:}\, B_1, \ldots, b_q{:}\, B_q],$$

  where $p > 0$ and $q \geq 0$. The a_i and b_j are propositional symbols, The B_j are attributes.
- $\prec$ is an order on $P \times \mathcal{A}$. It must be the case that $\prec$ is well-founded on $P \times \mathcal{A}$.

The elements of $P \times \mathcal{A}$, written as $a{:}\, A$ are called *indexed literals*. A *clause* of $\mathcal{G}$ is a finite multiset of indexed literals of $\mathcal{G}$.

The notion of interpretation is essentially the same as in Definition 4.4.3, but it has to be adapted for multisets and the indices.

Definition 6.1.2 Let $\mathcal{G} = (P, \mathcal{A}, \mathcal{R}, \prec)$ be a resolution game. A set I is an interpretation of $\mathcal{R}$ iff whenever for a rule

$$[a_1, \ldots, a_p] \vdash [b_1{:}\, B_1, \ldots, b_q{:}\, B_q] \in \mathcal{R},$$

it happens that

$$\{a_1, \ldots, a_p\} \subseteq I,$$

then

$$\{b_1, \ldots, b_q\} \cap I \neq \emptyset.$$

If c is a clause of $\mathcal{G}$, then an interpretation I of $\mathcal{G}$ is a *model* of c if $I \cap c \neq \emptyset$. An interpretation is a model of a set of clauses if it is a model of every clause in the set. A set of clauses is *consistent* iff it has a model. Otherwise it is *inconsistent*.

Resolution is defined in essentially the same way as in Section 4.4. However because clauses are defined as multisets it is necessary to explicitly define factorisation.

Definition 6.1.3 Let $\mathcal{G} = (P, \mathcal{A}, \mathcal{R}, \prec)$ be a resolution game. Let c be a clause of $\mathcal{G}$. An indexed literal $a{:}\, A$ is *maximal* in c, if for no indexed literal $b{:}\, B$ in c,

$$a{:}\, A \prec b{:}\, B.$$

We define:

Resolution If $[a_1, \ldots, a_p] \vdash [b_1{:}\, B_1, \ldots, b_q{:}\, B_q] \in \mathcal{R}$, and $c_1, \ldots, c_p$ are clauses, such that

1. the c_i can be written as:

$$c_i = [a_i{:}\, A_i] \cup R_i,$$

2. the $a_i{:}\, A_i$ are maximal in c_i,

then the clause $R_1 \cup \cdots \cup R_p \cup [b_1{:}\, B_1, \ldots, b_p{:}\, B_p]$ is a resolvent of $c_1, \ldots, c_p$.

Factoring Let $c = [a_1{:}\, A_1, \ldots, a_p{:}\, A_p]$ be a clause, such that:

1. $a_1{:}\, A_1$ is maximal in c, and
2. $a_1 = a_i$, for an $i > 1$.

Then $c \backslash [a_i{:}\, A_i]$ is a factor of c.

We have now defined the armour of the opponent. Next we define the shield of the defender: The replacements that the defender can make, will be called *reductions*. The notion of reduction that we give here is very general. It is more general than the notion of ([Nivelle94b]). It is probably the most general possible notion of reduction.

Definition 6.1.4

Reduction Let $c = [a_1{:}\, A_1 \ldots, a_p{:}\, A_p]$ be a clause of a resolution game $\mathcal{G} = (P, \mathcal{A}, \mathcal{R}, \prec)$. A clause d is a *reduction* of a clause c if it satisfies the following conditions:

1. For every indexed literal $a{:}\, A'$ that occurs in d, there is an indexed literal $a{:}\, A$ in c. So in d no new literals are introduced.
2. For every indexed literal $b{:}\, B$ in d, either
   (a) there is an indexed literal $a{:}\, A \in c$, such that $b{:}\, B \prec a{:}\, A$, or
   (b) $b{:}\, B$ itself occurs in c, and $b{:}\, B$ occurs not more often in d, then it occurs in c.

This is a rather complicated definition but we give an example:

Example 6.1.5 Let $\mathcal{G}$ be a resolution game, with for every literal $a{:}\, A_1 \prec a{:}\, A_2$ iff $A_1 < A_2$. The clause $[a{:}\, 1, a{:}\, 1, a{:}\, 2, a{:}\, 2]$ is a reduction of $[a{:}\, 2, a{:}\, 2]$. If $a{:}\, 2$ is the maximal element of $[b{:}\, 2, a{:}\, 2]$, then the following clauses are reductions of this clause:

$[b{:}1, a{:}2]$, $[b{:}1, b{:}1, b{:}1, a{:}2]$,
$[b{:}1, a{:}1, a{:}2]$, $[b{:}1, b{:}1, a{:}1, a{:}2]$,
$[b{:}1, a{:}0]$, $[b{:}1]$,
$[b{:}2, b{:}2, b{:}2, a{:}2]$ $[\,]$.

If it is assumed that $b{:}2 \prec b{:}3 \prec a{:}2$, then the clause $[b{:}3, a{:}2]$ is also a reduction of $[b{:}2, a{:}2]$. The following clauses are not reductions of this clause:

$[b{:}1, c{:}2, a{:}2]$, because c is introduced
$[b{:}1, a{:}2, a{:}2]$, because the maximal element is repeated.

We have now the necessary means to define how the game is played:

Definition 6.1.6 Let C be a set of clauses of a resolution game $\mathcal{G} = (P, \mathcal{R}, \mathcal{A}, \prec)$. There are two players, the *opponent* and the *defender*. The opponent will try to derive the empty clause by computing factors and resolvents. The defender will try to prevent this by replacing newly derived clauses by reductions. During the game a list of derived clauses $\overline{X}_1, \ldots, \overline{X}_n$ is constructed. The game starts with $n = 0$, and the opponent on turn. The possible moves of the opponent are:

1. The opponent has the right to select an initial clause from C and hand it to the defender.
2. The opponent has the right to derive a new clause from the $\overline{X}_1, \ldots, \overline{X}_n$, either by ordered factorisation, or by ordered resolution, and hand it to the defender.

After that the defender is on turn:

3. The defender has to choose a reduction of the clauses that are handed to him. When this is done, the clause is added as $\overline{X}_{n+1}$.

So a *game situation* consists of two lists of clauses $X_1, \ldots, X_n$, and $\overline{X}_1, \ldots, \overline{X}_n$, with the following properties:

1. Each X_i is either an initial clause, or obtained by ordered factorisation or ordered resolution from the $\overline{X}_1, \ldots, \overline{X}_{i-1}$.
2. Each $\overline{X}_i$ is a reduction of X_i.

We can define a winning strategy:

Definition 6.1.7 A *winning strategy* W for a clause set C is a manner to derive the X_{i+1} is such a way, that whatever $\overline{X}_{i+1}$ are

chosen, the sequence will end in $X_n = [\,]$. A winning strategy for *game situation* $\overline{X}_1, \ldots, \overline{X}_n$ obtained from a clause set C, is a strategy to continue the game in such a manner that the empty clause will always be derived. A winning strategy W can be seen as a function such that each $W(C, \overline{X}_1, \ldots, \overline{X}_i)$ is equal to a clause which can be legally derived from $\overline{X}_1, \ldots, \overline{X}_i$, if $\overline{X}_i \neq [\,]$.

We have defined the resolution game in such a way that the defender can only affect newly derived clauses. We could also have defined the resolution game in such a way that the defender is allowed to reduce every clause. In that case the following Theorem 6.1.8 still holds.
The resolution game is different from lock or indexed resolution [Boyer71], because in lock resolution the resolvent inherits the indices from the parent clause without any changes. Here the indices may change. We state the completeness of the blue resolution game:

Theorem 6.1.8 Let C be a set of clauses of a resolution game $\mathcal{G}$.

1. If C is unsatisfiable, then there exists a winning strategy for the opponent.
2. If C is satisfiable then the defender can play in such a way that the opponent will not derive the empty clause.

We call the first part of the theorem *completeness*, and the second part *soundness*. The proof of the soundness is not difficult.

All the actions of the opponent are semantically sound. The defender can play in such a manner that his actions are sound, by never deleting a literal. This guarantees that the empty clause will not be derived if C is satisfiable.
We will end this section with an example, and prove the completeness in the next section.

Example 6.1.9 Let $\mathcal{G} = (P, \mathcal{A}, \mathcal{R}, \prec)$ be defined from:

- $P = \{a, b, c, \neg\, a, \neg\, b, \neg\, c\}$,
- $\mathcal{A} = \{0, 1, 2\}$,
- $\mathcal{R} = [a, \neg\, a] \vdash [\,], [b, \neg\, b] \vdash [\,], [c, \neg\, c] \vdash [\,]$.
- $\prec$ is defined from:

$$\begin{array}{lllllllllll}
\neg\, c{:}0 & \prec & b{:}0 & \prec & \neg\, a{:}0 & \prec & \neg\, a{:}1 & \prec & b{:}1 & \prec \\
c{:}0 & \prec & \neg\, c{:}1 & \prec & a{:}0 & \prec & c{:}1 & \prec & \neg\, c{:}2 & \prec \\
\neg\, b{:}0 & \prec & \neg\, a{:}2 & \prec & \neg\, b{:}1 & \prec & b{:}2 & \prec & c{:}2 & \prec \\
\neg\, b{:}2 & \prec & a{:}1 & \prec & a{:}2. & & & & &
\end{array}$$

Let C be the following unsatisfiable set of clauses:

$$[b\!:\!2, c\!:\!2, a\!:\!2] \quad [c\!:\!2, \neg\, b\!:\!2] \quad [\neg\, c\!:\!2] \quad [\neg\, a\!:\!2, b\!:\!2].$$

The clauses are sorted according to $\prec$. So each last literal is the selected literal. If the defender doesn't make any reductions then the resolvent $[\neg\, a\!:\!2, c\!:\!2]$ is possible. This clause can be reduced to for example $[\neg\, a\!:\!0, c\!:\!0]$, $[c\!:\!1, \neg\, a\!:\!2]$, or $[\neg\, a\!:\!0, c\!:\!2]$. The defender can also replace the initial clause $[c\!:\!2, \neg\, b\!:\!2]$ by $[\neg\, b\!:\!1, c\!:\!2]$. In that case the only possible resolvent is $[\neg\, b\!:\!1]$. Whatever reductions the defender makes, the empty clause can always be derived.

6.2 Completeness by Combination

We now give the first completeness proof for the blue resolution game. The first proof is close to the proof of Theorem 4.4.16 in Section 4.4. In Section 4.4 a semantic tree is recursively broken down into smaller semantic trees. From these semantic trees resolution refutations are constructed which are combined into a resolution refutation of the initial semantic tree. This is not possible here, because the actual refutation that will be constructed depends on the mood of the defender. It is possible however to recursively construct a *winning strategy* instead of one refutation. So we prove a combination lemma for winning strategies. It is not necessary to prove a subsumption lemma, because subsumption can be seen as a form of reduction, and hence it will be taken care of automatically. So the combination lemma alone makes the transformation which is described in the proof of Theorem 4.4.16, possible.

The proof method here has the advantage that it uses game directed methods. The proof in Section 6.3.2 is technically less complicated, but it is rather disappointing to prove the completeness of the resolution game in such a crude manner, without using games at all.
The technique that is used here, is the technique of obtaining winning strategies from other winning strategies, by playing two games simultaneously. We will explain the technique here in the simplest case: Suppose that one wants to obtain a winning strategy for a game $\mathcal{G}_2$, from a winning strategy for a game $\mathcal{G}_1$. Then one starts two games simultaneously, one as opponent of $\mathcal{G}_2$, and one as defender of $\mathcal{G}_1$, and proceeds as follows:

1. Wait for the move of the opponent on $\mathcal{G}_1$.

2. Copy the move of the opponent of $\mathcal{G}_1$ onto $\mathcal{G}_2$.

3. Wait for the defender of $\mathcal{G}_2$ to make a move.

4. Copy the move of the defender of $\mathcal{G}_2$ onto $\mathcal{G}_1$, and go to 1.

If the opponent of $\mathcal{G}_1$ possesses a winning strategy, then he will win $\mathcal{G}_1$. As a consequence $\mathcal{G}_2$ will be won. In this way a winning strategy for $\mathcal{G}_2$ has been obtained from a winning strategy of $\mathcal{G}_1$.
We will first apply this technique in order to prove that is sufficient to prove the completeness of resolution games $\mathcal{G} = (P, \mathcal{A}, \mathcal{R}, \prec)$ in which $\prec$ is a well-order. We will use the following:

Lemma 6.2.1 Let $\mathcal{G}_1 = (P, \mathcal{A}, \mathcal{R}, \prec_1)$ and $\mathcal{G}_2 = (P, \mathcal{A}, \mathcal{R}, \prec_2)$ be resolution games, such that $\prec_1 \subseteq \prec_2$. Let C be a set of clauses of $\mathcal{G}_1$ and $\mathcal{G}_2$. If there exists a winning strategy for C on $\mathcal{G}_2$, then there exists a winning strategy for C on $\mathcal{G}_1$.

Proof
We have:

COPY1 Every resolvent, or factor that can be computed with $\mathcal{G}_2$, can also be computed with $\mathcal{G}_1$ This is because an indexed literal, that is maximal w.r.t to $\prec_2$ will certainly be maximal w.r.t. $\prec_1$

COPY2 Every reduction that can be made with $\mathcal{G}_1$ is also a reduction with $\mathcal{G}_2$

Using this it is possible to copy moves of the opponent from $\mathcal{G}_2$ to $\mathcal{G}_1$, and the the moves of the defender from $\mathcal{G}_1$ to $\mathcal{G}_2$. Because of this a winning strategy W of C on $\mathcal{G}_2$ is a winning strategy of C on $\mathcal{G}_1$.
End of Proof

Because every well-founded order is included in a well-order (Lemma 2.2.3), it is sufficient to prove the completeness for those resolution games in which the order is a well-order. In the sequel we will assume that $\prec$ is a well-order.
In the proof of the combination lemma, the copying techniques will be more complicated. In order to win a game with $C \cup \{X \cup Y\}$ two simultaneous games are started, one with $C \cup \{X\}$, and one with $C \cup \{Y\}$.
Before the combination lemma is proven it is necessary to prove some properties of reduction and factorisation.

Lemma 6.2.2 Let c be a clause of a resolution game $\mathcal{G} = (P, \mathcal{A}, \mathcal{R}, \prec)$ Let C be the set of clauses that can be obtained by finitely often applying the following rules:

1. $c \in C$,

2. if $c_1, c_2 \in C$, then $(c_1 \cup c_2) \in C$,

3. if $c \in C$, and c' is a reduction of c, then $c' \in C$,

4. if $c \in C$, and c' is a $\prec$-ordered factor of c, then $c' \in C$.

Then, if $d \in C$, and d has no ordered factor, then d is a reduction of c.

Proof
We must check the two conditions of Definition 6.1.4.

1. It is easily seen that there are no new literals in d, because none of the rules introduces new literals.

2. For every clause $d \in C$, holds

   R1: For every literal $b{:}\,B \in d$, there is an $a{:}\,A \in c$, such that $b{:}\,B \preceq a{:}\,A$.

   This is the case because it holds for c, and because it is preserved by every of the 4 rules above. Now let d be a clause in C, which cannot be factored. Let $b{:}\,B \in d$. We show that if $b{:}\,B$ violates (a) in Definition 6.1.4, then $b{:}\,B$ must satisfy condition (b). Suppose there is no $a{:}\,A \in c$, such that $b{:}\,B \prec a{:}\,A$. Then $b{:}\,B$ itself must be in c, because of R1. It must be the case $b{:}\,B$ is maximal in d, because if there were an indexed literal $b'{:}\,B' \in d$, with $b{:}\,B \prec b'{:}\,B'$, there would be an $a'{:}\,A'$ in c with $b'{:}\,B \preceq a'{:}\,A'$, by R1, and it would be the case that $b{:}\,B \prec a'{:}\,A'$. Then $b{:}\,B$ would satisfy condition (a).

   Then it must be the case that $b{:}\,B$ is not-repeated in d, because d has no factors, and $b{:}\,B$ will certainly not violate condition (b) of Definition 6.1.4.

End of Proof

Using this the combination lemma can be proven. It is more complicated than the proof of Lemma 4.4.17.

Lemma 6.2.3 Let $\mathcal{G} = (P, \mathcal{A}, \mathcal{R}, \prec)$ be a resolution game, with $\prec$ a total order. Let C be a set of clauses of $\mathcal{G}$, let X and Y be clauses of $\mathcal{G}$. If there exists a winning strategy for $C \cup \{X\}$, and there exists a winning strategy for $C \cup \{Y\}$, then there exists a winning strategy for $C \cup \{X \cup Y\}$. Moreover this strategy can be effectively obtained by consulting the strategies for X and Y.

