Handout Proof Methods in Computer Science

Sebastiaan A. Terwijn

Institute of Logic, Language and Computation
University of Amsterdam
Plantage Muidergracht 24
1018 TV Amsterdam
the Netherlands
terwijn@logic.at

Fall 2008

Contents

1 Preliminaries

These notes are meant to support part of the course Introduction to Logic in Computer Science. Below we treat various proof methods from computer science. In particular we treat the tableau method and resolution. Apart from

the basic definitions of propositional and predicate logic, no acquaintance with any particular proof system for these logics is required. Our lectures are loosely based on Nerode and Shore [2], which we recommend as background reading.

2 Tableau proofs in propositional logic

Tableaux are schemes to analyze composite formulas into their simpler components. They have become a standard way of presenting proofs in computer science, for example because they are a natural tool in proof search. If the search fails, as a bonus we obtain a countermodel. For readers familiar with proof theory we may note that tableaux are just proofs from Gentzen's sequent calculus (cf. Buss [1]) written upside-down.

Propositional logic models reasoning about propositional atoms that are true or false, using the propositional connectives $\neg$ (negation), $\wedge$ (and), $\vee$ (or), and $\rightarrow$ (implication). Usually the meaning (semantics) of the connectives is defined using truth tables:

p	q	$\neg p$	$p \wedge q$	$p \vee q$	$p \rightarrow q$
1	1	0	1	1	1
1	0	0	0	1	0
0	1	1	0	1	1
0	0	1	0	0	1

Given this, one can compute the truth value of any composite formula by writing out the complete truth table containing its value for all possible assignments of the propositional variables. A propositional formula is *satisfiable* if it has an assignment that makes it true, and a set of formulas is satisfiable if there is an assignment satisfying all its elements.

For a formula with n variables there are 2^n possible assignments, hence the truth table quickly becomes too big to handle. Since the Boolean satisfiability problem SAT is NP-complete, and NP is not known to be different from P, at present it is not clear whether and how exponential methods in the worst case can be avoided. But since it is clear that the truth table method is guaranteed to take exponential time, it is desirable to have methods that at least in some cases work much faster. The tableau method is one such method.

A *sequent* is an object of the form $\Phi \Rightarrow \Psi$, where Φ and Ψ are finite sets of formulas. We think of this as saying that the *conjunction* of the formulas in Φ implies the *disjunction* of the formulas in Ψ. A *tableau* is a scheme for reducing sequents to simpler ones. To start with, every sequent of the form $\Rightarrow p$ or $p \Rightarrow$, where p is a propositional variable, is an atomic tableau. For every propositional connective there are two atomic tableaux, a left version and a right version, listed in Figure 2.1. Formally a tableau is a (binary) tree obtained from a sequent by repeated applications of the atomic tableaux. Figure 2.2 contains an example of a propositional tableau.

Definition 2.1. Suppose that φ is a formula in a sequent on a path P through tableau τ. We define the following notions.

- φ is *reduced* on P if the atomic tableau for φ is applied somewhere along P.

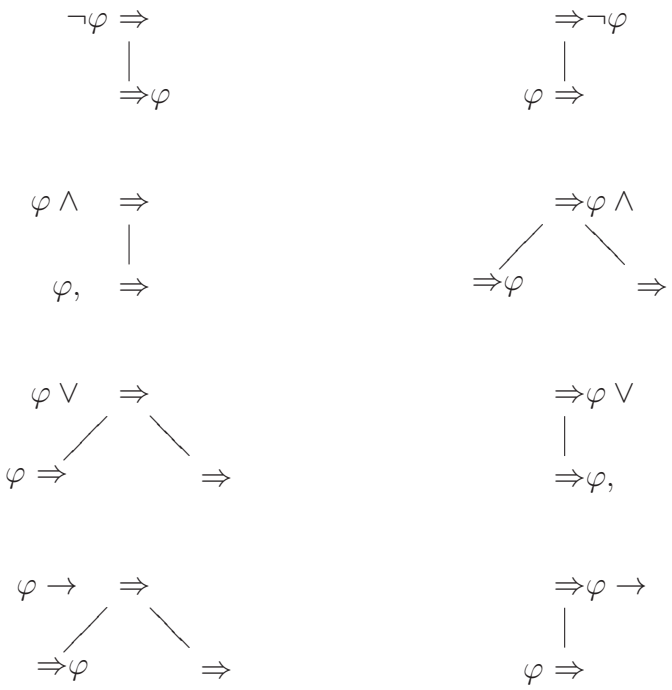


Figure 2.1: The atomic tableaux

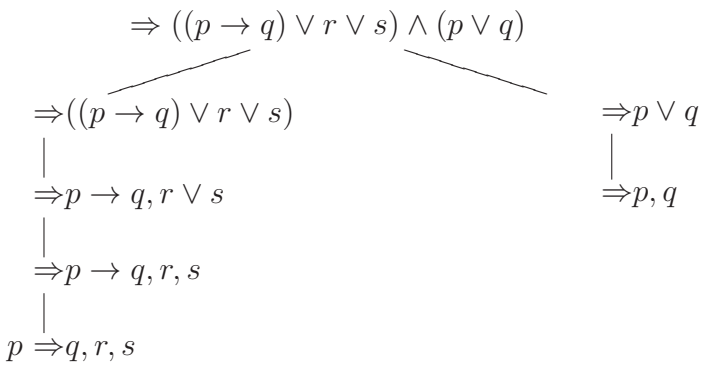
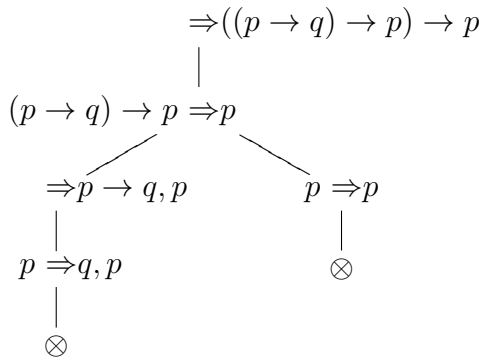


Figure 2.2: An example tableau

- P is *closed* if some formula occurs both on the left and on the right of $\Rightarrow$ on P. In the pictures below we mark the end of closed paths with $\otimes$.
- P is *finished* if it is closed or every formula on it is reduced. A finished path that is not closed is *open*.
- τ is finished if every path is finished, and closed if every path is closed.
- A *tableau proof* of a formula φ is a closed tableau with root $\Rightarrow \varphi$. In this case we say that φ is *tableau provable* and write $\vdash \varphi$.

As an example we consider the following proof of Peirce's law.



Definition 2.2. Given a sequent, the *complete systematic tableau* (CST) with this sequent as root is the tableau obtained by reducing at every step the first unreduced formula, where "first" is defined by choosing the least level at which an unreduced formula occurs and then choosing the leftmost one at that level.

We need the following basic result about trees. A tree is *finitely branching* if every node has only finitely many successors.

König's Lemma. *Every infinite tree that is finitely branching has an infinite path.*

Proof. Inductively define the infinite path as follows. Starting in the root, pick a successor such that the subtree below it is still infinite. Such a successor exists since the tree is infinite and there are only finitely many successors. For choosing the next successor, repeat the argument. □

Proposition 2.3. *Every CST is finished and finite.*

Proof. Since every level of the tableau is finite, if φ is an unreduced formula, at some point in the construction of the CST it would be the least unreduced formula and it would be reduced.

Next note that every path in the CST is finite. This is because at every step extending the path some sequent occurring on it is reduced. Since every formula can be reduced only finitely often, the path must be finite.

The CST has the form of a binary tree, and by the previous observation all its paths are finite, hence by König's Lemma it is finite. □

Let $\models \varphi$ denote that φ is a tautology, i.e. that it is valid under all assignments of its propositional variables.

Theorem 2.4. (Soundness) *For all formulas* φ, $\vdash \varphi \Longrightarrow \models \varphi$.

Proof. This follows immediately from the soundness of the atomic tableaux. Since general tableaux are built out of the atomic ones, these are sound too. □

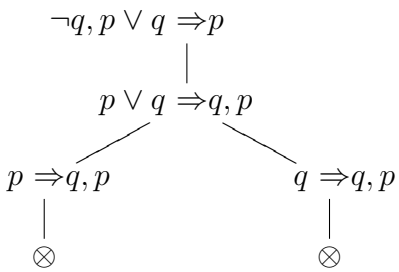
The proof of the following theorem demonstrates a feature that will be useful later on when we generalize our methods to predicate logic, namely that an open branch of a tableau yields a countermodel.

Theorem 2.5. (Completeness) *For all formulas* φ, $\models \varphi \Longrightarrow \vdash \varphi$.

Proof. Suppose that $\nvdash \varphi$. Then the CST for $\Rightarrow \varphi$ has an open branch P. Define an assignment V of the propositional variables by $V(p) = 1$ if and only if p occurs on the left of $\Rightarrow$ on P. By formula induction it follows that V agrees with all entries on P, i.e. makes all formulas on the left true and all those on the right false. Since φ occurs on the right, V falsifies φ. □

Given any (possibly infinite) set of formulas Σ we can define the notion of *tableau from* Σ as before, but now we consider every sequent $\varphi \Rightarrow$ with $\varphi \in \Sigma$ as an atomic tableau. This means that in the construction of a tableau we may add formulas from Σ on the left at any point. We write $\Sigma \vdash \varphi$ if there is a closed tableau from Σ with root $\Rightarrow \varphi$.

As an example, the following tableau shows that $\neg q, p \vee q \vdash p$.



Given a set of premises Σ, we can define in the obvious way all the analogues of the notions we had before. In the definition of the complete systematic tableau (CST) from Σ, we now have to make sure that all formulas of Σ appear on any open path. In particular, the CST is not always finite anymore, but it is always finished. Also, if a tableau is closed, i.e. all paths are closed, then by König's Lemma is is finite. Hence tableau proofs are still finite. Soundness and completeness follow as before.

Theorem 2.6. (Compactness) *If* $\Sigma \models \varphi$ *then there is a finite subset* $\Sigma' \subseteq \Sigma$ *such that* $\Sigma' \models \varphi$.

Proof. By completeness we have $\Sigma \models \varphi$ if and only if $\Sigma \vdash \varphi$, so the theorem follows from the fact that tableau proofs are finite. □

3 Resolution in propositional logic

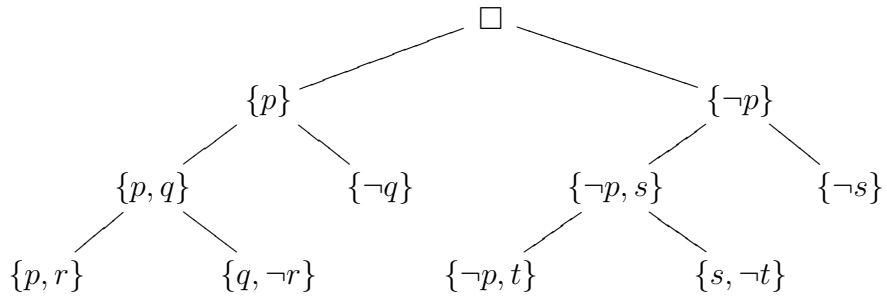
Like the tableau method, resolution is a method for refuting formulas. In resolution there is only one rule, which speeds up the search for proofs, but the method only works for formulas in CNF. (Recall that we have seen in the complexity part of the course that converting formulas to CNF may be expensive. Recall also that the CNF fragment of SAT is still NP-complete.) Resolution is the method underlying Prolog (cf. Section 5).

We use the following terminology. A *literal* is a propositional variable or the negation thereof. For a literal x its negation is denoted by $\bar{x}$. A *clause* is a finite disjunction of literals. Clauses are often denoted using set notation, so the set of literals $\{x_1, \ldots, x_k\}$ denotes the clause $x_1 \vee \ldots \vee x_k$. $\square$ denotes the empty clause, and stands for false. A formula is in *conjunctive normal form* (CNF) if it is a conjunction of clauses. Since in the context of resolution we work in the CNF-fragment, we identify formulas with (not necessarily finite) sets of clauses. Note that the empty set stands for true.