Proof
The strategy will be defined as a function Σ. Let

$$\Sigma(C; X; Y; \overline{X}_1, \ldots, \overline{X}_n; \overline{Y}[1], \ldots, \overline{Y}[n]; W_X; W_Y)$$

satisfy:

- W_X is a winning strategy for $C \cup \{X\}$,
- W_Y is a winning strategy for $C \cup \{Y\}$,
- The sequence $\overline{X}_1, \ldots, \overline{X}_n$ is a game situation, that is obtained by W_X. (It is not necessary to mention the X_i because they can be derived with W_X)
- The sequence $\overline{Y}[1], \ldots, \overline{Y}[n]$ consists of clauses that are a reduction of a multiple of Y. So each $\overline{Y}[i]$ is a reduction of a clause of the form $Y \cup \cdots \cup Y$.

In that case $\Sigma(C; X; Y; \overline{X}_1, \ldots, \overline{X}_n; \overline{Y}[1], \ldots, \overline{Y}[n]; W_X; W_Y)$ denotes a winning strategy from the following game situation:

- The initial clauses are the clauses in C.
- The first $n-1$ clauses of the game situation are the clauses $\overline{X}_1 + \overline{Y}[1], \ldots, \overline{X}_{n-1} + \overline{Y}[n-1]$, where $\overline{X}_i + \overline{Y}[i]$ is defined as

  $$\overline{X}_i \cup \overline{Y}[i] \text{ if } \mathrm{Max}(\overline{Y}[i]) \preceq \mathrm{Max}(\overline{X}_i),$$

  and

  $$\overline{X}_i \text{ if } \mathrm{Max}(\overline{X}_i) \prec \mathrm{Max}(\overline{Y}[i]).$$

- The n-th clause of the game situation equals $\overline{X}_n \cup \overline{Y}[n]$.

Then $\Sigma(C; X; Y; \ ; \ ; W_X; W_Y)$ is the strategy that we want.

First we show the following:

- There exists a winning strategy for the game situation $\overline{Y}[n]$.

In the sequel this winning strategy will be called W_2. If $\overline{Y}[n]$ is not a reduction of Y, then by Lemma 6.2.2 it is possible to compute a factor Z_1 of $\overline{Y}[n]$. The result will be reduced by the defender to a clause $\overline{Z}_1$. This process can be continued until a clause $\overline{Z}_i$ is obtained which cannot be factored anymore. This will eventually happen, because in every reduction or factorisation, either the maximal element, or the number of occurrences of

the maximal element decreases. Then, by Lemma 6.2.2, $\overline{Z}_i$ is a reduction of Y, and W_Y can be used to win.

Now suppose that an opponent is confronted with the game situation, which is given above. He proceeds as follows: If $\overline{X}_n$ is the empty clause, then $\overline{X}_n \cup \overline{Y}[n] = \overline{Y}[n]$. He can use W_2 to win the game. Otherwise two cases are distinguished:

1. $\mathrm{Max}(\overline{Y}[n]) \preceq \mathrm{Max}(\overline{X}_n)$. In that case $\Sigma(C; X; Y; \overline{X}_1, \ldots, \overline{X}_n; \overline{Y}[1], \ldots, \overline{Y}[n]; W_X; W_Y)$ can be obtained from $\Sigma(C; X; Y; \overline{X}_1, \ldots, \overline{X}_{n+1}; \overline{Y}[1], \ldots, \overline{Y}[n+1]; W_X; W_Y)$, because $\overline{X}_n \cup \overline{Y}[n]$ equals $\overline{X}_n + \overline{Y}[n]$, and it is possible to derive a clause of the form $\overline{X}_{n+1} \cup \overline{Y}[n+1]$, relying on W_X.

2. $\mathrm{Max}(\overline{X}_n) \prec \mathrm{Max}(\overline{Y}[n])$. Let W_1 denote the following strategy: $W_1 = \Sigma(C; X; Y; \overline{X}_1, \ldots, \overline{X}_n, \overline{X}_{n+1}; \overline{Y}[1], \ldots, \overline{Y}[n], [\,]; W_X; W_Y)$. It is winning strategy for a game situation based on C, with clauses $\overline{X}_1 + \overline{Y}[1], \ldots, \overline{X}_n + \overline{Y}[n], \overline{X}_{n+1}$. Let W_2 be as defined above. Then we apply Σ on W_1 and W_2 as follows: $W = \Sigma(C \cup \{\overline{X}_1 + \overline{Y}[1], \ldots, \overline{X}_{n-1} + \overline{Y}[n-1]\}; \overline{Y}[n]; \overline{X}_n; W_2; W_1)$. W is the strategy that we want, because it will win from a game situation containing formulae $C \cup \{\overline{X}_1 + \overline{Y}[1], \ldots, \overline{X}_{n-1} + \overline{Y}[n-1]\}, \overline{X}_n \cup \overline{Y}[n]\}$.

End of Proof

6.3 Direct Completeness

In this section we give a direct completeness proof of the resolution game. It is related to the completeness proofs in ([Bezem90]) and ([BG90]). In ([Bezem90]) the completeness of several refinements of resolution in classical logic are proven by showing that every set which is closed under the resolution rule, and does not contain the empty clause, has an interpretation. In ([BG90]) it is shown that every set which is closed under ordered resolution, and a certain restriction of paramodulation, which does not contain the empty clause, has an interpretation which satisfies the standard equality axioms. We will adapt these proofs to a completeness proof of the resolution game. In order to do this, it is necessary to prove that every closed set, which does not contain the empty clause, has a model in the sense of Definition 6.1.2. So the semantic tableaux are completely skipped. First it is necessary to adapt the notion of 'closed' to the resolution game, in order to take into account the reductions:

Definition 6.3.1 Let $\overline{C}$ be a set of clauses. We call $\overline{C}$ *closed* iff

1. For every $c_1, \ldots, c_p \in \overline{C}$, such that $c_1, \ldots, c_p$ have a resolvent d, there is a reduction d' of d in $\overline{C}$.
2. For every $c \in \overline{C}$, such that c has a factor d, there is a reduction d' of d in $\overline{C}$.

If $\overline{C}$ contains a reduction of every $c \in C$, then $\overline{C}$ is a *closure* of a clause set C

We will show that every closed set that does not contain the empty clause, is consistent. Then it follows that every closure $\overline{C}$ of an inconsistent clause set C contains the empty clause. This implies completeness because the opponent can play in such a manner that the resulting clause set will eventually be closed.
We adapt the proofs in two steps for the clarity of the presentation. We first give the proof for the case in which the defender never makes a reduction. In that case we have proven the completeness of a variant of lock resolution. After that we make some more adaptations to obtain the completeness of the full resolution game. We begin the first step:

6.3.1 Completeness of Restricted Resolution Games

If we consider games in which the defender never makes a reduction we have that $d = d'$, in both cases of Definition 6.3.1. We will prove that every closed clause set $\overline{C}$ in this restricted sense, which does not contain the empty clause, has a model.

Definition 6.3.2 Let $\overline{C}$ be a closed set of clauses. We define an *intersection set* of $\overline{C}$, as a set of indexed literals I, s.t. I contains an indexed literal of every $c \in \overline{C}$.

We will construct an intersection set I, s.t.

MAXUNIQUE for every $a{:}\,A \in I$, there is a clause $c \in \overline{C}$, such that $a{:}\,A$ is maximal in c, $a{:}\,A$ is not repeated in c, and there is no other indexed literal of c in I.

Lemma 6.3.3 If I is an intersection set of $\overline{C}$, and I satisfies MAXUNIQUE, then the set

$$[I] = \{a \mid a{:}\,A \in I, \text{ for an } A \in \mathcal{A}\}$$

is a model of $\overline{C}$.

Proof
Let $[a_1, \ldots, a_p] \vdash [b_1{:}B_1, \ldots, b_q{:}B_q] \in \mathcal{R}$, such that $\{a_1, \ldots, a_p\} \subseteq [I]$. Then there are $a_1{:}A_1, \ldots, a_p{:}A_p \in I$, and these occur in clauses which can be written as $[a_1{:}A_1] \cup R_1, \ldots, [a_p{:}A_p] \cup R_p$, where each $a_i{:}A_i$ is the maximal element of $[a_i{:}A_i] \cup R_i$ and $R_i \cap I = \emptyset$. Because of this a resolvent $R_1 \cup \cdots R_p \cup [b_1{:}B_1, \ldots, b_q{:}B_q]$ is possible, and so it is in $\overline{C}$. Because I is an intersection set, and $I \cap R_i = \emptyset$, it must be the case that $[b_1{:}B_1, \ldots, b_q{:}B_q] \cap I \neq [\,]$, and so $\{b_1, \ldots, b_q\} \cap I \neq [\,]$.
End of Proof

So what remains to show is that there exists an intersection set, satisfying MAXUNIQUE. We will construct this intersection set.

Lemma 6.3.4 Let $\overline{C}$ be a closed set (in which resolvents and factors are never reduced), s.t. $\emptyset \notin \overline{C}$. Then there exists an intersection set I of $\overline{C}$, that satisfies MAXUNIQUE.

Proof
Because $\prec$ is a well-order on the set of indexed literals it is possible to use recursion. Let λ be the ordinal length of $\prec$. Let $a_\alpha{:}A_\alpha$ be the α-th indexed literal, for $0 \leq \alpha < \lambda$. With I_α will be denoted the construction of the set I up to α.
The I_α are constructed as follow:

1. $I_0 = \emptyset$,
2. For any limit ordinal α, let
$$I_\alpha = \bigcup_{\beta<\alpha} I_\beta.$$
3. For any successor ordinal α, put
$$I_\alpha = I_{\alpha-1} \text{ if } I_{\alpha-1} \cup \{a_\beta{:}A_\beta \mid \alpha \leq \beta < \lambda\}$$
is an intersection set. Otherwise let
$$I_\alpha = I_{\alpha-1} \cup \{a_{\alpha-1}{:}A_{\alpha-1}\}.$$
(So at stage α it is decided whether or not $a_{\alpha-1}{:}A_{\alpha-1}$ is added)
4. Finally put $I = I_\lambda$.

We will show that I is an intersection set satisfying MAXUNIQUE.

Suppose that I is not an intersection set. Then there is a clause $c \in \overline{C}$, such that $I \cap c = \emptyset$. Let α be the index of the maximal literal in c. So $a_\alpha{:}A_\alpha$ is the maximal literal of c. Then at stage $\alpha + 1$ of the construction, $I_\alpha \cup \{a_{\alpha+1}{:}A_{\alpha+1}, a_{\alpha+2}{:}A_{\alpha+2}, \ldots\}$ is not an intersection set, and $a_\alpha{:}A_\alpha$ would have been added to I_α. This is a contradiction.
It remains to prove that I satisfies MAXUNIQUE. Suppose that I does not. Then I contains an indexed literal $a_{\alpha-1}{:}A_{\alpha-1}$ such that either

1. $a_{\alpha-1}{:}A_{\alpha-1}$ does not occur uniquely in a clause $c \in \overline{C}$. Then at stage α of the construction of I, $a_{\alpha-1}{:}A_{\alpha-1}$ would not have been added.

2. $a_{\alpha-1}{:}A_{\alpha-1}$ does occur uniquely in some clauses, but nowhere as maximal element. In that case the set $\{a_\beta{:}A_\beta \mid \alpha \leq \beta < \lambda\}$ contains all maximal elements of clauses in which $a_{\alpha-1}{:}A_{\alpha-1}$ uniquely occurs, and $a_{\alpha-1}{:}A_{\alpha-1}$ would not have been added at stage α.

3. $a_{\alpha-1}{:}A_{\alpha-1}$ occurs uniquely and maximally in a clause c, and as maximal element, but is repeated. In that case there is a (possible iterated) factor of c in $\overline{C}$, in which $a_{\alpha-1}{:}A_{\alpha-1}$ is not repeated.

End of Proof

6.3.2 Completeness of the Full Resolution Game

We will now adapt this proof to a completeness proof for the full resolution game. The first problem that we encounter is that Lemma 6.3.3 is not valid anymore, because the resolvent may be reduced. It is necessary to replace Definition 6.3.2 by

Definition 6.3.5 Let $\overline{C}$ be a closed set of clauses. An intersection set of $\overline{C}$ is a set of indexed literals, s.t.

1. If $a{:}A_1 \in I$, then for all $a{:}A_2$, such that $a{:}A_1 \prec a{:}A_2$, also $a{:}A_2 \in I$.
2. From every clause $c \in \overline{C}$, there is an indexed literal in I.

Then we can replace property MAXUNIQUE by

MAXUNIQUE2 For every $a{:}A_1 \in I$, for which there is no $a{:}A_2 \in I$, such that $a{:}A_2 \prec a{:}A_1$, there is a clause $c \in \overline{C}$, such that $a{:}A_1$ is maximal in c, $a{:}A_1$ is not repeated in c, and there is no other indexed literal of I in c.

It is possible to maintain Lemma 6.3.3 with a few adaptations in the proof:

Lemma 6.3.6 If I is an intersection set of a closed set $\overline{C}$, and I satisfies MAXUNIQUE2, then the set

$$[I] = \{a \mid a{:}A \in I, \text{ for an } A \in \mathcal{A}\}$$

is a model of $\overline{C}$

Proof
Let $[a_1, \ldots, a_p] \vdash [b_1{:}B_1, \ldots, b_q{:}B_q] \in \mathcal{R}$, and $\{a_1, \ldots, a_p\} \subseteq [I]$. There are indexed literals of the form $a_1{:}A_1, \ldots, a_p{:}A_p \in I$. Then I contains minimal indexed literals $a_1{:}\overline{A}_1, \ldots, a_p{:}\overline{A}_p$, for which $a_1{:}\overline{A}_1 \preceq a{:}A_1, \ldots, a_p{:}\overline{A}_p \preceq a_p{:}A_p$. For these literals there must be clauses of the form $[a_i{:}\overline{A}_i] \cup R_i$, such that $R_i \cap I = \emptyset$, and each $a_i{:}\overline{A}_i$ is maximal in $[a_i{:}A_i] \cup R_i$.
Because of this a resolvent $R_1 \cup \cdots \cup R_p \cup [b_1{:}B_1, \ldots, b_q{:}B_q]$ is possible, and there is a reduction of it in $\overline{C}$. Write $\overline{R}_1 \cup \cdots \cup \overline{R}_p \cup [\overline{b_1 B_1}, \ldots, \overline{b_q B_q}]$ for this reduction. Because I is an intersection set, I must contain an indexed literal of this clause. This indexed literal cannot be in one of the $\overline{R}_i$, because than I would contain a literal of R_i. Then it must be that I contains one of the $\overline{b_j B_j}$.
End of Proof

So it remains to show that there exists an intersection set, satisfying MAXUNIQUE2.

Lemma 6.3.7 Let $\overline{C}$ be a closed set of clauses, for which $[\,] \notin \overline{C}$. There exists an intersection set I of $\overline{C}$, that satisfies MAXUNIQUE2.

Proof
Let $\overline{C}_f \subseteq \overline{C}$ be the set of clauses of $\overline{C}$ with non-repeated maximal elements, i.e., the set of clauses that does not have a factor. We use the same recursion as in the proof of Lemma 6.3.4. Let λ be the length of $\prec$. Let $a_\alpha{:}A_\alpha$ be the α-th indexed literal, for $0 \leq \alpha < \lambda$. Let I_α be the construction of I up to α, (here $0 \leq \alpha \leq \lambda$) The construction goes as follows:

1. $I_0 = \emptyset$,

2. For any limit ordinal α, put

$$I_\alpha = \bigcup_{\beta<\alpha} I_\beta.$$

3. For any successor ordinal α do

(a) If there is an indexed literal $a_{\alpha-1}{:}\overline{A} \in I_{\alpha-1}$, such that $a_{\alpha-1}{:}\overline{A} \prec a_{\alpha-1}{:}A_{\alpha-1}$, then

$$I_\alpha = I_{\alpha-1} \cup \{a_{\alpha-1}{:}A_{\alpha-1}\}.$$

(b) Otherwise (if no such literal exists), then

i. if

$$I_{\alpha-1} \cup \{a_\beta{:}A_\beta \mid \alpha \leq \beta < \lambda\}$$

is an intersection set of $\overline{C}_f$, then $I_\alpha = I_{\alpha-1}$.

ii. otherwise $I_\alpha = I_{\alpha-1} \cup \{a_{\alpha-1}{:}A_{\alpha-1}\}$.

4. Finally we define $I = I_\lambda$.

It is not difficult to see that I is an intersection set of $\overline{C}$, because every intersection set of $\overline{C}_f$ is an intersection set of $\overline{C}$. We must show that I satisfies MAXUNIQUE2. Let $a{:}A_1$ be such that there is no $a{:}A_2 \in I$, for which $a{:}A_2 \prec a{:}A_1$ and despite this, there is no clause $c \in \overline{C}_f$, such that $a{:}A_1$ is maximal in c, and $a{:}A_1$ is the only literal of I in c. Let α be the moment at which adding of $a{:}A_1$ is decided, so $a{:}A_1 = a_{\alpha-1}{:}A_{\alpha-1}$. There are the following possibilities:

1. $a_{\alpha-1}{:}A_{\alpha-1}$ does not occur uniquely in a clause $c \in \overline{C}_f$. Then at stage α, $a_{\alpha-1}{:}A_{\alpha-1}$ would not have been added.