Definition 3.1. If $C_1 = \{l\} \sqcup C_1'$ and $C_2 = \{\bar{l}\} \sqcup C_2'$ are clauses, where $\sqcup$ denotes that we are taking a union of disjoint sets, then $C_1' \cup C_2'$ is called a *resolvent* of C_1 and C_2.

Note that resolution is sound, i.e. preserves satisfiability. That is, if both of the parent clauses are satisfiable then also their resolvent is satisfiable. A resolution *proof* of a clause C from a formula S is a finite sequence $C_1, C_2, \ldots, C_n = C$ of clauses such that each C_i is an element of S or a resolvent of clauses earlier in the sequence. If such a proof exists we write $S \vdash_{\mathcal{R}} C$. If $S \vdash_{\mathcal{R}} \square$ we say that S is *refutable*.

We can picture resolution proofs as binary trees. For example, the following is a refutation proof from the set $S = \big\{\{p, r\}, \{q, \neg r\}, \{\neg q\}, \{\neg p, t\}, \{\neg s\}, \{s, \neg t\}\big\}$.



Definition 3.2. $\mathcal{R}(S)$ is the *closure* of S under resolution, i.e. $S \subseteq \mathcal{R}(S)$ and if C_1, $C_2 \in \mathcal{R}(S)$ and C is a resolvent of C_1 and C_2 then also $C \in \mathcal{R}(S)$.

Theorem 3.3. (Soundness of propositional resolution) $S \vdash_{\mathcal{R}} \square \implies S$ *unsatisfiable.*

Proof. Obviously, if V is an assignment satisfying C_1 and C_2, then also $V \models C$ for any resolvent C of C_1 and C_2. Hence if S is satisfiable then $S \nvdash_{\mathcal{R}} \square$. $\square$

The following notion will be useful here and later on. Define

$$S^l = \big\{C - \{\bar{l}\} : C \in S \wedge l \notin C\big\}.$$

Motivation: When analyzing S by cases, S^l corresponds to the assumption that l is true. In this case, if we are trying to refute S, we can delete all C with $l \in C$ and delete $\{\bar{l}\}$ from the remaining clauses.

Lemma 3.4. (i) *S is satisfiable if and only if either S^l or $S^{\bar{l}}$ is satisfiable.*

(ii) *S is unsatisfiable if and only if both S^l and $S^{\bar{l}}$ are unsatisfiable.*

Proof. This follows from the "meaning" of S^l: S is satisfiable if and only if S is satisfiable with $l = 1$ or S is satisfiable with $l = 0$ if and only if S^l is satisfiable or $S^{\bar{l}}$ is satisfiable. Item (ii) follows from this by taking the contrapositive. □

Corollary 3.5. UNSAT $= \big\{S : S$ *unsatisfiable*$\big\}$ *can be inductively defined by*

(i) $\square \in S \Longrightarrow S \in$ UNSAT,

(ii) S^l, $S^{\bar{l}} \in$ UNSAT $\Longrightarrow S \in$ UNSAT.

Note that resolution is not complete in the sense that for every set of clauses S and every clause C, whenever $S \models C$ then $S \vdash_{\mathcal{R}} C$. For example, $\models p \vee \neg p$ but $\nvdash_{\mathcal{R}} \{p, \neg p\}$. However, resolution is complete in the sense that any inconsistent S is refutable. Sometimes this fact is called *refutation completeness*.

Theorem 3.6. (Completeness of propositional resolution) *S unsatisfiable $\Longrightarrow$ $S \vdash_{\mathcal{R}} \square$.*

Proof. This can be proven using compactness, but it is more useful for later applications to prove it directly, with a proof that we can reuse later. The proof is by induction on the number of applications of Corollary 3.5 (ii). Suppose that T_0 and T_1 are proofs witnessing $S^l \vdash_{\mathcal{R}} \square$ and $S^{\bar{l}} \vdash_{\mathcal{R}} \square$, respectively. If every leaf of T_0 is a clause in S then T_0 is already a proof of $S \vdash_{\mathcal{R}} \square$. Otherwise change every node C in T_0 above[1] a leaf not in S to $C \cup \{\bar{l}\}$. Note that every $C \in S^l$ is either in S or of the form $C - \{\bar{l}\}$ with $C \in S$. Thus we obtain a proof T_0' of $S \vdash_{\mathcal{R}} \{\bar{l}\}$. Similarly, if T_1 is not already a proof that $S \vdash_{\mathcal{R}} \square$, we obtain a proof T_1' of $S \vdash_{\mathcal{R}} \{l\}$ by suitably adding $\{l\}$ at some nodes. Putting T_0' and T_1' together, we obtain a proof of $\square$. □

4 Refinements of resolution

We introduce various restricted versions of resolution that try to further reduce the search space (and hence increase the speed of proof search). These variants are often useful in practise, even though we know that SAT-CNF is NP-complete. Of course, all restricted versions are still *sound*.

[1] Here our proof trees grow downwards.

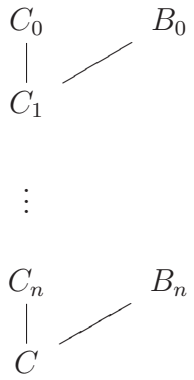


Figure 4.1: A linear resolution deduction

- *T-resolution*: Here neither of the parent clauses is a tautology. The idea is that tautologies cannot really help to produce a contradiction. Note that checking whether a clause is a tautology is easy, namely check if it contains both a literal p and $\bar{p}$. T-resolution is still *complete*: Use the same proof as above for Theorem 3.6. If T_0 and T_1 there have no tautologies then also T_0' and T_1' do not.

- $\mathcal{A}$*-resolution*, also called *semantic resolution*: For a fixed assignment $\mathcal{A}$, require that at least one of the parents is false under $\mathcal{A}$. The idea is that if both parent clauses are true under $\mathcal{A}$ then they are *consistent*, which cannot help in producing a contradiction. The closure of S under $\mathcal{A}$-resolution is denoted by $\mathcal{R}^{\mathcal{A}}(S)$. To see the completeness of $\mathcal{A}$-resolution, again use the proof of Theorem 3.6, modifying T_0 and T_1, which is now a bit more work.

- *Ordered resolution*: Given an ordering of the propositional variables, only allow resolutions $C_1 \sqcup \{p\}$, $C_2 \sqcup \{\bar{p}\}$ when p is larger than all elements in C_1 and C_2. For completeness we can again reuse the previous proof.

- *Linear resolution*: A linear resolution deduction of C from S is a sequence $(C_0, B_0), \ldots, (C_n, B_n)$ so that for $C = C_{n+1}$

  - C_0 and each B_i are either elements of S or some C_j for some $j < i$,
  - each C_{i+1}, $i \leqslant n$, is a resolvent of C_i and B_i.

  Notation: $S \vdash_{\mathcal{L}} C$. Figure 4.1 depicts a linear deduction. The C_i are called the *center clauses* and the B_i the *side clauses*. Linear resolution is still complete. We defer the proof until Section 11. Linear resolution is also the proof method underlying Prolog.

5 Prolog

Prolog is a declarative programming language built on (linear) resolution. It has its own terminology and notation, that we now discuss. Consider a clause

C with positive literals A_i and negative ones $\neg B_j$:

$$C = A_1 \vee \ldots \vee A_m \vee \neg B_1 \vee \ldots \vee \neg B_n.$$

The terminology for C for the various possible values of m and n is as follows.

If $m \leqslant 1$ C is a *Horn clause*.

$m = 1$: C is a *program clause*.

$n > 0$: C is a *rule* A :- $B_1 \ldots B_n$. A is the *head* and the B_i are the *body* or *subgoals*.

$n = 0$: C is a *fact* A :-

$m = 0$: C is a *goal clause* :- $B_1 \ldots B_n$.

A *Prolog program* P is a set of rules and facts. Given $B_1 \ldots B_n$, $P \models B_1 \wedge \ldots \wedge B_n$ if and only if $P \cup \{\{\neg B_1, \ldots, \neg B_n\}\} \models \square$. Entering ?- $B_1 \ldots B_n$ in Prolog results in adding the goal clause $\{\neg B_1 \ldots \neg B_n\}$ to the program P, after which Prolog searches for a proof of $\square$. Hence this corresponds to trying to prove $B_1 \wedge \ldots \wedge B_n$ from P.[2]

Lemma 5.1. *If a set of Horn clauses S is unsatisfiable then it contains at least one fact and one goal clause.*

Proof. This is beause the assignment making all variables true satisfies every program clause (rules and facts), and the assignment making all variables false satisfies every rule and every goal clause. $\square$

As mentioned in Section 4, linear resolution is complete (Theorem 11.3). Here we prove a weaker result, namely completeness for the Horn fragment.

Theorem 5.2. (Completeness of linear resolution for Horn clauses) *If S is unsatisfiable and Horn then $S \vdash_{\mathcal{L}} \square$.*

Proof. As noted in Lemma 5.1, S contains at least one fact $\{p\}$. Then also S^p is unsatisfiable: If $\mathcal{A} \models S^p$ then $\mathcal{A} \cup \{p\} \models S$ by definition of S^p. Since S^p contains fewer literals, by induction it follows that $S^p \vdash_{\mathcal{L}} \square$. As in the proof of Theorem 3.6, either this is already a proof of $S \vdash_{\mathcal{L}} \square$ or by adding $\bar{p}$ to every clause above a clause not in S we obtain $S \vdash_{\mathcal{L}} \bar{p}$, and by resolving on p we get $S \vdash_{\mathcal{L}} \square$. $\square$

LI-resolution. Let P be a set of program clauses and G a goal. A *linear input resolution refutation* of $S = P \cup \{G\}$ is a linear refutation that starts with G and in which all side formulas are from P (the *input clauses*). In general, LI-resolution is *not* complete: Consider the set $S = \big\{\{p, q\}, \{p, \neg q\}, \{\neg p, q\}, \{\neg p, \neg q\}\big\}$ where $\{\neg p, \neg q\}$ is the goal clause. In any linear derivation from S, as soon as we get to a center clause with one literal, any resolution produces another such clause, hence we cannot derive $\square$.

[2]The term "goal clause" for $\{\neg B_1 \ldots \neg B_n\}$ is perhaps a misnomer, since it actually corresponds to the negation of the goal, which is to prove $B_1 \wedge \ldots \wedge B_n$.

6 Tableau proofs in predicate logic

We assume the reader is familiar with the language and classical semantics of predicate logic.

In the following we work with a countable first-order language $\mathcal{L}$, that is, we assume that $\mathcal{L}$ has at most countably many nonlogical symbols (predicates, functions, and constants). Although the restriction to countable languages is not essential, it makes the presentation much more transparent. Given a first-order language $\mathcal{L}$, consider the terms built from variables and constants by applying function symbols in $\mathcal{L}$. *Ground terms* are terms without variables. To ensure that there are enough terms, expand $\mathcal{L}$ to $\mathcal{L}_C$ by adding a set of fresh constants $c_0, c_1, \ldots$. The *atomic tableaux* are defined as in Section 2, with the following extra cases for the quantifiers:

$$\begin{array}{lll} & \forall x\varphi(x) \Rightarrow & \qquad \Rightarrow\forall x\varphi(x) \\ & \quad | & \qquad\quad | \\ t \text{ any ground term of } \mathcal{L}_C & \quad \varphi(t) \Rightarrow & \qquad \Rightarrow\varphi(c) \qquad c \text{ fresh constant} \end{array}$$

$$\begin{array}{lll} & \exists x\varphi(x) \Rightarrow & \qquad \Rightarrow\exists x\varphi(x) \\ & \quad | & \qquad\quad | \\ c \text{ fresh constant} & \quad \varphi(c) \Rightarrow & \qquad \Rightarrow\varphi(t) \qquad t \text{ any ground term of } \mathcal{L}_C \end{array}$$

Note that in applying these rules, fresh constants are always available since we added the set of constants C.