2. $a_{\alpha-1}{:}A_{\alpha-1}$ does occur uniquely in some clauses in $\overline{C}_f$, but nowhere as maximal element. In that case $I_{\alpha-1} \cup \{a_\beta{:}A_\beta \mid \alpha \leq \beta < \lambda\}$ is an intersection set, and $a_{\alpha-1}{:}A_{\alpha-1}$ would not have been added.

3. $a_{\alpha-1}{:}A_{\alpha-1}$ does occur uniquely and maximally in a clause $c \in \overline{C}_f$, but is repeated. This is impossible because of the nature of $\overline{C}_f$.

End of Proof

We will end by giving two examples demonstrating that the resolution game is not complete when the following conditions are dropped:

1. The condition that $a{:}A' \prec a{:}A$, when $a{:}A$ is replaced by $a{:}A'$ in a reduction.

2. The condition that $\prec$ is well-founded,

Example 6.3.8 Define $\mathcal{G} = (P, \mathcal{R}, \mathcal{A}, \prec)$ with

- $P = \{p, q, r\}$,

- $\mathcal{R} = \{[p, q] \vdash [\,], [q, r] \vdash [\,], [r, p] \vdash [\,]\}$,
- $\mathcal{A} = \{0, 1\}$,
- $\prec$ is defined from:

$$r{:}0 \prec q{:}0 \prec p{:}0 \prec r{:}1 \prec q{:}1 \prec p{:}1$$

The following clause set C is inconsistent:

$[q{:}0, p{:}0]$
$[r{:}0, q{:}0]$
$[p{:}0, r{:}1]$.

If the defender is allowed to replace $r{:}0$ by $r{:}1$, then every time the opponent derives a new clause, the defender can reduce it in such a manner that it is already present. Therefore there is no winning strategy for the opponent.

It is possible to modify this counterexample to a counterexample when $\prec$ is not well-founded. Define $\mathcal{G}' = (P', \mathcal{R}', \mathcal{A}', \prec')$, with $P' = P, \mathcal{R}' = \mathcal{R}$, and $\mathcal{A}' = \{0, -1 - 2, -3, \ldots\}$, and $\prec'$ is defined from:

$$, \ldots, r{:}i \prec' q{:}i \prec' p{:}i \prec' r{:}i+1 \prec' q{:}i+1 \prec' p{:}i+1 \prec' \cdots \prec' r{:}0 \prec' q{:}0 \prec' p{:}0.$$

Consider the following clause set $\overline{C} =$

$[q{:}0, p{:}0]$,
$[r{:}0, q{:}0]$,
$[p{:}-1, r{:}0]$,
$[q{:}-1, p{:}-1]$,
$[r{:}-1, q{:}-1]$,
$[p{:}-2, r{:}-1]$,
$\ldots$
$[q{:}-i, p{:}-i]$,
$[r{:}-i, q{:}-i]$,
$[p{:}-(i+1), r{:}-i]$.
$\ldots$

It is closed, inconsistent, and does not contain the empty clause.

6.4 Simultaneous Resolution Games

The simultaneous resolution game is related to the blue resolution game. A variant of it will be used in Section 7.4. Instead of allowing the defender to

replace indices he is allowed to replace orderings. Every time the opponent has computed a resolvent, the defender has the right to choose the ordering for the resolvent. If the opponent wants to use the resolvent he can only use the maximal elements under the ordering that the defender has selected.

Definition 6.4.1 A *simultaneous resolution game* $\mathcal{G} = (P, \mathcal{R}, \mathcal{O}, \sqsubset)$ consists of the following elements:

- P is a set of propositional symbols,
- $\mathcal{R}$ a set of rules of the form:

$$[a_1, \ldots, a_p] \vdash [b_1, \ldots, b_q].$$

- $\mathcal{O}$ defines a set of orderings on P. Every element of $\mathcal{O}$ is an ordered pair $(o_i, \prec_i)$, where o_i is an identifier, and $\prec_i$ is a total order on P. It is assumed that there are no $(o_i, \prec_i), (o_i, \prec_j) \in \mathcal{O}$ with $\prec_i \neq \prec_j$.
- $\sqsubset$ is well-order on O.

A clause of $\mathcal{G}$ is a finite multiset of literals.

The identifiers o_i have been added to the elements of $\mathcal{O}$, in order to make it possible that the same order of P, occurs at different places in $\sqsubset$. For every new clause the defender can choose an ordering by selecting an element of $\mathcal{O}$ for each derived clause. The selected ordering has to be $\sqsubseteq$-smaller than the orderings of the parent clauses.

Definition 6.4.2 Let $\mathcal{G} = (P, \mathcal{R}, \mathcal{O}, \sqsubset)$ be a simultaneous resolution game. Let C be a set of clauses. During the game the following things will be constructed:

1. A list of clauses $X_1, \ldots, X_n$.
2. A list of usable clauses $\overline{X}_1, \ldots, \overline{X}_n$.
3. A list $\alpha_1, \ldots, \alpha_n$ of elements of $\mathcal{O}$.

The elements α_i indicate how the clauses $\overline{X}_i$ are to be ordered. The clauses $\overline{X}_i$ are subsets of X_i. The game starts with the opponent. Begin with $n = 0$. We will explain how the $n + 1$-th clause is constructed.

- The opponent is allowed to do one of the following three things:
  - Select an initial clause $c \in C$, and add it as X_{n+1}.

  - Compute an ordered factor c from one of the $\overline{X}_i$, and add it as X_{n+1}. Here the opponent has to respect the ordering $\prec_j$, if $\alpha_i = (o_j, \prec_j)$.
  - Compute an ordered resolvent c from some of the clauses $\overline{X}_1, \ldots, \overline{X}_n$ and add it as X_{n+1}. For each clause X_j that he uses, the opponent has to respect the ordering $\prec_j$, if $\alpha_i = (o_j, \prec_j)$.
- After the opponent has constructed a new X_{n+1}, the defender has the right to do the following:
  1. First he can construct $\overline{X}_{n+1}$ by deleting some literals in X_{n+1}.
  2. After that he may choose an $\alpha_{n+1} \in \mathcal{O}$, for $\overline{X}_{n+1}$. There are some limitations on possible α_{n+1}, depending on how X_{n+1} is derived.
     (a) If X_{n+1} is an initial clause then the defender can choose every $\alpha_{n+1} \in \mathcal{O}$.
     (b) If X_{n+1} is a factor of $\overline{X}_i$, then it must be the case that $\alpha_{n+1} \sqsubseteq \alpha_i$.
     (c) If X_{n+1} is derived from clauses $\overline{X}_i$ by resolution, then it must be the case that $\alpha_{n+1} \sqsubseteq \alpha_i$, for each $\overline{X}_i$, that has been used.

A *winning strategy* for a simultaneous resolution game is defined in the same manner as for the blue resolution game.

We prove the soundness and completeness:

Theorem 6.4.3 Let C be a set of clauses of a simultaneous resolution game $\mathcal{G} = (P, \mathcal{R}, \mathcal{O}, \sqsubset)$. There exists a winning strategy for the opponent if and only if C is inconsistent.

Proof
Soundness is immediate. In order to prove the completeness we will simulate $\mathcal{G}$ with a blue resolution game $\mathcal{G}'$. Assume that C is inconsistent. Then there exists a closed semantic tableau that refutes C, and which uses $S = (P, \mathcal{R})$. In this semantic tableau only a finite number of propositional symbols occurs. We will construct a blue resolution game $\mathcal{G}'$ and a set of clauses C', and copy a winning strategy from this blue resolution game. The blue resolution game $\mathcal{G}'$ is defined as follows: Define $\mathcal{G}' = (P', \mathcal{A}', \mathcal{R}', \prec')$, where

- P' is the finite subset of P, which is used in the semantic tableau refutation of C.

- $\mathcal{A}' = \{o_i \mid (o_i, \prec_i) \in \mathcal{O}\} \cup \{\top\}$. $\top$ is intended as top element of the ordering $\prec'$.
- $\mathcal{R}'$ contains the rule $[a_1, \ldots, a_p] \vdash [b_1{:}\top, \ldots, b_q{:}\top]$, for every rule $[a_1, \ldots, a_p] \vdash [b_1, \ldots, b_q] \in \mathcal{R}$.
- $\prec'$ is defined from the following:

  $$a{:}A \prec' b{:}B \text{ if either}$$

  - A is of the form o_i, and $B = \top$,
  - A is of the form o_i, and B is of the form o_j, and $(o_i, \prec_i) \sqsubset (o_j, \prec_j)$,
  - $A = B$, and both can be written as o_i, and $a \prec_i b$, or
  - $A = B = \top$, and $a \prec_\top b$.

  Here $\prec_\top$ is an arbitrary, total order on P, which is added, in order to make $\prec'$ total.
- The initial clause set C' is defined from: For every clause $[a_1, \ldots, a_p] \in C$, C' contains the clause $[a_1{:}\top, \ldots, a_p{:}\top]$.

We have to show that $\mathcal{G}'$ is a correct game and that $\mathcal{G}'$ can be used to win $\mathcal{G}$.
In order to show that $\mathcal{G}'$ is a correct game it is sufficient to show that $\prec'$ is a well-order. This is not difficult to see, because $\sqsubset$ is a well-order, and the $\prec_i$ are well-orders, because P is finite.
We will now show how to imitate a strategy from $\mathcal{G}'$ on $\mathcal{G}$. We will play two games simultaneously, one as defender of $\mathcal{G}'$, and one as opponent of $\mathcal{G}$. During the games the following lists of clauses will be constructed on $\mathcal{G}'$:

$$X_1, \ldots, X_n, \text{ and } \overline{X}_1, \ldots, \overline{X}_n.$$

On $\mathcal{G}$ the following will be constructed:

$$Y_1, \ldots, Y_n, \ \overline{Y}_1, \ldots, \overline{Y}_n, \ \alpha_1, \ldots, \alpha_n.$$

We will preserve the following invariant:

INV All clauses $\overline{X}_i$ are of the form

$$[a_1{:}o_j, \ldots, a_p{:}o_j],$$

and for each $\overline{X}_i$, the corresponding $\overline{Y}_i$ equals $[a_1, \ldots, a_p]$, and $\alpha_i = (o_j, \prec_j)$.

As a consequence an indexed literal $a_\lambda{:}o_j$, (with $1 \leq \lambda \leq p$) is maximal in $\overline{X}_i$ iff a_λ is $\prec_j$-maximal in $\overline{Y}_i$. Then we proceed as follows:

1. (a) If the opponent on $\mathcal{G}'$ selects an initial clause $[a_1{:}\top, \ldots, a_p{:}\top]$, we select $[a_1, \ldots, a_p]$ on $\mathcal{G}$.
   (b) If the opponent on $\mathcal{G}'$ factors a clause $\overline{X}_i = [a_1{:}o_j, \ldots, a_p{:}o_j]$, we factor the corresponding $\overline{Y}_i = [a_1, \ldots, a_p]$ on $\mathcal{G}$.
   (c) If the opponent on $\mathcal{G}'$ resolves clauses $\overline{X}_{i_1}, \ldots, \overline{X}_{i_p}$, we resolve the corresponding $\overline{Y}_{i_1}, \ldots, \overline{Y}_{i_p}$ on $\mathcal{G}$.
2. After that we wait for the reductions on $\mathcal{G}$. The defender on $\mathcal{G}$ will select a subset $\overline{Y}_{n+1} \subseteq \overline{Y}_{n+1}$, and an $\alpha_{n+1} = (o_j, \prec_j)$, for a j. Then it is always possible to reduce X_{n+1} to a clause of the form:
$$\overline{X}_{n+1} = [a_1{:}o_j, \ldots, a_p{:}o_j].$$

Using this strategy, all we have to do is hire a qualified opponent of $\mathcal{G}'$, and we will win $\mathcal{G}$.
End of Proof

6.5 Yellow Resolution Games

At this moment yellow resolution games have no applications but they remain interesting. A yellow resolution game is obtained from a semantic tableau calculus by allowing infinite consequence sets in the rules. When such a rule is used, the result will be an infinite clause. The defender has the right (and the duty) to select a finite subset of each infinite clause. With this subset the opponent can continue the resolution process.
We will shortly explain where the idea of the yellow resolution game comes from. After that we introduce the yellow resolution game, and prove the completeness of the ordered yellow resolution game under the assumption that the order is well-founded. The general problem (without the restriction that the order is well-founded) is still open.
The original idea of the yellow resolution game was to have rules with multiple sets of consequences, and allow the defender to choose the set consequences that is to be used when the rule is applied. This would give the following type of rule:

$$\{A_1, \ldots, A_p\} \vdash \begin{array}{l} \{B_{1,1}, \ldots, B_{1,l_1}\}, \\ \{B_{2,1}, \ldots, B_{2,l_2}\}, \\ \cdots, \\ \{B_{n,1}, \ldots, B_{n,l_n}\}. \end{array}$$

Here all l_i are finite. When $\{A_1\} \cup R_1, \ldots, \{A_p\} \cup R_p$ are resolved, the defender may choose which of the possible $R_1 \cup \cdots \cup R_p \cup \{B_{i,1}, \ldots, B_{i,l_i}\}$ will be the result. If one wants to prove the completeness of resolution using this type of rules, then the following has to be proven: Let C be a clause set. If, for every possible selection of a consequence set for each rule it is possible to construct a closed semantic tableau, then there is a winning strategy for the opponent.
At this point there is some simplification possible: Let $\{A_1, \ldots, A_p\} \vdash C_1, \ldots, C_n$ be a rule with consequence sets $C_1, \ldots, C_n$. The fact that it is possible to construct a closed semantic tableau with any of the consequence sets means: Whatever consequence set C_i is chosen, whatever element of this consequence set is chosen, every path will close. This is the same as having one rule: $\{A_1, \ldots, A_p\} \vdash C_1 \cup \cdots \cup C_n$ and trying to construct a semantic tableau with this rule. As a consequence it is possible to replace all rules $\{A_1, \ldots, A_p\} \vdash C_1, \ldots, C_n$ by one rule $\{A_1, \ldots, A_p\} \vdash C_1 \cup \cdots \cup C_n$, which can be used instead of the rules with multiple consequence sets. Because in this type of rules the different C_i cannot be distinguished anymore, we have to give more rights to the defender: He is allowed to select any subset of the consequence of the rule.
This gives rise to the following definitions:

Definition 6.5.1 A *yellow resolution game* is a pair $\mathcal{G} = (P, \mathcal{R})$ with

- P is a set of propositional symbols,
- $\mathcal{R}$ is a set of rules $\{A_1, \ldots, A_p\} \vdash \overline{B}$. Here $\overline{B} \subseteq P$ is a possibly infinite set of propositional symbols.

A clause is a (possibly infinite) set $c \subseteq P$. An *interpretation* I of $\mathcal{G}$ is a subset of P, such that for every rule, if $\{A_1, \ldots, A_p\} \vdash \overline{B} \in \mathcal{R}$ and $\{A_1, \ldots, A_p\} \subseteq I$, then $\overline{B} \cap I \neq \emptyset$. I is a model of a clause c if $c \cap I \neq \emptyset$. The definitions of interpretation and model are the same same as in Definition 4.4.3. A set of clauses C is *consistent* iff it has a model. Otherwise it is inconsistent.

Definition 6.5.2 The (yellow) resolution game is played as follows: Let $\mathcal{G} = (P, \mathcal{R})$ be resolution game. Let C be a set of clauses. Like the blue resolution game, the yellow resolution game has 2 players: An *opponent* and a *defender*. During the game a list of clauses $\overline{X}_1, \ldots, \overline{X}_n$ is constructed. The game starts with the empty list.

1. The opponent has the right to select an initial clause, and hand it to the defender.
2. The opponent has the right to derive a new clause from the $\overline{X}_1, \ldots, \overline{X}_n$, either by resolution, or by factorisation. This clause has to be handed over to the defender.
3. The clauses that are handed over to the defender are possibly infinite. The defender has the right, (and the obligation) to select a finite subclause of the clause that is given to him. The selected subset has to be appended to $\overline{X}_1, \ldots, \overline{X}_n$ as $\overline{X}_{n+1}$.

As a consequence we have: A *game situation* consists of two lists of clauses

$$X_1, \ldots, X_n,$$

and

$$\overline{X}_1, \ldots, \overline{X}_n.$$

1. All $\overline{X}_i$ are finite, and each $\overline{X}_i \subseteq X_i$,
2. Every X_i is either,
   - An initial clause from C, or
   - a factor or resolvent, obtained from $\overline{X}_1, \ldots, \overline{X}_{i-1}$.

If $\overline{X}_n = \emptyset$, then the game is won by the opponent. A *winning strategy* (for the opponent) is a manner to derive the X_i in such a manner that whatever $\overline{X}_i$ are chosen, the empty clause will be derived. It can be seen as a total function from game situations to clauses that can be derived from these game situations. We define the *ordered yellow* resolution game by adding an order of P and using it in the standard manner.