Now tableau proofs, possibly with premises, are defined as before. As an example we prove $\forall x\varphi(x) \to \exists x\varphi(x)$.

$$\begin{array}{c} \Rightarrow\forall x\varphi(x) \to \exists x\varphi(x) \\ | \\ \forall x\varphi(x) \Rightarrow\exists x\varphi(x) \\ | \\ \forall x\varphi(x) \Rightarrow\varphi(c) \\ | \\ \varphi(c) \Rightarrow\varphi(c) \\ | \\ \otimes \end{array}$$

In the example shown in Figure 6.1, note that we do not necessarily repeat formulas along a path. Figure 6.2 contains an example of a false proof.

Let $t_1, t_2, \ldots$ be a list of all ground terms of $\mathcal{L}_C$. Since there are infinitely many terms to instantiate formulas, we cannot guarantee anymore that tableaux are

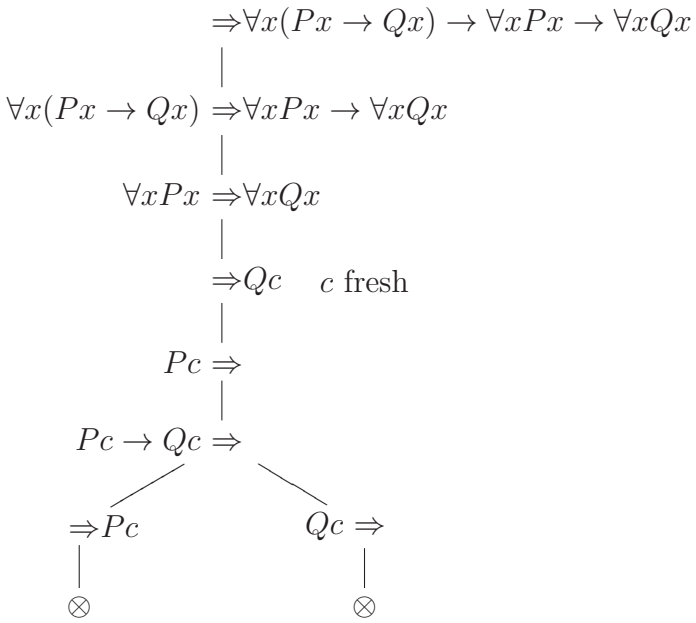


Figure 6.1: Example of a tableau proof

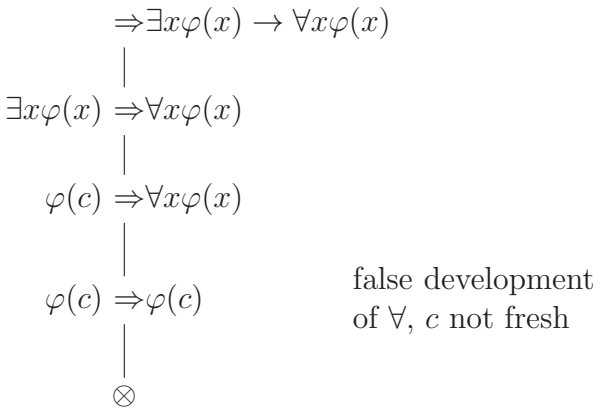


Figure 6.2: Example of a false proof

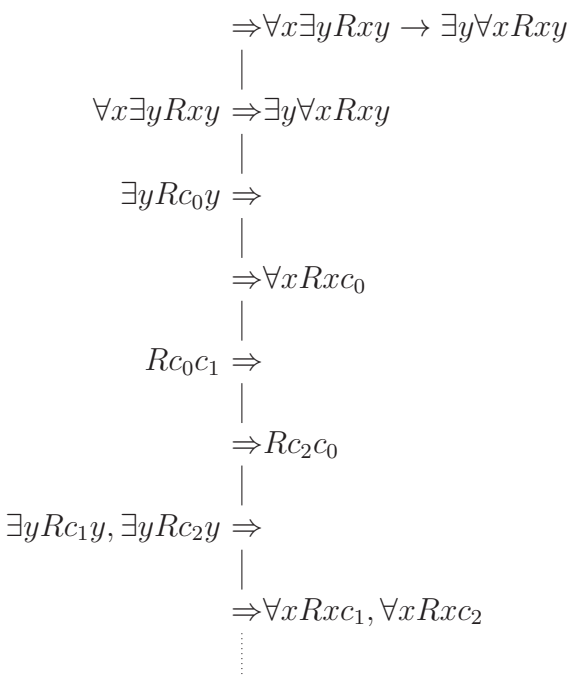


Figure 6.3: Example of an open tableau

finite. In the construction of the complete systematic tableau CST, we now make sure that for formulas $\varphi = \forall x \ \ (x)$ occurring on the left, x is instantiated with every possible ground term t_i. Similarly for formulas $\varphi = \exists x \ \ (x)$ occurring on the right. The CST is uniquely specified by always using the smallest ground term t_i, and for $\exists$ on the left and $\forall$ on the right the smallest fresh constant c_i. A (possibly infinite) path through a tableau is now called *finished* if it is either closed (i.e. some formula occurs both on the left and the right) or all such instantiations occur along it. A finished path that is not closed is *open*. A tableau is finished or closed if all its paths are. Provability using tableaux is again denoted by $\vdash$. That is, $S \vdash \varphi$ if there is a closed tableau from S with root $\Rightarrow \varphi$.

As an example of an open tableau, consider an initial part of a tableau for the formula $\forall x \exists y Rxy \to \exists y \forall x Rxy$ depicted in Figure 6.3. The right hand side provides us with ever more examples of y's with Rxy, and the left hand side ensures that none of the y's that are introduced is a uniform witness. By Theorem 6.3 the open branch provides us with an infinite countermodel for the formula. Note however that in this case there is also a finite countermodel with only two elements.

Proposition 6.1. *Every CST is finished (but not necessarily finite!).*

Proof. As before. □

Theorem 6.2. (Soundness) *For all formulas φ, $\vdash \varphi \Longrightarrow \models \varphi$.*

Proof. Again this follows from the soundness of the atomic tableaux. □

For completeness, again we use an open branch in the CST to build a model. The idea is crucial for later applications, including Herbrand's theorem.

Theorem 6.3. *Suppose that P is an open path in CST τ from S with root $\Rightarrow \varphi$. Then there is a model $\mathcal{A}$ in which φ is false and S is true.*

Proof. Build $\mathcal{A}$ from the ground terms t_i (in the language $\mathcal{L}_C$). Define

$$f^{\mathcal{A}}(t_{i_1}, \ldots, t_{i_n}) = f(t_{i_1}, \ldots, t_{i_n})$$

and let $R^{\mathcal{A}}(t_{i_1}, \ldots, t_{i_n})$ hold if and only if $R(t_{i_1}, \ldots, t_{i_n})$ occurs on the left side of P. Now prove by formula induction that a formula occurs on the right of P if and only if $\mathcal{A} \not\models$, and occurs on the left of P if and only if $\mathcal{A} \models$. □

Theorem 6.4. (Completeness) *For all S, φ, either*

(i) *the CST from S with root $\Rightarrow \varphi$ is a tableau proof of φ from S, or*

(ii) *there is an open path that gives a model of S and $\neg\varphi$.*

Proof. If every path in the CST τ is closed then all paths are finite, hence by König's Lemma τ is finite. If there is an open path in τ then Theorem 6.3 gives us a model. □

Theorem 6.5. (Löwenheim-Skolem) *If S is countable and satisfiable then it has a countable model.*

Proof. This follows since the path in Theorem 6.4 (ii) is countable. □

As in Theorem 2.6, compactness follows from completeness.

In propositional logic, every CST is finite, hence we have a decision procedure for validity, albeit a bad one in terms of complexity. For predicate logic, when we search for a proof of φ we will find one if φ is valid, but if this is not the case the CST may be infinite so we may never find out whether φ is valid. This is unavoidable, since by Church's Theorem predicate logic is undecidable. A proof of this result belongs to a course in computability theory.

7 Skolemization

In our quest to reduce predicate logic to propositional logic, we will try to eliminate quantifiers by introducing new function symbols. The idea is that the formulas $\forall x \exists y R(x, y)$ and $\forall x R(x, f(x))$ are *equisatisfiable*, that is, one of them is satisfiable if and only if the other is too. (In general the two formulas are not equivalent, cf. Exercise 7.3.) For ease of presentation, we assume that all formulas are in *prenex normal form*, but we remark that this assumption, although it is made in most presentations on this subject, is not strictly necessary. In fact, we have seen before that putting a formula in prenex form may be costly, so that it is good to know that we can work with general formulas. To put a formula in prenex form, we can move all quantifiers in front by repeatedly using the elementary equivalences $\neg\forall \equiv \exists\neg$ and $\neg\exists \equiv \forall\neg$,

$$\forall x \varphi \vee \quad \equiv \forall z(\varphi(x/z) \vee \quad),$$

etcetera.

Theorem 7.1. *For every sentence φ in a given first-order language $\mathcal{L}$ there is a universal sentence φ' in an expanded language $\mathcal{L}'$ obtained by adding new function symbols such that φ and φ' are equisatisfiable.*

Proof. Without loss of generality φ is in prenex normal form. It suffices to show that $\forall\vec{x}\exists y \ \ (\vec{x}, y)$ is equisatisfiable with $\forall\vec{x} \ \ (\vec{x}, f(\vec{x}))$. This is obvious. □

For example, $\exists x\forall y\exists z\forall u\exists v R(x, y, z, u, v)$ is equisatisfiable with

$$\forall y\forall u R\big(c, y, f(y), u, g(y, u)\big).$$

Here c is a 0-place function symbol, i.e. a constant. We call formulas obtained in this way *Skolemized*.

Corollary 7.2. *For any set S of sentences in $\mathcal{L}$, there is a formula (i.e. a set T of clauses) in an expansion $\mathcal{L}'$ such that S and T are equisatisfiable.*

Exercise 7.3. Show that in Theorem 7.1 φ and φ' are not necessarily equivalent. We always have $\varphi' \to \varphi$ but not necessarily $\varphi \to \varphi'$.

8 Herbrand's theorem

In what follows we assume that the language $\mathcal{L}$ contains at least one constant symbol.

Definition 8.1. The set of ground terms of $\mathcal{L}$ is called the *Herbrand universe* of $\mathcal{L}$. A model $\mathcal{A}$ is a *Herbrand model* if its universe is the Herbrand universe and for every function symbol f and elements $t_1, \ldots, t_n$ of $\mathcal{A}$,

$$f^{\mathcal{A}}(t_1, \ldots, t_n) = f(t_1, \ldots, t_n), \tag{1}$$

and $c^{\mathcal{A}} = c$ for all constants $c \in \mathcal{L}$. Note that the left hand side of (1) denotes the interpretation $f^{\mathcal{A}}$ of f in the model $\mathcal{A}$, and the right hand side is an element of $\mathcal{A}$. That is, ground terms are purely syntactically interpreted by themselves.

By assumption, $\mathcal{L}$ always contains a constant symbol, so that its Herbrand universe is never empty. Note that the model built in the proof of the Completeness Theorem 6.4 was a Herbrand model, but in the language $\mathcal{L}_C$ with added constants C. Note further that in Definition 8.1 there are no restrictions on the *predicates* of $\mathcal{L}$, so there are many possible Herbrand models.

Theorem 8.2. (Herbrand's Theorem) *Let S be a formula, i.e. a set of clauses interpreted as a universal formula. Then either*

(i) *S has a Herbrand model, or*

(ii) *S is unsatisfiable, and there are finitely many ground instances of elements of S whose conjunction is unsatisfiable.*

Proof. Let S' be the set of all ground instances of formulas from S (in the language $\mathcal{L}$, not in the language $\mathcal{L}_C$ from the completeness theorem). Note that $S' \neq \emptyset$ because $\mathcal{L}$ always contains a constant. Consider the CST from S', again in $\mathcal{L}$, starting with the sequent $\Rightarrow \bot$, where $\bot$ denotes some false formula. Notice that in the construction of the CST on page 12 we do not need to apply the steps that add fresh constants, because S is purely universal. If there is an open path then we obtain a model of S' as in the proof of the Completeness Theorem 6.4, which is then a Herbrand model for S. If the tableau is finite and closed, then it is a proof that the finite subset of S' appearing in the tableau is unsatisfiable, and it follows that S is unsatisfiable. □

Corollary 8.3. *An existential formula $\exists\vec{x}\varphi(\vec{x})$, with φ quantifier free, is valid if and only if there are finitely many ground terms $\vec{t}_1, \ldots, \vec{t}_n$ such that*

$$\varphi(\vec{t}_1) \vee \ldots \vee \varphi(\vec{t}_n)$$

is valid.