Example 6.5.3 Define $\mathcal{G} = (P, \mathcal{R})$, where

- P consists of the literals

$$\{a_i \mid i \in \mathcal{N}\} \cup \{b_{i,j} \mid i, j \in \mathcal{N}\}$$

- $\mathcal{R}$ contains the following rules: For all $i \in \mathcal{N}$,

$$\{a_i\} \vdash \{b_{i,0}, \ldots, b_{i,j}, \ldots\} \in \mathcal{R}.$$

For all $i, j \in \mathcal{N}$,

$$\{a_i, b_{i,j}\} \vdash \emptyset \in \mathcal{R}.$$

The following clause set C is inconsistent:

$$C = \{a_0, a_1, \ldots, a_i, \ldots\}.$$

Theorem 6.5.4 Let C be a set of clauses of a yellow resolution game $\mathcal{G} = (P, \mathcal{R})$ with a well-founded order $\prec$. There exists a winning strategy for the opponent if and only if C is inconsistent.

We will here prove the soundness of the yellow resolution game. The completeness is proven in the next section.
Proving the soundness of the yellow resolution game is a bit more subtle than proving the soundness of the blue resolution game. In Theorem 6.1.8, the soundness is proven by having the defender never delete a literal, but this is not possible here.
Proof
Assume that C is consistent. Then there exists an interpretation I of C. The defender will play by always selecting a literal that is in I. As a result the clause set will consist of singleton clauses $\{A\}$, for which $A \in I$.

- If the opponent selects an initial clause c, the defender reduces it to $\{A\}$, for a literal $A \in I$. This is possible because $c \cap I \neq \emptyset$.
- If the opponent derives a new clause c, the defender reduces it to $\{B\}$, for a literal $B \in I$. This is possible because of the following: The opponent must have used a rule $\{A_1, \ldots, A_p\} \vdash \overline{B}$, for which all $\{A_1\}, \ldots, \{A_p\}$ are present. Then all A_i must be in I. Because I is an interpretation $I \cap \overline{B} \neq \emptyset$.

End of Proof

6.6 Completeness of Yellow Resolution Games

We give a direct completeness proof of the ordered yellow resolution game, similar to the completeness proof given in Section 6.3.2

Definition 6.6.1 Let $\overline{C}$ be a set of clauses of a resolution game $\mathcal{G} = (P, \mathcal{R})$. Let $\prec$ a well-founded order on P. We call $\overline{C}$ *closed* if

1. for every $c_1, \ldots, c_p \in \overline{C}$, such that c_1, up to c_p have a $\prec$-ordered resolvent d, there is a finite $d' \subseteq d$ in $\overline{C}$.

$\overline{C}$ is a *closure* of C if $\overline{C}$ contains a finite subset of every clause in C.

Definition 6.6.2 Let $\overline{C}$ be a closed set of clauses of $\mathcal{G} = (P, \mathcal{R})$ with $\prec$. An intersection set I of $\overline{C}$ is a set of elements of P, such that I contains a literal of every clause in $\overline{C}$.

The property MAXUNIQUE is defined in the same way as in 6.3.1. An intersection set I satisfies MAXUNIQUE if

MAXUNIQUE For every $A \in I$, there is a clause $c \in \overline{C}$, such that A is the maximal element of c, and no other element of I occurs in c.

Lemma 6.6.3 Every intersection set I, that satisfies MAXUNIQUE is an interpretation of $\overline{C}$.

Proof
Let $\{A_1, \ldots, A_p\} \vdash \overline{B}$ be a rule of $\mathcal{G}$. If all $A_i \in I$, then there are clauses $c_1, \ldots, c_p \in I$, such that each A_i is the maximal element of c_i, and no other element of I occurs in c_i. As a consequence

$$[(c_1 \backslash \{A_1\}) \cup \cdots \cup (c_p \backslash \{A_p\})] \cap I = \emptyset.$$

Because there is a clause $c \subseteq [(c_1 \backslash \{A_1\}) \cup \cdots \cup (c_p \backslash \{A_p\})] \cup \overline{B}$ in $\overline{C}$, for which $c \cap I \neq \emptyset$, it must be the case that $\overline{B} \cap I \neq \emptyset$.
End of Proof

It remains to show that such sets exist: The construction is analogous to the construction in the proof of Lemma 6.3.4.

Lemma 6.6.4 Let $\overline{C}$ be a minimal closure of a clause set C using a game $\mathcal{G} = (P, \mathcal{R})$ and a well-order $\prec$, such that $\emptyset \notin \overline{C}$. There exists an intersection set of $\overline{C}$, that satisfies MAXUNIQUE.

Proof
The proof could be almost he same as the proof of Lemma 6.3.4. Because it is boring to give the same proof twice we modify it. This brings it closer to the proof in ([Bezem90]).
Let I_1 and I_2 be two intersection sets of $\overline{C}$. Define $I_1 \sqsubset I_2$ as follows: Let A be the $\prec$-smallest literal where I_1 and I_2 differ. $I_1 \sqsubset I_2$ if $A \notin I_1$ and $A \in I_2$. Otherwise $I_2 \sqsubset I_1$. This is a well-order on the intersection sets. Let I be the $\sqsubset$-minimal intersection set. This is the set that we want. Suppose that I does not satisfy MAXUNIQUE. Then there is an $A \in I$ with one the following problems:

- A does not occur uniquely in any clause $c \in \overline{C}$. In that case $I \backslash \{A\}$ is an intersection set, with $I \backslash \{A\} \sqsubset I$, and so I is not $\sqsubset$-minimal

- A occurs uniquely in some clause in $\overline{C}$, but nowhere as maximal element. Let $M = \{\text{Max}(c) \mid c \in \overline{C} \text{ and } A \text{ occurs uniquely in } c\}$. Then $(I \backslash \{A\}) \cup M$ is also an intersection set, and $(I \backslash \{A\}) \cup M \sqsubset I$.

In both cases we have derived a contradiction.
End of Proof

The proof of Lemma 6.6.4 made use of the fact that $\prec$ is a well-order. We would like to know what happens if $\prec$ is an order, instead of a well-order.

Question 6.6.5 Let $\mathcal{G} = (P, \mathcal{R})$ be a yellow resolution game. Let $\prec$ be an order on P, instead of a well-order. Does Theorem 6.5.4 still hold?

6.7 Factorisation Causes Incompleteness

In this section we show that the fact that there exist incomplete ordering refinements is caused by factorisation. We do this by proving that for every clause set that has a resolution proof without factorisation, it is possible to derive the empty clause with every refinement that works by selecting literals from clauses. The proof is based on applying permutations to resolution proof trees.
We will consider clauses as multisets in this section, in order to make all applications of the factorisation rule explicit. In order to avoid excessive complexity of the permutations of resolution proofs we consider only binary resolution in classical logic,in this section, but all the results can be generalised to resolution, as defined in Section 4.4.
We begin by defining the permutations. There are 3 types of permutations:

1. Permutations that permute two applications of the resolution rule,

2. Permutations that permute an application of the resolution rule with a permutation of the factorisation rule,

3. Permutations that permute an application of the factorisation rule with an application of the factorisation rule.

Definition 6.7.1 In binary resolution the following permutations are possible: With A_1, A_2, we denote the pair $A, \neg A$, or $\neg A, A$. In the same way B_1, B_2 denotes $B, \neg B$, or $\neg B, B$.

RES

RES permutes two applications of the resolution rule:

$$\dfrac{\dfrac{[A_1, B_1] \cup R_1 \qquad [A_2] \cup R_2}{[B_1] \cup R_1 \cup R_2} \qquad [B_2] \cup R_3}{R_1 \cup R_2 \cup R_3}$$

permutes into

$$\dfrac{\dfrac{[A_1, B_1] \cup R_1 \qquad [B_2] \cup R_3}{[A_1] \cup R_1 \cup R_3} \qquad [A_2] \cup R_2}{R_1 \cup R_2 \cup R_3}$$

FACTRES1 The permutation FACTRES1 permutes an application of the resolution rule with an application of the factorisation rule, when the application of the factorisation rule is before the application of the resolution rule and the result of the factorisation rule is resolved upon by the resolution rule.

$$\dfrac{\dfrac{[A_1, A_1] \cup R_1}{[A_1] \cup R_1} \qquad [A_2] \cup R_2}{R_1 \cup R_2}$$

permutes into

$$\dfrac{\dfrac{\dfrac{[A_1, A_1] \cup R_1 \qquad [A_2] \cup R_2}{[A_1] \cup R_1 \cup R_2} \qquad [A_2] \cup R_2}{R_1 \cup R_2 \cup R_2}}{\dfrac{\cdots}{R_1 \cup R_2}}$$

Here one application of the factorisation rule is replaced by possibly many applications of the factorisation rule.

FACTRES2 The permutation FACTRES2 permutes an application of the resolution rule with an application of the factorisation rule, when the application of the factorisation rule is before the application of the resolution rule and the result of the factorisation rule is not resolved upon by the resolution rule.

$$\dfrac{\dfrac{[A, A] \cup [B_1] \cup R_1}{[A] \cup [B_1] \cup R_1} \qquad [B_2] \cup R_2}{[A] \cup R_1 \cup R_2}$$

permutes into

$$\dfrac{\dfrac{[A, A] \cup [B_1] \cup R_1 \qquad [B_2] \cup R_2}{[A, A] \cup R_1 \cup R_2}}{[A] \cup R_1 \cup R_2}$$

FACTRES3 Permutation FACTRES3 permutes an application of the resolution rule with an application of the factorisation rule, when the resolution rule is applied before the factorisation rule. It is the converse of FACTRES2.

FACT1 The permutation FACT1 permutes two applications of the factorisation rule, when the result of the first application is not factored upon by the second.

$$\dfrac{\dfrac{[A, A] \cup [B, B] \cup R}{[A] \cup [B, B] \cup R}}{[A] \cup [B] \cup R}$$

permutes into

$$\dfrac{\dfrac{[A,A] \cup [B,B] \cup R}{[A,A] \cup [B] \cup R}}{[A] \cup [B] \cup R}$$

FACT2 The permutation FACT2 permutes two application of the factorisation rule when the result of the first application is again factored upon in the second application:

$$\dfrac{\dfrac{[A^1, A^2, A^3] \cup R}{[A^{12}, A^3] \cup R}}{[A^{123}] \cup R}$$

permutes into

$$\dfrac{\dfrac{[A^1, A^2, A^3] \cup R}{[A^1, A^{23}] \cup R}}{[A^{123}] \cup R}$$

Here A^1, A^2, and A^3 are identical literals, but they have been indexed to make it possible to distinguish the different occurrences.

Theorem 6.7.2 All permutations of Definition 6.7.1 are correct, i.e., they transform correct resolution derivations into correct resolution derivations.

This can be easily checked.

For every clause set that has a resolution refutation it is possible to construct a proof tree. We define this formally:

Definition 6.7.3 A *proof tree* of C is a labelled tree $(N, <, \Lambda)$, of which the nodes are labelled with clauses, and which satisfies the following conditions:

1. If a node n has one successor n_1, then $\Lambda(n)$ can be obtained from $\Lambda(n_1)$ by factorisation.
2. If a node n has two successors n_1 and n_2 then $\Lambda(n)$ can be obtained from $\Lambda(n_1)$ and $\Lambda(n_2)$ by resolution.
3. If a node n is an end node, then $\Lambda(n) \in C$.
4. $(N, <)$ has one root, and this root is labelled with $\emptyset$.

So, if a clause is used more than once, it occurs more than once in the tree. Then, if n_2 and n_1 are nodes of the proof tree, and n_2 is a direct successor of n_2, then it is always possible to apply a permutation rule on n_1 and n_2. The only rule that makes the proof tree possibly grow is the rule FACTRES1. Using this, the following can be proven:

Theorem 6.7.4 Let C be a clause set, which has a resolution refutation without factorisation. Then with every refinement which works by selecting at least one literal from every derived clause, the empty clause will be derived from C.

Proof
C has a resolution refutation without factorisation. Construct a proof tree $(N, <, \Lambda)$ from this refutation. Now suppose that the refinement is violated somewhere. Then there exists a node $n \in N$, such that $\Lambda(n)$ is used improperly, and derivation up to the successors n_1 and n_2 (or only n_1) is without violation of the refinement, or n has no successors. Write $\Lambda(n) = [A] \cup [B] \cup R$, where A is the literal that is actually used, and B is the literal that should have been used instead. Then somewhere above n the literal A is resolved upon. Then it is possible, using permutation rule RES to permute this application downwards until it arrives in n. Because during this process the length of the proof tree does not increase, the number of abused clauses strictly decreases, and in the end all clauses are used correctly.
End of Proof

Corollary 6.7.5 Every refinement of the selection type is complete for all clause sets consisting of Horn clauses.

Proof
Because every unsatisfiable set of Horn clauses has a proof without factorisation.
End of Proof

Chapter 7

Non-liftable Orderings

In this chapter we will prove several completeness results on ordered resolution with non-liftable orderings. All proofs use resolution games. We will first give an example of a non-liftable ordering, then explain why it is desirable to have non-liftable orderings, and then give an idea of how the completeness results are obtained.
In Theorem 3.9.3, it was stated that ordered resolution is complete when the ordering satisfies the following condition, which was called liftability:

Liftability:

$$A \preceq B \Rightarrow A\Theta \preceq B\Theta.$$

This property is necessary for the proof of the Lifting lemma, which makes it possible to reconstruct ground refutations on the non-ground level. Liftable orders can be obtained for example by sorting literals on their predicate symbol.
We will now give an example of a non-liftable order.

Example 7.0.6 Let $\prec$ be defined from:

$$r(X) \prec q(X) \prec p(X) \prec p(0) \prec q(0) \prec r(0).$$

The order $\prec$ is clearly non-liftable, because $q(X) \prec p(X)$ and $p(0) \prec q(0)$. Using $\prec$, it is possible to resolve $\{\neg\, r(0)\}$ and $\{p(0), q(0), r(0)\}$ to $\{p(0), q(0)\}$, but it is impossible to resolve $\{\neg\, r(0)\}$ and $\{r(X), q(X), p(X)\}$.

We will explain why non-liftable orderings are useful. Orderings refinements are used for two purposes:

- To obtain a higher efficiency in proof search, i.e. when there exists a proof, this proof will be found more efficiently.
- To obtain a better behaviour when the clause set is satisfiable. In many cases it is possible to enforce termination with ordered resolution, when a clause set is satisfiable. In this way *decision procedures* are obtained. (See, for example ([Zam72]), or ([FLTZ93]).

For both purposes it is desirable to use non-liftable orderings.

1. In order to have the order as restrictive as possible, it is desirable to have it compare as many literals as possible. The liftability property is the only constraining factor in making the order more total, because without this property a total order would be possible.

   It is for example impossible to compare $p(X)$ and $p(s(Y))$ with a liftable order. Suppose that $p(X) \prec p(s(Y))$. Then it follows, using substitutions $\Theta_1 = \{X := s(0), Y := 1\}$, and $\Theta_2 = \{X := s(1), Y := 0\}$, that $p(s(0)) \prec p(s(1)) \prec p(s(0))$. In the same manner $p(s(Y)) \prec p(X)$ is not possible. With a non-liftable ordering more literals can be compared.

2. In some decision procedures, based on ordered resolution, a non-liftable ordering is necessary to ensure that the resolution process will stop when the initial clause set is satisfiable. In order to prove that resolution using this order is complete, it is necessary to have a technique to prove the completeness of resolution with non-liftable orderings.

Completeness results for non-liftable orderings can be obtained by using resolution games. It is for this reason that the resolution game has been introduced in ([Nivelle94b]). We will give a short idea how resolution games can be used to obtain completeness results for non-liftable orderings.
The standard manner to prove the completeness of an ordering refinement is as follows: (See, for example, the proof of Theorem 3.9.3) Let C be an unsatisfiable clause set. Then the following things are proven:

1. There exists a non-satisfiable set of ground instances of C.
2. This set has an ordered ground refutation.
3. From this ground refutation a non-ground refutation can be obtained by lifting. Here the property of liftability is used. This property ensures that maximal literals on the ground level, will be maximal on the non-ground level.

With a non-liftable order, step 3 is not possible anymore. As a consequence, if one wants to prove the completeness of non-liftable orderings, it is necessary to have a different technique. Of course one should try to solve this problem in a courteous way as follows:

1. Try to understand how the literals will be sorted during the resolution process, and

2. prove, using this deep and subtle insight, that there exists a ground refutation, which is ordered in the correct way to make lifting possible.

Unfortunately this project fails desperately at step (1). For this reason we try a villain strategy:

1. Don't try to understand how the literals will be sorted during the resolution process, but

2. prove that for every manner in which the literals can be sorted, he empty clause will be derived.

This approach turns out fruitful. It is possible to prove the completeness of a large class of non-liftable orderings using this method.
In order to be successful at step 2, for a given non-liftable order, it is necessary to prove the completeness of a class of non-deterministic orderings, and prove that the non-liftable order is included in this class of non-deterministic orderings. It is for this purpose that the resolution game has been introduced. In the next section we will give an example, after that we give the proofs.