Proof. $\exists\vec{x}\varphi(\vec{x})$ is valid if and only if $\forall\vec{x}\neg\varphi(\vec{x})$ is unsatisfiable. By Theorem 8.2 there are ground terms $\vec{t}_i$ such that $\bigwedge_i \neg\varphi(\vec{t}_i)$ is unsatisfiable, hence $\bigvee_i \varphi(\vec{t}_i)$ is valid. □

Exercise 8.4. Suppose that the language $\mathcal{L}$ does not contain any function symbols, and suppose that φ is a purely universal satisfiable formula (i.e. φ does not contain any existential quantifiers). Show that the proof of Theorem 8.2 can be used to obtain a finite model of φ. (Hint: Without loss of generality the language $\mathcal{L}$ only contains one constant plus the constants occurring in φ.)

Exercise 8.5. Give an example to show that the result of Exercise 8.4 no longer holds in the presence of function symbols. If you like you can use equality. If you have an example using equality, show how you can get rid of equality by substituting a suitably axiomatized binary relation R for it. (This exercise reveals some of the power of function symbols. That they have this power should not come as a surprise, since we have seen that they can be used to replace existential quantifiers.)

9 Unification

In principle, we could do resolution for predicate logic with all possible ground terms, as in the proof of Herbrand's Theorem 8.2. Needless to say, this is not efficient. Instead we use unification, which is a procedure for making terms equal by applying appropriate substitutions.[3]

Definition 9.1. A *substitution* θ is a finite set $\{x_1/t_1, \ldots, x_n/t_n\}$, where the x_i are variables and the t_i are expressions such as terms or atomic formulas. θ is *ground* if all the terms t_i are. For an expression E, or a set of expressions S, we write $E\theta$ (or $S\theta$) for the result of applying the substitution θ.

[3]In unification, term equations $t = s$ are solved by applying the same substitution θ on both sides to obtain equal terms $t\theta$ and $s\theta$. If the substitution is applied to only one of the sides the procedure is called *matching*. Both procedures are of fundamental importance in computer science, both for first-order and higher order terms.

N.B. Substitutions are supposed to be done *simultaneously*, e.g. in

$$E\{x_1/t_1, x_2/t_2\}$$

occurrences of x_2 in t_1 are not affected by the substitution x_2/t_2.

We can also *compose* substitutions θ and σ to obtain $\theta\sigma$, which is the substitution obtained by first applying θ and then σ. For example, the substitutions

$$\theta = \{x/f(y), y/g(z), w/v\},$$
$$\sigma = \{x/a, y/b, z/f(y), v/w, u/c\}$$

give the composed substitution

$$\theta\sigma = \big\{x/f(b), y/g(f(y)), u/c, v/w, z/f(y)\big\}.$$

Note that the substitution w/w is deleted from $\theta\sigma$ since it is of no consequence.

Proposition 9.2. (i) $(E\theta)\sigma = E(\theta\sigma)$.

(ii) $(\ \theta)\sigma = \ (\theta\sigma)$.

Proof. Exercise. □

Definition 9.3. A substitution θ is a *unifier* for a set of expressions $S = \{E_1, \ldots, E_n\}$ if $E_1\theta = \ldots = E_n\theta$. θ is a *most general unifier (m.g.u.)* for S if for every unifier σ for S there is a substitution ρ such that $\theta\rho = \sigma$.

For example, consider the following sets of terms, with constants a, b, c, and variables x, y, w:

$\big\{P(x,a), P(b,c)\big\}$ is not unifiable.

$\big\{P(f(x),z), P(a,w)\big\}$ is also not unifiable.

$\big\{P(f(x),y), P(f(a),w)\big\}$ is unifiable by $\{x/a, y/w\}$, but also by $\{x/a, y/a, w/a\}$ and by $\{x/a, y/b, w/b\}$ etcetera. Of these, $\{x/a, y/w\}$ is a most general unifier.

The *unification algorithm* (J. A. Robinson, 1965) is a systematic way of searching for a m.g.u. Without giving a formal description of the algorithm, we can simply describe it by saying that we always locate the first difference from the left and try to unify at that point. It can be shown that the m.g.u. σ found in this way (if it exists) satisfies

$$\sigma\theta = \theta \tag{2}$$

for every other unifier θ. (Note that property (2) is not automatic for every m.g.u. σ. Consider for example the term x. $\sigma = \{x/z\}$ and $\theta = \{x/y\}$ are m.g.u.'s for this term, and $\sigma\{z/y\} = \theta$, but $\sigma\theta \neq \theta$.)

For example:

$S_1 = \big\{f(x, g(x)), f(h(y), g(h(z))\big\}$ has m.g.u. $\{x/h(y)\}\{y/z\} = \{x/h(z), y/z\}$.

$S_2 = \big\{f(h(x), g(x)), f(g(x), h(x))\big\}$ is not unifiable.

10 Resolution in predicate logic

In the context of predicate logic, a *literal* is any atomic formula (possibly with free variables) or a negation thereof. Again, a *clause* is a finite set of literals. The clause $C = \{Q(x,y), R(y)\}$ is interpreted as $\forall x \forall y\big(Q(x,y) \vee R(y)\big)$. A formula S (i.e. any set of clauses) is interpreted as the conjunction of its clauses. Clauses are universally quantified *separately*. If necessary we *rename* variables from the various clauses. This is called "standardizing the variables apart". E.g. $S = \big\{\{P(x)\}, \{Q(x)\}\big\}$ corresponds to $\forall x P(x) \wedge \forall z Q(z)$.