7.1 Example

We will illustrate the general method how resolution games will be used with an example:
Suppose one wants to prove that the empty clause can be derived from the following clause set C, and making use of the following non-liftable order $\prec$:

$\{p(X,Y,Z), q(X,Y,Z), r(X,Y,Z)\}$,
$\{\neg\, p(0,Y,Z)\}$,
$\{\neg\, q(X,0,Z)\}$,
$\{\neg\, r(X,Y,0)\}$.

1. $X(\overline{A}) \prec Y(\overline{B})$ iff $\overline{A}$ contains more 0's than $\overline{B}$, (Here X, and Y are $\pm p, \pm q$, or $\pm r$)
2. $p(\overline{A}) \prec q(\overline{B}) \prec r(\overline{C})$ iff $\overline{A}, \overline{B}, \overline{C}$ contain no 0's.
3. $q(\overline{A}) \prec p(\overline{B}) \prec r(\overline{C})$ iff $\overline{A}, \overline{B}, \overline{C}$ contain one 0.
4. $r(\overline{A}) \prec p(\overline{B}) \prec q(\overline{C})$ iff $\overline{A}, \overline{B}, \overline{C}$ contains two 0's.
5. $p(\overline{A}) \prec r(\overline{B}) \prec q(\overline{C})$ iff $\overline{A}, \overline{B}, \overline{C}$ contain three 0's.

This order is clearly non-liftable. By Herbrands theorem there exists a set of ground instances of C, which is unsatisfiable. For example, the following set of ground instances C_{gr} of C is unsatisfiable:

$\{p(0,0,0), q(0,0,0), r(0,0,0)\}$,
$\{\neg\, p(0,0,0)\}$,
$\{\neg\, q(0,0,0)\}$,
$\{\neg\, r(0,0,0)\}$.

It is possible to index the literals in C_{gr} with the literals of which they are an instance. In this way the following clause set $\overline{C}$ is obtained:

$\{p(0,0,0){:}p(X,Y,Z), q(0,0,0){:}q(X,Y,Z), r(0,0,0){:}r(X,Y,Z)\}$,
$\{\neg\, p(0,0,0){:}\neg\, p(0,Y,Z)\}$,
$\{\neg\, q(0,0,0){:}\neg\, q(X,0,Z)\}$,
$\{\neg\, r(0,0,0){:}\neg\, r(X,Y,0)\}$.

The clauses in $\overline{C}$ can be ordered by applying $\prec$ on the indices. Then every time a new resolvent is computed, it is possible to unify the indices of the literals resolved upon. This can be seen as a reduction in a resolution game, when the order $\prec$ satisfies the condition $A\Theta \prec A$. Then Theorem 6.1.8, will guarantee that there exists a refutation which can be lifted.
In the first clause the literal $r(0,0,0){:}r(X,Y,Z)$ will be maximal. The resolvent with the clause

$$\{\neg\, r(0,0,0){:}\neg\, r(X,Y,0)\}$$

will be the clause

$$\{p(0,0,0){:}p(X,Y,0), q(0,0,0){:}q(X,Y,0)\}.$$

7.2 Preliminaries

In this section three things are done. First we make more formal the indexing technique of the previous section.

Second we consider the notion 'invariant under renaming'. If we want to use a non-liftable order $\prec$, then it is necessary that the order $\prec$ is invariant under renaming because otherwise $\prec$ is not well-defined since the theorem prover can freely replace variables during the deduction, and different theorem provers will do this in different manners. (For example the theorem prover may replace $\{p(X), q(X)\}$ by $\{p(Y), q(Y)\}$. If these clauses are ordered differently then the order is not well-defined) We will consider three reasonable notions of 'invariant under renaming' and prove they are equivalent.
Third we introduce a small modification of the resolution game which is more adapted to the application that we need here. We give this modification here instead of in Chapter 6, because it is not fundamental, but completely directed to the application.
We begin with the indexing:

Definition 7.2.1 Let $c = \{a_1{:}\, A_1, \ldots, a_p{:}\, A_p\}$ be an indexed clause. c is *representation-indexed* if there exists one substitution Θ, such that for all i, with $1 \leq i \leq p$, $A_i\Theta = a_i$.

Factorisation and resolution can be defined for representation-indexed clauses as follows:

Resolution If $c_1 = \{a_1{:}\, A_1, \ldots, a_p{:}\, A_p\}$, and $c_2 = \{b_1{:}\, B_1, \ldots, b_q{:}\, B_q\}$, and $a_1 = \neg\, b_1$, then

$$\{a_2{:}\, A_2\Theta, \ldots, a_p{:}\, A_p\Theta, b_2{:}\, B_2\Theta, \ldots, b_q{:}\, B_q\Theta\}$$

is a resolvent of c_1 and c_2, where Θ is the most general unifier of A_1 and B_1.

Factorisation If $c = \{a_1{:}\, A_1, \ldots, a_p{:}\, A_p\}$, and a_1 equals one of the a_i, then the clause

$$c' = \{a_1{:}\, A_1\Theta, \ldots, a_{i-1}{:}\, A_{i-1}\Theta, a_{i+1}{:}\, A_{i+1}\Theta, a_p{:}\, A_p\Theta\}$$

is a factor of c, where Θ is the most general unifier of A_1 and A_i.

In both cases the results are representation-indexed, because Θ is the most general unifier.
When a clause set C is unsatisfiable it is possible to construct an unsatisfiable set of ground instances and have it representation-indexed with the clauses in C.

Definition 7.2.2 If $C = \{c_1, \ldots, c_n\}$ is an unsatisfiable set of clauses, and

$$\Theta_{1,1}, \ldots, \Theta_{1,l_1}, \ldots, \Theta_{n,1}, \ldots, \Theta_{n,l_n}$$

is a list of substitutions such that the resulting clause set

$$\{c_1\Theta_{1,1} \ldots, c_1\Theta_{1,l_1}, \ldots, c_n\Theta_{n,1} \ldots, c_n\Theta_{n,l_n}\}$$

is unsatisfiable and ground, then $\overline{C}$ is defined as the following representation-indexed clause set: For every $c_i = \{A_1, \ldots, A_p\} \in C$, and substitution $\Theta_{i,j}$,

$$\{A_1\Theta_{i,j}{:}\, A_1, \ldots, A_p\Theta_{i,j}{:}\, A_p\} \in \overline{C}.$$

There is the following simple fact:

Lemma 7.2.3 Let C be an unsatisfiable set of clauses. Let $\overline{C}$ be defined as in Definition 7.2.2. Let $\prec$ be an order on predicate literals, which is extended to indexed literals as follows:

$$a{:}\, A \prec b{:}\, B \text{ iff } A \prec B.$$

Then if $\overline{C}$ has an ordered refutation, using the extension of $\prec$, the original clause set C has an ordered refutation, using $\prec$.

Proof
In the refutation of $\overline{C}$, replace every clause $\{a_1{:}\, A_1, \ldots, a_p{:}\, A_p\}$, by $\{A_1, \ldots, A_p\}$. It is easily checked that this gives a correct refutation.
End of Proof

This will be used in the sequel. We will construct a $\prec$-ordered refutation of $\overline{C}$, using a resolution game, which by Lemma 7.2.3 yields a $\prec$-ordered refutation of C.

Example 7.2.4 The following clause set is unsatisfiable:

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

The following set of representation-indexed ground clauses can be constructed:

$\{p(0){:}p(0)\}$
$\{\neg\, p(0){:}\neg\, p(X), p(s(0)){:}p(s(X))\}$
$\{\neg\, p(s(0)){:}\neg\, p(X), p(s(s(0))){:}p(s(X))\}$
$\{\neg\, p(s(0)){:}\neg\, p(X), p(s(s(s(0)))){:}p(s(X))\}$
$\{\neg\, p(s(s(s(0)))){:}\neg\, p(s(s(s(0))))\}$.

We will now consider different notions of 'invariant under renaming'. The following seems reasonable:

$$A \prec B \Rightarrow A\Theta \prec B\Theta,$$

for renamings Θ of A and B.
Unfortunately this condition has two possible meanings: The fact that $A\Theta$ is a renaming of A, means that A can be recovered from $A\Theta$ by a substitution. In the condition above, it is not specified if A and B have to be recovered by the same substitution, or if it is allowed to recover them by different substitutions. This gives two different notions of invariant under renaming. Consider, for example, $p(X) \prec q(Y)$, and the substitution $\Theta = \{X := Y\}$. Then $p(X)\Theta$ is a a renaming of $p(X)$ and $q(Y)\Theta$ is a renaming of $q(Y)$. There is no substitution Σ for which $p(Y)\Sigma = p(X)$, and $q(Y)\Sigma = q(Y)$, but $p(X)$ and $q(Y)$ can be recovered by different substitutions.
Things are made more complicated by the fact that in the proof we need the following notion:

$$A \prec B \Rightarrow A\Theta_1 \prec B\Theta_2$$

for renamings. This notion is stronger than the notions that we have considered until now.
For example if $p(X) \prec q(X)$, then the strongest notion demands that for all $X, Y : p(X) \prec q(Y)$, whereas for the weaker notions it is sufficient that for all $X : p(X) \prec q(X)$.
Fortunately it can be proven that these notions are equivalent in the following sense: Let $\prec$ be a maximal order satisfying one of the conditions above: Then $\prec$ satisfies all the other conditions. Because in general we are interested in maximal orders the notions (1-3) coincide.

Theorem 7.2.5 Consider the following notions of invariant under renaming:

1. If $A \prec B$, then for every pair of substitutions Θ_1, Θ_2, such that there exist Σ_1 and Σ_2, such that

   $$A\Theta_1\Sigma_1 = A, \text{ and } B\Theta_2\Sigma_2 = B,$$

   we have

   $$A\Theta_1 \prec B\Theta_2.$$

2. If $A \prec B$, then for every substitution Θ, for which there exist Σ_1 and Σ_2, such that
$$A\Theta\Sigma_1 = A, \text{ and } B\Theta\Sigma_2 = B,$$
we have
$$A\Theta \prec B\Theta.$$

3. If $A \prec B$, then for every substitution Θ, for which there exists a substitution Σ, such that
$$A\Theta\Sigma = A, \text{ and } B\Theta\Sigma = B,$$
we have
$$A\Theta \prec B\Theta.$$

Then for each pair i, j of these properties, every ordering satisfying (i) is contained in an ordering satisfying (j).

Proof
It is easily seen that (1) $\Rightarrow$ (2). (2) $\Rightarrow$ (3) follows from the fact that the condition of (3) implies the condition of (2). As a consequence every order $\prec_1$ satisfying (1) also satisfies (2), and every order $\prec_2$ satisfying (2) also satisfies (3).
We will show that every order $\prec_3$ satisfying (3) is contained in an order $\prec_1$ satisfying (1). Let $\prec_3$ be an order satisfying (3). Define $\prec_1$ from the following: $A \prec_1 B$ iff there are renamings $A\Theta_1$ of A, and $B\Theta_2$ of B, such that $A\Theta_1 \prec_3 B\Theta_2$.
It is easily seen that $\prec_3 \subseteq \prec_1$. We have to check that $\prec_1$ is an order and that $\prec_1$ satisfies (1). In order to prove that $\prec_1$ is an order we use the fact that $\prec_3$ cannot compare literals that are renamings of each other. This will be proven separately in Lemma 7.2.6.

O1 Suppose that $A \prec_1 A$. There exist renamings Θ_1 and Θ_2 of A, such that $A\Theta_1 \prec_3 A\Theta_2$. $A\Theta_1$ is a renaming of $A\Theta_2$. This is impossible because of Lemma 7.2.6.

O2 Assume that $A \prec_1 B$, and $B \prec_1 C$. There exist renamings $A\Sigma_1$ of A, and $B\Sigma_2$ of B, and $B\Theta_1$ of B, and $C\Theta_2$ of C, such that
$$A\Sigma_1 \prec_3 B\Sigma_2 \text{ and } B\Theta_1 \prec_3 C\Theta_2.$$
$B\Sigma_2$ and $B\Theta_1$ are renamings of each other because both are renamings of B. There exists one substitution Ξ, such that $B\Theta_1\Xi = B\Sigma_2$, and $C\Theta_2\Xi$ is a renaming of $C\Theta_2$, (and hence of C.) It follows by transitivity of $\prec_3$, that $A\Sigma_1 \prec_3 C\Theta_2\Xi$, and hence $A \prec_1 C$.

We check that $\prec_1$ satisfies (1).

(1) Suppose that $A \prec_1 B$, and that $A\Theta_1$ and $B\Theta_2$ are renamings of A and B. It is the case that $A \prec_1 B$ because of renamings $A\Sigma_1$ and $B\Sigma_2$, for which $A\Sigma_1 \prec_3 B\Sigma_2$. Because $A\Theta_1$ is a renaming of $A\Sigma_1$, and $B\Theta_2$ is a renaming of $B\Sigma_2$, also $A\Theta_1 \prec_1 B\Theta_2$.

End of Proof

This has as consequence that 1,2,3 coincide for maximal orderings.
In the proof of Theorem 7.2.5 we used the following property:

Lemma 7.2.6 Let $\prec$ be an order satisfying (3) in Theorem 7.2.5. It is impossible that $A \prec B$, and A is a renaming of B.

Proof
We prove this in two steps. First we prove a subcase, and then we prove that the general problem follows from the subcase. First we prove: It is impossible that $A \prec A\Theta$ for renamings Θ, for which for every variable V that occurs in A, either

1. $V\Theta = V$, or
2. $V\Theta$ does not occur in A.

Suppose there were such Θ. It is possible to add to Θ the reverse assignments: Define

$$\overline{\Theta} = \{V := W \mid V \neq W \text{ and } (V = W\Theta \text{ or } W = V\Theta)\}.$$

This is a correct substitution because $V \neq V\Theta$ implies that $V\Theta$ does not occur in A, and so it is safe to add assignments for these variables. Then:

$$\overline{\Theta}^2 = \{\}, \text{ and } A\overline{\Theta} = A\Theta.$$

It follows that $A \prec A\overline{\Theta}$, and $A\overline{\Theta} \prec A\overline{\Theta}^2$. By transitivity of $\prec$ it follows $A \prec A$, and this is impossible.
It remains to prove the general case. This is done by showing the following: If $A \prec A\Sigma$, for a renaming $A\Sigma$ of A, then there exists a substitution $\overline{\Sigma}$, such that A and $\overline{\Sigma}$ satisfy the first subcase.
Suppose that $A \prec A\Sigma$. If V is a variable in A, then, when Σ is iterated, one of the following two things will happen:

1. For all $i, V\Sigma^i$ occurs in A.
2. For one $i, V\Sigma^i$ is not in A.

Let $V_1, \ldots, V_n$ be the variables for which case 1 occurs. There must be periods $p_1, \ldots, p_n$, for which $V_i\Sigma^{p_i} = V_i$. Let p be smallest common period. Let W_1 be the set of variables that that occur in $A\Sigma$, but not in A.
Let $W_2, W_3, W_4, \ldots, W_p$ be disjoint sets of new variables with the same number of elements as W_1. All variables in W_i do not occur in A or $A\Sigma$. We write these sets as:

$$\begin{array}{lcl} W_1 & = & \{V_{1,1}, \ldots, V_{1,l}\}, \\ W_2 & = & \{V_{2,1}, \ldots, V_{2,l}\}, \\ & & \\ W_i & = & \{V_{i,1}, \ldots, V_{i,l}\}, \\ & & \\ W_p & = & \{V_{p,1}, \ldots, V_{p,l}\}. \end{array}$$

Then we define the following substitution Σ' by adding all assignments $V_{i,j} \Rightarrow V_{i+1,j}$ to Σ. So

$$\Sigma' = \Sigma \cup \{V_{i,j} \Rightarrow V_{i+1,j} \mid 1 \leq i < p \text{ and } 1 \leq j \leq l\}.$$

We define $\overline{\Sigma}$ as the p-th iteration of Σ'. For every variable V in A, either

1. $V\overline{\Sigma} = V$, or
2. $V\overline{\Sigma}$ is not in A.

Because $A \prec A'\Sigma$, it follows that

$$A \prec A\Sigma', A\Sigma' \prec A(\Sigma')^2, \ldots, A(\Sigma')^i \prec A(\Sigma')^{i+1}, \ldots, A(\Sigma')^{p-1} \prec A(\Sigma')^p.$$

By transitivity it follows that

$$A \prec A\overline{\Sigma}.$$

End of Proof

From now on, when we speak about 'invariant under renaming', we will always mean the first notion of Theorem 7.2.5.
The resolution game, as defined in Section 6.1 cannot be applied without adaptations, because of renaming substitutions. During the resolution process the theorem prover may freely replace $p(X)$ by $p(Y)$, and later replace it by $p(X)$ again. This would allow not well-founded reductions. This type of reductions is harmless because the position of $p(X)$ and $p(Y)$ in the order will be the same, and these reductions can be ignored. We define a variant of the resolution game in which this type of reductions is possible. For this we need:

Definition 7.2.7 A *weak order* $\preceq$ is a relation which has the following properties:

Reflexivity: $\forall d(d \preceq d)$.
Transitivity: $\forall d_1 d_2 d_3\ (d_1 \preceq d_2, d_2 \preceq d_3 \Rightarrow d_1 \preceq d_3)$.

We call two elements d_1 and d_2 *equivalent*, notation $d_1 \equiv d_2$, if $d_1 \preceq d_2$ and $d_2 \preceq d_1$. A weak order is *well-founded* iff there exists no infinite sequence:

$$d_1 \succeq d_2 \succeq d_3 \succeq \cdots \succeq d_i \succeq \cdots$$

with $d_1 \not\equiv d_2, d_2 \not\equiv d_3, \ldots, d_i \not\equiv d_{i+1}, \ldots$

Definition 7.2.8 We define the following variant of the resolution game: A resolution game is an ordered 4-tuple $\mathcal{G} = (P, \mathcal{A}, \mathcal{R}, \preceq)$ with

- P is a set of propositional symbols,
- $\mathcal{A}$ is a set of attributes,
- $\mathcal{R}$ is a set of rules, of the form: $\{a_1, \ldots, a_p\} \vdash \{b_1{:}\, B_1, \ldots, b_q{:}\, B_q\}$ (The rules consist of sets now, instead of multisets).
- $\preceq$ is a weak, well-founded order on $P \times \mathcal{A}$.

The game is played essentially the same as the standard resolution game, but there are the following modifications:

- A clause is defined as a set instead of a multiset.
- An indexed literal $A{:}\, a$ is considered maximal in a clause if $a{:}\, A \in c$, and for no $b{:}\, B \in c$, $a{:}\, A \preceq b{:}\, B$ and not $b{:}\, B \equiv a{:}\, A$.
- The definitions of ordered resolvent and ordered factor are the same, but using the new notion of maximality.
- A reduction of a clause c is obtained by replacing one or more literals $a{:}\, A_1$ by a literal $a{:}\, A_2$ with $a{:}\, A_2 \preceq a{:}\, A_1$.

This last definition of reductions is not the most general possible, but it is sufficient for the application.

Theorem 7.2.9 The resolution game, as defined in Definition 7.2.8, is complete.

Proof
By identifying all indexed literals $a{:}\, A_1$ and $a{:}\, A_2$ with $a{:}\, A_1 \equiv a{:}\, A_2$, this type of resolution game can be transformed into a standard type of resolution game.
End of Proof

7.3 Decreasing Orders

In this section we will give the first completeness theorem for non-liftable orderings. It is from ([Nivelle94b]). We begin by given some necessary definitions.

Definition 7.3.1 Let $A\Theta$ be a substitution instance of a literal A. If $A\Theta$ and A are not renamings of each other, then $A\Theta$ is called a *strict instance* of A.

Definition 7.3.2 Let $\overline{C}$ be a set of representation-indexed clauses. We define an *in-between literal* of $\overline{C}$ as a literal B, for which there is an indexed literal $a{:}A$ in $\overline{C}$, such that

1. a is an instance of B, and
2. B is an instance of A.

Lemma 7.3.3 If $\overline{C}$ is a set of representation-indexed clauses, then every order $\prec$ which is invariant under renaming defines a weak, well-founded order $\preceq'$ on the set of in-between literals of $\overline{C}$ as follows:

$$A \preceq' B \text{ iff}$$

1. $A \prec B$, or
2. A and B are renamings of each other.

Proof
We show that $\prec'$ is reflexive. Always $A \preceq' A$, because A is a renaming of A. In order to show that $\preceq'$ is transitive assume that $A \prec' B$, and $B \prec' C$.

- If $A \prec B$, and $B \prec C$, then $A \prec C$, by transitivity of $\prec$. Then also $A \preceq' C$.
- If $A \prec B$, and B is a renaming of C, then $A \prec C$, because $\prec$ is invariant under renaming. As a consequence $A \preceq' C$.
- If A is a renaming of B, and $B \prec C$, then $A \prec C$, because $\prec$ is invariant under renaming. As a consequence $A \preceq' C$.
- If A is a renaming of B, and B is a renaming of C, then A is a renaming of C. Then also $A \preceq' C$.

It remains to show that $\preceq'$ is well-founded. This is the case, because there are only a finite number of non-equivalent in-between literals of $\overline{C}$.
End of Proof

Theorem 7.3.4 Let $\prec$ be an ordering with the following properties:

1. If $A \prec B$, then for all renamings $A\Theta_1$ of A, and $B\Theta_2$ of B,

$$A\Theta_1 \prec B\Theta_2.$$

2. If $A\Theta$ is a strict instance of A, then $A\Theta \prec A$.

Then binary resolution and factoring using $\prec$ is a complete refinement of resolution.

Proof
The proof will use the method suggested by Lemma 7.2.3. Let C be an unsatisfiable clause set. Construct a set $\overline{C}$ of representation-indexed ground instances of C, in the manner of Definition 7.2.2. Then construct the following resolution game of the type defined in Definition 7.2.8: Define $\mathcal{G} = (P, \mathcal{RA}, \preceq)$, as follows:

- P is the set of ground literals that occur in $\overline{C}$.
- $\mathcal{R}$ contains the rule $\{A, \neg A\} \vdash \emptyset$, for every ground atom $A \in P$,
- $\mathcal{A}$ contains, for every indexed literal $a{:}\,A$ that occurs in $\overline{C}$, the set of all in-between literals of $a{:}\,A$.
- $\preceq$ is defined from: $a{:}\,A \preceq b{:}\,B$ iff
  1. $A \prec B$, or
  2. A and B are renamings of each other.

The initial clause set equals $\overline{C}$. Now two things must be proven:

1. $\mathcal{G}$ is a resolution game, in the sense of Definition 7.2.8, and
2. resolution and factoring, as defined in Definition 7.2.1, is a valid strategy of the defender of $\mathcal{G}$.

In order to prove (1) it is sufficient to prove that $\preceq$ is a weak, well-founded order. This follows easily from Lemma 7.3.3.

(2) follows from the fact that in Definition 7.2.1 indices are always replaced by instances. These replacements are valid reductions because always

$A\Theta \preceq A$.
End of Proof

Theorem 7.3.5 Let $\prec$ be an ordering with the following properties:

1. If $A \prec B$, then $A\Theta_1 \prec B\Theta_2$, for renamings $A\Theta_1$ and $B\Theta_2$.
2. If $A\Theta$ is a strict instance of A, then $A\Theta \prec A$.

Let I be an interpretation, as defined in Definition 3.9.1. Then $\prec$-ordered hyperresolution using I is a complete refinement of resolution.

The proof is obtained in the same manner as the proof of Theorem 7.3.4.

We will end this section by giving a maximal order, which satisfies the conditions of Theorem 7.3.4.

Definition 7.3.6 Assume that the variables are ordered as $V_0, V_1, V_2, \ldots, V_i, \ldots$, A literal A is *standardised* if the following holds:

- If the variable V_{i+1} occurs in A, then the variable V_i occurs in A and, when A is written in the standard notation, every occurrence of V_{i+1} is preceded by an occurrence of V_i.

For example $p(f(V_1))$ is not standardised, but $p(f(V_0))$ is standardised. $q(V_1, V_0)$ is not standardised, but $q(V_0, V_1)$ is standardised.

Lemma 7.3.7 Every literal A has exactly one renaming, that is standardised. This literal is called the *standardisation* of A.

Definition 7.3.8 Let $\sqsubset$ be an order on predicate symbols (possibly negated), function symbols, and constant symbols. We first extend $\sqsubset$ to terms, and then to literals.

1. If f is a function symbol, or constant, and V is a variable, then $f \sqsubset V$.
2. For all variables V_i and V_j, we define
$$V_i \sqsubset V_j \text{ iff } i < j.$$
3. If f and g are n and m-ary function symbols, (possibly with $n = 0$ or $m = 0$), and $f \sqsubset g$, then
$$f(t_1, \ldots, t_n) \sqsubset g(u_1, \ldots, u_m).$$

4. If f is an n-ary function symbol, and $f(t_1, \ldots, t_n) \neq f(u_1, \ldots, u_n)$, then let i the smallest integer, for which $t_i \neq u_i$. Then

$$f(t_1, \ldots, t_n) \sqsubset f(u_1, \ldots, u_n) \text{ iff } t_i \sqsubset u_i.$$

$\sqsubset$ is extended to literals by:

1. If f and g are (possibly negated) n and m-ary predicate symbols, and $f \sqsubset g$, then

$$f(t_1, \ldots, t_n) \sqsubset g(u_1, \ldots, u_m).$$

2. If f is an n-ary (possibly negated) predicate symbol, and $f(t_1, \ldots, t_n) \neq f(u_1, \ldots, u_n)$, then let i the smallest integer, for which $t_i \neq u_i$. Then

$$f(t_1, \ldots, t_n) \sqsubset f(u_1, \ldots, u_1) \text{ iff } t_i \sqsubset u_i.$$

Using this we define the following order $\sqsubset_s$ on literals: If A and B are literals then,

$$A \sqsubset_s B \text{ iff } A' \sqsubset B',$$

where A' is the standardisation of A, and B' is the standardisation of B.

Theorem 7.3.9 Let $\sqsubset$ be a total order on the (possibly negated) predicate symbols, function symbols, and constant symbols. Then the order $\sqsubset_s$, as defined in Definition 7.3.7 is an order that satisfies the conditions of Theorem 7.3.4, and $\sqsubset_s$ is total in the following sense: If A is not a renaming of B, then $A \sqsubset_s$ or $B \sqsubset_s A$.

7.4 Decomposable Clauses

In this section we prove a completeness theorem for non-liftable orderings on decomposable clause sets. This result is especially important because many decidable classes are based on decomposable clause sets. A clause is a *decomposable clause* if all its literals either have no overlapping variables, or exactly the same variables. For such clause sets condition (2) in Theorem 7.3.4 can be dropped. This is caused by the fact that when a literal A increases its position in the ordering, during a substitution, then all literals which have the same variables, change. Because of this solidarity it is possible to have the other literals decrease, instead of A increase.

Definition 7.4.1 Let A be a literal. $\mathrm{Var}(A)$ is defined as the set of variables, that occur in A.

A clause c is *decomposable* if for all $A, B \in c$, either $\mathrm{Var}(A) = \mathrm{Var}(B)$, or $\mathrm{Var}(A) \cap \mathrm{Var}(B) = \emptyset$.

The maximal subsets $\overline{c} \subseteq c$, with $A, B \in \overline{c} \Rightarrow \mathrm{Var}(A) = \mathrm{Var}(B)$ are called *components*.

For example $\mathrm{Var}(p(X, Y, f(A))) = \{X, Y, A\}$. The clause $\{p(X), q(Y), r(Y)\}$ is decomposable. The components are $\{p(X)\}$ and $\{q(Y), r(Y)\}$. The clause $\{p(X, Y), q(Y, Z)\}$ is not decomposable. The clause $\{p(X, Y), q(X, Y)\}$ is decomposable.
We will now prove that sets of decomposable clauses are closed under resolution and factoring.

Lemma 7.4.2 Let A and B be two literals, with $\mathrm{Var}(A) = \mathrm{Var}(B)$. Let Θ be a substitution. Then $\mathrm{Var}(A\Theta) = \mathrm{Var}(B\Theta)$.

Lemma 7.4.3

1. Every factor $c\Theta$ of a decomposable clause c is decomposable. If c consists of one component, then $c\Theta$ consists of one component.

2. If $c_1\Theta \cup c_2\Theta$ is resolvent of two decomposable clauses $\{A_1\} \cup c_1$ and $\{\neg\ A_2\} \cup c_2$, then $c_1\Theta \cup c_2\Theta$ is a decomposable clause. If $\{A_1\} \cup c_1$ and $\{\neg\ A_2\} \cup c_2$ consist of one component, then $c_1\Theta \cup c_2\Theta$ consists of one component.

Proof

1. This follows immediately from Lemma 7.4.2. The substitution Θ will affect only one component.

2. Assume first that $\{A_1\} \cup c_1$ and $\{\neg\ A_2\} \cup c_2$ are one component clauses. By Lemma 7.4.2, $c_1\Theta$ will have exactly same variables as $A_1\Theta$, and $c_2\Theta$ will have exactly the same variables as $A_2\Theta$. Because $A_1\Theta = A_2\Theta$, all literals in $c_1\Theta \cup c_2\Theta$ will have exactly the same variables. Now if c_1 or c_2 consists of more than one component, the other components will not be affected, and copied into the resolvent.

End of Proof

We will use the modification of the resolution game of Definition 7.2.8. The clauses will be indexed slightly differently than in Section 7.3. A literal will

be indexed with a pair (A, Θ), where A is a predicate literal, and Θ is a substitution. We now give the corresponding adaptation of Definitions 7.2.1 and 7.2.2, and after that we give a small modification of Lemma 7.2.3.

Definition 7.4.4 Let c be an indexed clause of the form $\{a_1{:}(A_1, \Theta_1), \ldots, a_p{:}(A_p, \Theta_p)\}$. Clause c is representation-indexed if all Θ_i are equal, and for all i, we have $a_i = A_i\Theta$.

Resolution, and factoring are defined as in Definition 7.2.1, but the substitutions are also modified when a resolvent or factor is computed.

Example 7.4.5 The clause

$$c_1 = \{p(0,1){:}(p(X,Y), \{X := 0, Y := 1\}), q(0,1){:}(p(X,Y), \{X := 0, Y := 1\})\}$$

is representation-indexed. The clause

$$c_2 = \{\neg\, p(0,1){:}(p(X,1), \{X := 0\})\}$$

is representation-indexed. The resolvent equals

$$c = \{q(0,1){:}p(X,1), \{X := 0\})\}$$

Using this new notion, Definition 7.2.2 can be adapted:

Definition 7.4.6 Let $C = \{c_1, \ldots, c_n\}$ be an unsatisfiable set of clauses. We write $\overline{C}$, for a set of representation-indexed clauses that is obtained as follows: Let

$$\Theta_{1,1}, \ldots, \Theta_{1,l_1}, \ldots, \Theta_{n,1}, \ldots, \Theta_{n,l_n}$$

be a list of substitutions, such that

$$\{c_1\Theta_{1,1}, \ldots, c_1\Theta_{1,l_1}, \ldots, c_n\Theta_{n,1}, \ldots, c_n\Theta_{n,l_n}\}$$

is unsatisfiable. Then, for each $c_i = \{A_1, \ldots, A_p\}$ and $\Theta_{i,j}$,

$$\{A_1\Theta_{i,j}{:}(A_1, \Theta_{i,j}), \ldots, A_p\Theta_{i,j}{:}(A_p, \Theta_{i,j})\} \in \overline{C}.$$

Lemma 7.4.7 Let C be an unsatisfiable set of clauses. Let $\overline{C}$ be obtained as in Definition 7.4.6. Let $\prec$ be an order on predicate literals, which is extended to indexed literals as follows:

$$a{:}(A, \Theta) \prec b{:}(B, \Theta) \text{ iff } A \prec B.$$

Then, if $\overline{C}$ has an ordered refutation, using the extension of $\prec$, then C has an ordered refutation, using $\prec$ itself.

We give the main theorem of this section:

Theorem 7.4.8 Let C be a set of decomposable clauses. Let $\prec$ be an order satisfying:

- If $A \prec B$ then for all renamings $A\Theta_1$ of A, and $B\Theta_2$ of B,

$$A\Theta_1 \prec B\Theta_2.$$

Then, if C is unsatisfiable, it is possible to derive the empty clause with ordered resolution and factoring based on $\prec$.

Proof
The proof will be in two steps:

1. First the completeness will be proven in the case that C consists of one component clauses.

2. Second the full completeness will be proven.

For step one the following resolution game will be used. Let C be a set of clauses that is unsatisfiable, and that consist of one component. Let $\overline{C}$ obtained as in Definition 7.4.6. We construct the following resolution game $\mathcal{G} = (P, \mathcal{R}, \mathcal{A}, \preceq)$ of the type of Definition 7.2.8 as follows:

- P is the set of ground literals that occur in $\overline{C}$.

- $\mathcal{R}$ is the set that consists of the following rules: For each ground literal A that occurs in $\overline{C}$,
$$\{A, \neg A\} \vdash \{\} \in \mathcal{R}.$$

- $\mathcal{A}$ contains of all pairs (A, Θ), where A is a literal, and Θ is a substitution, which satisfy the following conditions:

  1. A is an in-between literal of $\overline{C}$.
  2. Θ is an in-between substitution. i.e. a substitution, which makes an in-between literal equal to the ground literal to which it belongs. It is assumed that Θ has no redundant assignments, so Θ, makes no assignments to variables that do not occur in A.

- $\preceq$ is defined from:

$$a_1{:}(A_1, \Theta_1) \preceq a_2{:}(A_2, \Theta_2)$$

  if one of the following:

1. There is a Σ, such that $\Theta_1 = \Theta_2 \cdot \Sigma$, and there is no Σ', such that $\Theta_2 = \Theta_1 \cdot \Sigma'$.
2. There are Σ_1 and Σ_2, such that

$$\Theta_1 = \Theta_2 \cdot \Sigma_1, \text{ and } \Theta_2 = \Theta_1 \cdot \Sigma_2,$$

and ($A_1 \prec A_2$ or A_1 is a renaming of A_2)

We must show that this is a correct resolution game. For this, it is sufficient to show that $\preceq$ is a weak, well-founded order. Reflexivity is immediate, Transitivity follows from a case analysis. $\preceq$ is well-founded because the sets of non-equivalent in-between literals, and non-equivalent in-between substitutions are finite. It is easily seen that resolution and factoring, as defined in Definition 7.4.4 is a correct strategy of the defender.

It remains to prove the full completeness in the case that the clauses do not consist of one component. This can be proven by noticing that the different components in a clause are completely independent. Because of this they can be treated as literals. Then the results of Section 4.5 can be applied.
End of Proof

With Theorem 7.4.8, an open question in ([FLTZ93]) can be solved. Before we give it we need some definitions.

Definition 7.4.9 A term is *functional* if it is of the form $f(t_1, \ldots, t_n)$. A literal A is called *weakly covering* if every (sub)term of A, that is non-ground and functional, contains all variables that occur in A.

Definition 7.4.10 Let C be a set of clauses. C is in E^+ iff

1. All literals occurring in clauses of C are weakly covering, and
2. all clauses in C are decomposable.

Definition 7.4.11 The $<_v$ order is the following order on literals: $A_1 <_v A_2$ iff the maximal depth of occurrence of a variable in A_1 is strictly less then the maximal depth of occurrence of a variable in A_2.

It is proven in ([FLTZ93]) that, when resolution with factoring is applied on a set of clauses in the E^+-class, then only a finite set of clauses will be derived. However the completeness of resolution with the $<_v$-order was open.

The completeness of resolution with the $<_v$-order on the E^+-class follows from Theorem 7.4.8. So, now it is proven that resolution and factoring with the $<_v$-order is a decision procedure for clause sets that are in E^+.

7.5 Clauses consisting of two literals

In this section we prove that the condition $A\Theta \prec A$ can also be dropped when the clause set consists of clauses with length at most 2. The proof is almost the same as the completeness proof for decomposable clause sets, but a little bit more complicated.

First note that the class is closed under resolution and factorisation:

Lemma 7.5.1 Let c_1 and c_2 be clauses with $\mid c_1 \mid \leq 2$, and $\mid c_2 \mid \leq 2$. If c_1' is a factor of c_1, then $\mid c_1' \mid \leq 2$. If c' is a resolvent of c_1 and c_2, then $\mid c' \mid \leq 2$.

Definition 7.5.2 Let t be a term. The complexity of a term t, written as $\#t$ is recursively defined from:

1. $\#c = 1$, for a constant c,
2. $\#f(t_1, \ldots, t_n) = 1 + \#t_1 + \cdots + \#t_n$, for a compound term.

Let $c = \{a{:}\, A, b{:}\, B\}$ be a representation-indexed clause, and let $\{V_1, \ldots, V_n\} = \mathrm{Var}(A) \cap \mathrm{Var}(B)$. Let Ξ be a substitution, such that

$$A\Xi = a, \text{ and } B\Xi = b.$$

The *connection height* of c, written as $\#c$, is defined as $\#V_1\Xi + \cdots + \#V_n\Xi$.

The reason that we are interested in the connection height is that it decreases through the deduction. Before we prove this we prove two technical lemmata:

Lemma 7.5.3 Let $c = \{a{:}\, A, b{:}\, B\}$ be a representation-indexed clause. Let Θ be the most general unifier of A with a literal C, for which $\mathrm{Var}(B) \cap \mathrm{Var}(C) = \emptyset$, and which has a as instance. Then

$$\#\{a{:}\, A\Theta, b{:}\, B\Theta\} \leq \#\{a{:}\, A, b{:}\, B\}.$$

If $B\Theta \neq B$, then strictly

$$\#\{a{:}\, A\Theta, b{:}\, B\Theta\} < \#\{a{:}\, A, b{:}\, B\}.$$

Proof
The proof is rather terrible. We use the algorithm of Theorem 3.7.11. Write $\Theta = \Theta_n = \Sigma_1 \cdot \ldots \cdot \Sigma_n$.
First we prove that Θ will not affect the variables in $\mathrm{Var}(B)\backslash\mathrm{Var}(A)$, and that for no variable $V \notin (\mathrm{Var}(B)\backslash\mathrm{Var}(A))$, the result $V\Theta$ will contain a variable from $\mathrm{Var}(B)\backslash\mathrm{Var}(A)$.

IH All variables that occur in Θ_i are present in A, or C.

IH clearly holds for $\Theta_i = \{\}$, because $\{\}$ is rather empty. If one assumes IH(i) then all variables in $A\Theta_i$ and $C\Theta_i$ are in A or C. Because of this all variables in Σ_{i+1} are in A or C, and so all variables in Θ_{i+1} are in A or C. This implies that the variables in $\mathrm{Var}(B)\backslash\mathrm{Var}(A)$ occur neither in the range, nor in the domain of Θ.
Now define $c_i = \{a{:}\, A\Theta_i, b{:}\, B\Theta_i\}$. Define Ξ_i as a substitution such that

$$A\Theta_i\Xi_i = a, \text{ and } B\Theta_i\Xi_i = b.$$

We prove that $\#c_{i+1} < \#c_i$. Let Σ_{i+1} be the $i+1$-st mesh substituent. (This notion is used in Theorem 3.7.11) Then either:

1. Σ_{i+1} is of the form $\{V := W\}$, where W is a variable. Again two cases:
   (a) Both V and W are in $\mathrm{Var}(A\Theta_i) \cap \mathrm{Var}(B\Theta_i)$. Write $\#c_i$ as $\#V_1\Xi + \cdots + \#V_l\Xi$. In that case in this sum $\#V_1\Xi + \cdots + V_l\Xi$ one of the terms will be dropped, and strictly $\#c_{i+1} < \#c_i$.
   (b) Either V or W is not in $\mathrm{Var}(A\Theta_i) \cap \mathrm{Var}(B\Theta_i)$. In that case $\#c_{i+1} = \#c_i$.
2. Σ_{i+1} is of the form $\{V := f(t_1, \ldots, t_n)\}$. Two cases:
   (a) If $V \notin (\mathrm{Var}(A\Theta_i) \cap \mathrm{Var}(B\Theta_i))$, then $\#c_{i+1} = \#c_i$.
   (b) Otherwise write $\#c_i = \#V_1\Xi + \cdots + \#V_l\Xi$. One of the V_i equals V. Assume it is V_1. $V_1\Xi$ must be of the form $f(u_1, \ldots, u_n)$. Let $W_1, \ldots, W_k$ be the variables that occur in the u_j Then $\#c_{i+1} = \#W_1 + \cdots + \#W_k + \#V_2\Xi + \cdots + V_l\Xi$. It must be the case that $\#W_1 + \cdots + \#W_k < \#u_1 + \cdots + \#u_n = \#V_1\Xi - 1$. Because of this $\#c_{i+1} < \#c_i$.

Then, finally, if $B\Theta \neq B$, there must be a Σ_i with $B\Theta_i\Sigma_{i+1} \neq B$. For this Σ_{i+1} either the 1st of the 1st case, or the 2nd of the 2nd case holds, because the other cases will not affect $B\Theta_i$. This makes $\#c\Theta < \#c$.
End of Proof

Lemma 7.5.4 Let $a{:}A, b{:}B, c{:}C$ be indexed literals.
If $(\mathrm{Var}(B) \cap \mathrm{Var}(C)) \subseteq \mathrm{Var}(A)$, then

1. $\#\{b{:}B, c{:}C\} \leq \#\{a{:}A, c{:}C\}$,
2. $\#\{b{:}B, c{:}C\} \leq \#\{a{:}A, b{:}B\}$.

Proof
It is sufficient to prove the first inequality. Let Σ be a substitution for which

$$B\Sigma = b, \text{ and } C\Sigma = c.$$

Let $\{V_1, \ldots, V_n\}$ be the variables that occur in $\mathrm{Var}(B) \cap \mathrm{Var}(C)$. Then

$$\#\{b{:}B, c{:}C\} = \#W_1\Sigma + \cdots + \#W_n\Sigma.$$

Let Θ be a substitution for which

$$A\Theta = a, \text{ and } C\Theta = c.$$

Let $\{W_1, \ldots, W_m\}$ be the variables that occur in $\mathrm{Var}(A) \cap \mathrm{Var}(C)$. Then

$$\#\{a{:}A, c{:}C\} = \#W_1\Theta + \cdots + \#W_m\Theta.$$

Now $\mathrm{Var}(B) \cap \mathrm{Var}(C) \subseteq \mathrm{Var}(A) \cap \mathrm{Var}(C)$. This makes that every V_i is a W_j. Because both V_i and W_j occur in C, it must be the case that $V_i\Sigma = W_j\Theta$, and so $\#V_i\Sigma = \#W_j\Theta$. This will make $\#V_1\Theta + \cdots + \#V_n\Theta < \#W_1\Theta + \cdots + \#W_m\Theta$.
End of Proof

Theorem 7.5.5 Let $c_1 = \{a{:}A_1, b{:}B\}$ and $c_2 = \{\neg\, a{:}\neg\, A_2, c{:}C\}$ be two representation-indexed clauses. Let $c = \{b{:}B\Theta, c{:}C\Theta\}$ be the resolvent of c_1, and c_2. Then

$$\#c \leq \mathrm{Min}\#c_1, \#c_2.$$

If either $B\Theta \neq B$, or $C\Theta \neq C$, then

$$\#c < \mathrm{Min}(\#c_1, \#c_2)).$$

Proof
Two things happen when the resolvent is constructed.

1. A most general unifier Θ of A_1 and A_2 is applied. Initially there are no overlapping variables between c_1 and c_2. As a consequence by Lemma 7.5.3,

$$\#c_1\Theta \prec \#c_1, \text{ and } \#c_2\Theta \prec \#c_2,$$

and if B is affected by Θ, then

$$\#c_1\Theta < \#c_1,$$

and if C is affected by Θ, then

$$\#c_2\Theta < \#c_2.$$

2. After that $b{:}B\Theta$ and $c{:}C\Theta$ are combined into one clause. Because $(\mathrm{Var}(B\Theta) \cap \mathrm{Var}(C\Theta)) \subseteq \mathrm{Var}(A\Theta)$, Lemma 7.5.4 can be used. From this follows Theorem 7.5.5.

End of Proof

Definition 7.5.6 Let $c = \{a_1{:}(A_1, n_1), a_2{:}(A_2, n_2)\}$ be an indexed clause. We call c representation-indexed if c satisfies the following conditions:

1. $n_1 = n_2$,
2. There is one substitution Θ, such that $a_1 = A_1\Theta$, and $a_2 = A_2\Theta$,
3. n_1 equals the connection height of $\{a_1{:}A_1, a_2{:}A_2\}$.

A clause $\{a{:}(A, n)\}$ is representation-indexed if $n = 0$. Resolution and factoring for representation-indexed clauses can be obtained by modifying the indices.

Definition 7.5.7 Let $C = \{c_1, \ldots, c_n\}$ be an unsatisfiable set of two element clauses, and

$$\Theta_{1,1}, \ldots, \Theta_{1,l_1}, \ldots, \Theta_{n,1}, \ldots, \Theta_{n,l_n}$$

be a list of substitutions such that the resulting clause set

$$c_1\Theta_{1,1}, \ldots, c_1\Theta_{1,l_1}, \ldots, c_n\Theta_{n,1}, \ldots, c_n\Theta_{n,l_n}$$

is unsatisfiable, and ground. Then $\overline{C}$ is defined as the following set of representation-indexed ground clauses: For every $c_i = \{A_1, A_2\} \in C$, and substitution $\Theta_{i,j}$,

$$\{A_1\Theta_{i,j}{:}(A_1, n), A_2\Theta_{i,j}{:}(A_2, n)\} \in \overline{C},$$

where n is the connection height of the clause $\{A_1\Theta_{i,j}{:}A_1, A_2\Theta_{i,j}{:}A_2\}$. For every $c_i = \{A\} \in C$, and substitution $\Theta_{i,j}$,

$$\{A\Theta_{i,j}{:}(A, 0)\} \in \overline{C}.$$

Lemma 7.5.8 Let C be an unsatisfiable set of clauses, that consist of at most two literals. Let $\overline{C}$ be defined as in Definition 7.5.7. Let $\prec$ be an order on predicate literals, which is extended to indexed literals as follows:

$$a{:}(A, n) \prec b{:}(B, n) \text{ iff } A \prec B.$$

Then the following holds: If $\overline{C}$ has an ordered refutation, using the extension of $\prec$, then C has an ordered refutation, using $\prec$.

We will use this to construct a resolution game, with which the following can be proven:

Theorem 7.5.9 Let C be a set of clauses, s.t. for all $c \in C, (\mid c \mid \leq 2)$. Let $\prec$ be an order, s.t.

- $A \prec B$ implies $A\Theta_1 \prec B\Theta_2$, for all renamings $A\Theta_1$ of A, and $B\Theta_2$ of B.

Proof
Let C be a set of clauses, which have a length of at most 2, that is unsatisfiable. Let $\overline{C}$ be obtained as in Definition 7.5.7. We construct the following resolution game $\mathcal{G} = (P, \mathcal{R}, \mathcal{A}, \preceq)$, with

- P is the set of ground literals that occur in $\overline{C}$.
- $\mathcal{R}$ consists of the following rules: For each ground literal A that occurs in $\overline{C}$,
$$\{A, \neg A\} \vdash \emptyset \in \mathcal{R}.$$
- $\mathcal{A}$ contains all pairs (A, n), where $\mathcal{A}$ is an in-between literal, and $0 \leq n \leq h$, in which h is the maximal connection height that occurs in $\overline{C}$.
- $\preceq$ is defined from $(A_1, n_1) \preceq (A_2, n_2)$ iff
  1. $n_1 < n_2$, or
  2. $n_1 = n_2$, and A_1 is an instance of A_2.

The rest of the proof is completely standard: It is easily seen that $\preceq$ is a well-founded, weak order, and that resolution and factoring, as defined in Definition 7.5.6 are a valid strategy of the defender.
End of Proof

7.6 Conclusions and Future Work

We now have applied the resolution game to non-liftable orders and have obtained many results on the completeness of non-liftable orders. A lot of progress has been made on this point, but it is not possible to oversee now the possible applications of resolution games. It is likely that resolution games can be applied to resolution in modal logics as well, and to ordered paramodulation.

Another important unsolved question is the following: Is it the case that resolution and factoring with every ordering $\prec$ with the property $A \prec B \Rightarrow A\Theta_1 \prec B\Theta_2$ for renamings Θ_1 and Θ_2, is complete? Until now no proof is known for this fact, but there is also no counterexample. If this is true, then it is not likely that the proof will use resolution games, because the completeness seems to depend on delicate properties of substitution. We also have strong doubt if the resulting refinements will be efficient. Nevertheless it is important to solve this question, from the theoretical point of view.

Furthermore it is necessary to do experimentation with non-liftable orders in order to see how well they behave in practice. The main problem here is that we do not know how well non-liftable orders are compatible with subsumption, and that subsumption is a very important refinement.

It is not known yet how well non-liftable orders can be applied to obtain decidable classes. It was possible to solve one open question in ([FLTZ93]), and it is well possible that with non-liftable orders more decidable classes can be obtained.

Chapter 8

Epilogue

We think that the main contribution of this thesis to the field of automated theorem proving are the resolution game and the non-liftable orderings. The resolution game as it is developed here is very general, because it is developed in the context of the transformation of Chapter 4. This means that every closed semantic tableau can be transformed into a winning strategy for the resolution game. As a consequence a resolution game is possible for every logic which has a semantic tableau calculus.
It is at this moment not completely possible to oversee the possible applications of the resolution game and the proof method that is used for its completeness. It is likely that resolution games can be applied to resolution in modal logics to model the combination of subsumption and ordered resolution in modal logics. Also do we expect that it will be possible to apply the resolution game to reasoning with equality. It may be possible to develop a combination of ordered resolution and term rewriting as in ([BG90]).
We did not do yet any experimentation with non-liftable orders. It is likely that resolution with non-liftable orders will yield an improvement of the efficiency of resolution theorem provers, but experiments are necessary.
It will also be useful to consider the possibility of new decision classes. It is possible that with non-liftable orders more decidable classes are possible.

Bibliography

[BG90] L. Bachmair, H. Ganzinger, On restrictions of ordered paramodulation with simplification, CADE 10, pp 427-441, Keiserslautern, Germany, Springer Verlag, 1990.

[Baum92] P. Baumgartner, An Ordered Theory Calculus, in LPAR92, Springer Verlag, Berlin, 1992.

[Bezem90] M. Bezem, Completeness of Resolution Revisited, Theoretical Computer Science 74, pp. 227-237, 1990.

[Boyer71] R.S. Boyer, Locking: A Restriction of Resolution, Ph. D. Thesis, University of Texas at Austin, Texas 1971.

[Catach91] L. Catach, TABLEAUX, A General Theorem Prover for Modal Logics, Journal of Automated Reasoning 7, pp. 489-510, 1991.

[ChangLee73] C-L. Chang, R. C-T. Lee, Symbolic logic and mechanical theorem proving, Academic Press, New York 1973.

[Chellas80] B. F. Chelass, Modal Logic, an Introduction, Cambridge University Press, 1980.

[Egly] On Definitional Transformations to Normal Form for Intuitionistic Logic, unpublished.

[EnjFar89] P. Enjalbert, L. Fariñas del Cerro, Modal Resolution in Clausal Form, Theoretical Computer Science 65, 1989.

[FarHerz88] L. Fariñas del Cerro and A. Herzig, Linear Modal Deductions, CADE '88, pp. 487-499, 1988.

[FLTZ93] C. Fermüller, A. Leitsch, T. Tammet, N. Zamov, Resolution Methods for the Decision Problem, Springer Verlag, 1993.

[Fitting88] M. Fitting, First-Order Modal Tableaux, Journal of Automated Reasoning 4, pp. 191-213, 1991.

[Fitting91] M. Fitting, Destructive Modal Resolution, Journal of Logic and Computation, volume 1, pp. 83-97, 1990.

[Foret92] A. Foret, Rewrite Rule Systems for Modal Propositional Logic, Journal of Logic Programming 12, pp. 281-298, 1992.

[Galllier86] Jean H. Gallier, Logic for Computer Science, (Foundations of Automatic Theorem Proving), Harper & Row, Publishers, New York, 1986.

[Girard87] Linear Logic, in Theoretical Computer Science 50, pages 1-102, 1987.

[GiTaLai89] J.-Y. Girard, P. Taylor, Y. Lafont, Proofs and Types, Cambridge University Press, Cambridge, 1989.

[Goldbl87] Robert Goldblatt, Logics of Time and Computation, CSLI Stanford/Palo Alto/Menlo Park, 1987.

[HR91] J. Hsiang and M. Rusinowitch, Proving Refutational Completeness of Theorem-Proving Strategies: The Transfinite Semantic Tree Method, Journal of the ACM, Vol. 38, no. 3, July 1991, pp. 559-587.

[HuCre68] G.E. Hughes, M.J. Cresswell, An Introduction to Modal Logic, Methuen and Co., London, New York, 1968.

[HuCre84] G.E. Hughes, M.J. Cresswell, A Companion to Modal Logic, Methuen and Co, London, New York, 1984.

[Joy76] W.H. Joyner, Resolution Strategies as Decision Procedures, J. ACM 23, 1 (July 1976), pp. 398-417.

[KH69] R. Kowalski, P.J. Hayes, Semantic trees in automated theorem proving, Machine Intelligence 4, B. Meltzer and D. Michie, Edingburgh University Press, Edingburgh, 1969.

[Kripke59] A Completeness Theorem in Modal Logic, in Journal of Symbolic Logic, pp. 1-14, Vol. 24, 1959.

[Ladner77] R.E. Ladner, The Computational Complexity of Provability in Systems of Modal Propositional Logic, SIAM Journal on Computing 6, pp 467-480, 1977.

[Leitsch88] A. Leitsch, On Some Formal Problems in Resolution Theorem Proving, Yearbook of the Kurt Gödel Society, pp. 35-52, 1988.

[Lifschitz87] V. Lifschitz, What is the Inverse Method? Journal of Automated Reasoning, Kluwer Academic Publishers, pp. 1-23, nr. 5, 1989.

[Lovelnd78] D. W. Loveland, Automated Theorem Proving, a Logical Basis, North Holland Publishing Company, Amsterdam, New York, Oxford, 1978.

[Maslov71] S. Yu. Maslov, Connection Between the Strategies of the Inverse Method and the Resolution Method, in Seminars in Mathematics, Plenum Publishers, 16, 1971.

[Maslov86] S. Yu. Maslov, Theory of Deductive Systems and its Applications, (translated by M. Gelfond and V. Lifschitz) The MIT Press, Cambridge, Massachussets, London, England, 1986.

[Mints88] G. Mints, Gentzen-Type Systems and Resolution Rules, Part 1, Propositional Logic, in COLOG-88, International Conference on Computational Logic, Talinn (at that time) USSR, 1988.

[Mints93] G. Mints, Resolution Calculus for the First Order Linear Logic, in Journal of Logic, Language and Information 2: 59-83, Kluwer Academic Publishers, the Netherlands.

[Nivelle92] H. de Nivelle, Generic modal resolution, Technical Report 92-90, Delft University of Technology, fac. TWI, 1992.

[Nivelle93] H. de Nivelle, A General Resolution Scheme, Report 93-62, Reports of the Faculty of Technical Mathematics and Computer Science, Delft University of Technology, 1993.

[Nivelle93a] H. de Nivelle, Generic Resolution in Propositional Modal Systems, in LPAR93, Springer Verlag, Berlin, 1993.

[Nivelle94] H. de Nivelle, A Unification of Ordering Refinements of Resolution in Classical Logic, in JELIA '94, York, UK, September 1994, Springer Verlag.

[Nivelle94b] H. de Nivelle, Resolution Games and Non-Liftable Resolution Orderings, Internal report 94-36, Department of Mathematics and Computer Science, Delft University of Technology, 1994.

[Nivelle94c] H. de Nivelle, Application of Resolution Games to Resolution Decision Procedures, Internal Report 94-50, Department of Mathematics and Computer Science, Delft University of Technology, 1994.

[Nivelle95] Resolution Games and Non-Liftable Resolution Orderings, in CSL 94, pp 279-293, Springer Verlag, Kazimierz, Poland, 1994,

[Ohlbach88a] H.J. Ohlbach, A Resolution Calculus for Modal Logics, PhD thesis, Universität Kaiserslautern, 1988.

[Ohlbach88b] H.J. Ohlbach, A Resolution Calculus for Modal Logics, CADE '88, pp. 500-516, 1988.

[Reynlds66] J. Reynolds, Unpublished Seminar Notes, Stanford University, Palo Alto, California, 1966.

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

[Somm92] R. Sommerhalder, A Resolution Method for Some Systems of Modal Logic, 1992.

[Stat79] R. Statman, Lower Bounds on Herbrand's Theorem, in Proceedings of the American Mathematical Society, Vol. 75, Number 1, 1979.

[Tamm93] Proof Search Strategies in Linear Logic, Report 70, Programming Methodology Group, Department of Computer Sciences, Chalmers University of Technology and University of Göteburg, 1993.

[Tamm94] T. Tammet, Seperate Orderings for Ground and Non-Ground Literals Preserve Completeness of Resolution, unpublished, 1994.

[Zam72] N.K. Zamov: On a Bound for the Complexity of Terms in the Resolution Method, Trudy Mat. Inst. Steklov 128, pp. 5-13, 1972.

Appendix A

Ordening Verfijningen van Resolutie

A.1 Inleiding

Dit proefschrift houdt zich bezig met het volgende probleem: Gegeven een verzameling formules $F_1, \ldots, F_n$ en een formule G :

- Hoe bepaal ik zo efficient mogelijk met een computer of G een logisch gevolg is van $F_1, \ldots, F_n$?

De methode die hiervoor de meest veelbelovende is, is de *resolutiemethode*. De resolutiemethode kan verder verfijnd worden: Een belangrijke groep van die verfijningen is de groep van de *ordening verfijningen*.
Voor we de resolutieregel kunnen beschrijven moeten er wat voorbereidende begrippen geïntroduceerd worden: Een *atoom* is een uitspraak van de vorm $P(X_1, \ldots, X_n)$. Voorbeelden zijn:

Socrates is een mens	mens(Socrates)
Plato is een mens	mens(Plato)
Socrates is ouder dan Plato	ouder(Socrates, Plato)
X is een mens	mens(X)

Een *literal* is een atoom, of de ontkenning van een atoom. Alle zinnen uit de bovenstaande tabel zijn atomen, en ook de uitspraken: Socrates is geen mens, Plato is geen mens etc., zijn literals. Twee literals zijn *tegengesteld* als L_1 van de vorm niet(L_2)is, of L_2 van de vorm niet(L_1) is. Twee

tegengestelde literals kunnen nooit gelijktijdig waar zijn. Voorbeelden zijn: mens(Socrates) en niet mens(Socrates).
Een *clause* is een uitspraak van de volgende vorm:

- Voor alle $X_1, \ldots, X_n$ geldt: $P_1(X_1, \ldots, X_n)$ of $\cdots$ of $P_m(X_1, \ldots, X_n)$.

Hierin zijn de P_i literals. Een *substitutie* van t voor X_i wordt verkregen door overal in de clause de variabele X_i te vervangen door t.
Bijvoorbeeld de volgende uitspraak:

- Alle mensen zijn sterfelijk

kan gelezen worden als: Voor alle X, als X een mens is dan is X sterfelijk. Dit geeft de volgende clause:

- Voor alle X, (niet mens(X) of sterfelijk(X)).

Het is mogelijk om Socrates voor X te substitueren. Dit geeft:

- niet(mens(Socrates)) of sterfelijk(Socrates).

De volgende zin is ook een clause:

- Voor alle X (griek(X) of niet griek(X))

We kunnen nu de resolutieregel definieren. Neem aan dat er twee clauses zijn:

- Voor alle $\overline{X}$ geldt: $P_1(\overline{X})$ of $\cdots$ of $P_n(\overline{X})$.
- Voor alle $\overline{Y}$ geldt: $Q_1(\overline{Y})$ of $\cdots$ of $Q_m(\overline{Y})$.

en dat het mogelijk is om voor de variabelen $\overline{X}$ en de variabelen $\overline{Y}$ een substitutie te doen, zo dat er twee tegengestelde literals ontstaan in clause 1 en clause 2. Schrijf dan de resultaten van de substitutie als:

- Voor alle $\overline{X}'$ geldt: $P_1(\overline{X}')$ of $\cdots$ of $P_n(\overline{X}')$.
- Voor alle $\overline{Y}'$ geldt: $Q_1(\overline{Y}')$ of $\cdots$ of $Q_m(\overline{Y}')$.

Als $P_1(\overline{X}')$ en $Q_1(\overline{Y}')$ de tegengestelde literals zijn, dan kan met resolutie de volgende clause worden afgeleid:

- Voor alle $\overline{X}', \overline{Y}'$ geldt: $P_2(\overline{X}')$ of $\cdots$ of $P_n(\overline{X}')$ of $Q_2(\overline{Y}')$ of $\cdots$ of $Q_m(\overline{Y}')$.

De volgende redeneerstap kan bijvoorbeeld met resolutie gemaakt worden:

Alle Grieken zijn sterfelijk.
Socrates is een Griek.

Socrates is sterfelijk.

De eerste twee uitspraken geven de volgende clauses:

1. Voor alle X (niet griek(X) of sterfelijk(X)).
2. Griek (Socrates).

Het is mogelijk om 'Socrates' te substitueren voor X in de eerste clause. Dit geeft:

1. niet griek(Socrates) of sterfelijk(Socrates).

Nu zijn de literals 'griek(Socrates)' en 'niet griek(Socrates)' tegengesteld. Er is dus resolutie mogelijk. Het resultaat is:

3. sterfelijk(Socrates).

Ook ingewikkeldere afleidingen zijn mogelijk met resolutie:

Alle Grieken zijn sterfelijk.
Alle Romeinen zijn sterfelijk.
Cicero is een Romein of een Griek.
Cicero is sterfelijk.

De vertaling van de eerste drie uitspraken is:

1. Voor alle X (niet griek (X) of sterfelijk(X).
2. Voor alle Y (niet Romein(X) of sterfelijk(X).
3. Romein(Cicero) of griek(Cicero).

De tweede en de derde clause geven, met resolutie:

4. Griek(Cicero) of sterfelijk(Cicero).

Deze geeft, met de eerste clause:

4. sterfelijk(Cicero) of sterfelijk(Cicero).

Dit kan worden vereenvoudigd tot 'sterfelijk(Cicero)'

De resolutiemethode is erg geschikt om door een computer te worden uitgevoerd, en efficient vergeleken met andere bewijszoekmethoden. Er is echter verbetering mogelijk. Dit leidt tot zogenaamde *verfijningen* van resolutie. Een verfijning van resolutie wordt verkregen door het opleggen van beperkingen aan de mogelijkheden om resolutie toe te passen.

Een *ordening verfijning* wordt verkregen door het opleggen van preferenties van de volgende vorm:

- Als er een literal L_1 en L_2 aanwezig zijn in een clause, geef dan de voorkeur aan L_1. Deze preferentie is dwingend: Wanneer L_1 in een clause voorkomt, is het gebruik van L_2 verboden.

Het is een bekend resultaat uit de literatuur dat bepaalde vormen van ordening verfijningen kunnen worden toegepast zonder dat de bewijskracht afneemt. Het is bijvoorbeeld mogelijk om altijd literals van de vorm griek( ) te prefereren over literals van de vorm romein(). In dat geval is de bovenstaande afleiding niet meer mogelijk, maar het is nog steeds mogelijk om sterfelijk(Cicero) af te leiden door de volgorde om te wisselen.

A.2 Samenvatting

Aan dit proefschrift liggen de volgende ideeën ten grondslag:

1. Resolutiecalculi kunnen verkregen worden uit semantische tableau calculi met behulp van een standaardtransformatie.

2. Bovendien kunnen met een aanpassing van deze transformatie geordende resolutiecalculi worden verkregen.

3. Het is ook mogelijk om niet deterministisch geordende resolutiecalculi te verkrijgen. Deze heten resolutiespelen.

Hoofdstuk 2 en hoofdstuk 3 zijn een inleiding. Vanaf hoofdstuk 4 wordt eigen werk behandeld. In hoofdstuk 4 wordt gekeken wat het is dat resolutie mogelijk maakt. Deze vraag is zinvol, omdat het wenselijk is om resolutie op andere logica's te kunnen toepassen. In hoofdstuk 4 wordt aangetoond dat de mogelijkheid van resolutie gerelateerd is aan een andere bewijszoekmethode, namelijk de *semantische tableau* methode. Er wordt bewezen dat elk bewijs in de semantische tableau methode kan worden omgezet in een resolutiebewijs. We vergelijken deze methode met een andere methode uit de literatuur, waarin resolutiebewijzen uit bewijzen in sequentencalculus worden verkregen. Het voornaamste verschil ligt in de richting

van het geconstrueerde resolutiebewijs. Onze methode zoekt het bewijs van de doelformule naar de axioma's, terwijl de andere methode vanuit de axioma's de doelformule probeert te bewijzen. Dit verschil is belangrijk om dat er soms oneindig veel axioma's zijn en het niet van te voren bekend is welke axioma's gebruikt gaan worden.
Vervolgens wordt in hoofdstuk 5 de transformatiemethode van hoofdstuk 4 toegepast op modale logica. Dit is een geschikte plaats om die methode toe te passen omdat er veel modale logica's zijn. Met een algemene transformatie wordt het een stuk eenvoudiger om resolutiecalculi te verkrijgen voor al die logica's. We definieren een algemene manier om de regels van een semantische tableau calculus te verkrijgen uit de frame eigenschappen die de modale logica definieren. Daarna definieren we met behulp van de semantische tableau methode de resolutiemethode.
In hoofdstuk 6 wordt onderzocht wat er gebeurt als de preferenties veranderlijk worden gemaakt. In dat geval is het resultaat een soort spel tussen twee spelers, waarbij de ene speler probeert om een bewijs te vinden, en de andere speler steeds andere preferenties mag opleggen. Dit spel heeft *resolutiespel*. Het hoofdresultaat van hoofdstuk 6 is dat de speler die het bewijs wil vinden, het bewijs toch altijd vindt. Het belang van het resolutiespel ligt erin dat het gebruikt kan worden om orderingen met een ingewikkeld gedrag te modelleren.
In hoofdstuk 7 wordt het resolutiespel toegepast op zogenaamde *niet liftbare* orderingen. Bij de ordening verfijningen die tot nu toe bekend waren was er de volgende restrictie: Als L_1 wordt geprefereerd over L_2 en L_1' en L_2' zijn verkregen uit L_1 en L_2 door dezelfde substitutie, dan moet ook L_1' over L_2' geprefereerd worden. Deze conditie was een drastische beperking van het aantal mogelijke ordening verfijningen.
Met behulp van resolutiespelen is het mogelijk om deze restrictie te laten vervallen en te vervangen door een zwakkere restrictie. We bewijzen onder een aantal verschillende condities de volledigheid van resolutie met niet-liftbare ordeningen.
Dit maakt selectievere ordeningen mogelijk. Het is de bedoeling dat deze selectievere ordeningen gaan leiden tot grotere efficientie.

Appendix B

Curriculum Vitae

Hans de Nivelle was born in Waalre in the Netherlands on June 25th, 1967. From 1979 to 1985 he went to secondary school in Lisse, from which he graduated in May 1985. (Gymnasium β.)
From 1985 to 1991 he studied computer science at the Delft University of Technology. During the study he developed interest in mathematics and took courses in algebra. The final thesis was in theoretical computer science. (title: 'Complexity of Interpretation Problems in Truth Maintenance Systems')

From 1991 to 1995 he occupied a temporary research position at the Faculty of Computer Science of the Delft University of Technology. The original aim was to develop efficient deduction methods for modal logics, but the research has shifted into a more fundamental direction: The study of efficient resolution refinements in general.