Definition 10.1. Suppose the clauses

$$C_1 = C_1' \sqcup \big\{P\vec{t}_1, \ldots, P\vec{t}_n\big\}$$
$$C_2 = C_2' \sqcup \big\{\neg P\vec{s}_1, \ldots, \neg P\vec{s}_m\big\}.$$

have no variables in common. Here the $\vec{t}_i$ and $\vec{s}_j$ are sequences of terms. If σ is a m.g.u. for $\big\{P\vec{t}_1, \ldots, P\vec{t}_n, P\vec{s}_1, \ldots, P\vec{s}_m\big\}$ then $C_1'\sigma \cup C_2'\sigma$ is called a *resolvent* of C_1 and C_2.

Resolution proofs of a clause C from a set of clauses S and resolution refutations of S are defined in the same way as before in propositional logic. Again we write $S \vdash_{\mathcal{R}} C$ if there is a resolution proof of C from S, and $\mathcal{R}(S)$ denotes the closure of S under resolution.

Remark (i). Renaming of variables is necessary: $\big\{\{P(x)\}, \{\neg P(f(x))\}\big\}$ is unsatisfiable and resolution refutable, but the clauses cannot be unified without renaming x. (Note that the substitution $x/f(x)$ is allowed but yields $\big\{\{P(f(x))\}, \{\neg P(f(f(x)))\}\big\}$ because by definition we have to apply the substitution to both sides.)

Remark (ii). In general we cannot assume that $n = m = 1$, as before in Definition 3.1. Consider for example $S = \big\{\{P(x), P(y)\}, \{\neg P(x), \neg P(y)\}\big\}$. S is unsatisfiable, but if we eliminate only one literal we end up with $\big\{\{P(y), \neg P(y)\}\big\}$ which is even *valid*.

Example 10.2. Consider

(a) $\forall x, y, z\big(Pxy \wedge Pyz \to Pxz\big)$

(b) $\forall x, y\big(Pxy \to Pyx\big)$

(c) $\forall x, y, z\big(Pxy \wedge Pzy \to Pxz\big)$

We prove that (a),(b)$\vdash$(c). First we put the formulas in clausal form, while at the same time standardizing the variables apart, to obtain

$C_1 = \big\{\neg Pxy, \neg Pyz, Pxz\big\}$

$C_2 = \big\{\neg Puv, Pvu\big\}$

$C_3 = \big\{\neg Pxy, \neg Pzy, Pxz\big\}$

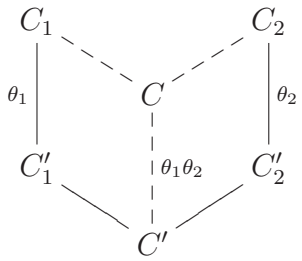
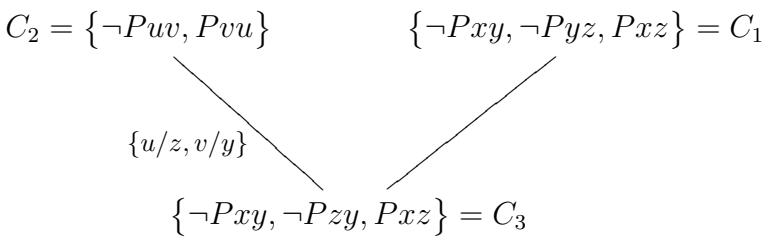


Figure 10.1: Lifting a single resolution

Now we can resolve as follows:



□

Correct answer substitutions in Prolog. When using Prolog for predicate logic, it is customary to denote predicates by small and variables by capital letters. In Prolog, the query ?- $p(X), q(Y)$ is interpreted as $\exists X \exists Y(p(X) \wedge q(Y))$. Prolog adds $\{\neg p(X), \neg q(Y)\}$ and tries to deduce $\Box$, and if successful returns a correct answer substitution, i.e. a substitution $\{X/s, Y/t\}$ with terms s and t such that $P \models p(s) \wedge q(t)$. In general, for a program P and goal $G = \{\neg A_1, \ldots, \neg A_n\}$, θ is a *correct answer substitution* if $P \models (A_1 \wedge \ldots \wedge A_n)\theta$. N.B. If $P \cup \{G\}$ is unsatisfiable then by [2, Exercise 10.5] (which is an application of Herbrand's Theorem) there is a correct answer substitution that is *ground*. That one can always find such a substitution with *resolution* follows from the completeness of resolution below.

Theorem 10.3. (Soundness of resolution) $\Box \in \mathcal{R}(S) \Longrightarrow S$ *unsatisfiable.*

Proof. Again this is obvious, since the resolution rule is sound. □

We now prove the completeness of resolution. The proof shows that in a sense we have succeeded in reducing predicate logic to propositional logic. Using Skolemization we got rid of existential quantifiers, and Herbrand's Theorem enables us to get down to the level of ground instances. We then derive completeness from the completeness of resolution for propositional logic by "lifting" resolution proofs back from the level of ground instances.

Lemma 10.4. *If $C_1' = C_1\theta_1$ and $C_2' = C_2\theta_2$ are ground instances of the clauses C_1 and C_2 that have no common variables, and C' is a resolvent of C_1' and C_2', then there is a resolvent C of C_1 and C_2 such that $C' = C\theta_1\theta_2$, cf. Figure 10.1.*

Proof. Suppose that C_1' and C_2' resolve on $P(\vec{t})$, that is, there are clauses

$$A_1 = \{P(\vec{s}_{1,1}), \ldots, P(\vec{s}_{1,i})\} \subseteq C_1,$$
$$A_2 = \{\neg P(\vec{s}_{2,1}), \ldots, \neg P(\vec{s}_{2,j})\} \subseteq C_2$$

that become unified to $\{P(\vec{t})\}$ and $\{\neg P(\vec{t})\}$ respectively by θ_1 and θ_2. Since by assumption the substitutions are disjoint, $\theta_1\theta_2$ unifies both. Hence, $C = \big((C_1 - A_1) \cup (C_2 - A_2)\big)\sigma$ is a resolvent of C_1 and C_2, where σ denotes a m.g.u. for A_1 and A_2, and

$$\begin{aligned} C\theta_1\theta_2 &= \big((C_1 - A_1) \cup (C_2 - A_2)\big)\sigma\theta_1\theta_2 \\ &= \big((C_1 - A_1) \cup (C_2 - A_2)\big)\theta_1\theta_2 \quad \text{by property (2) on page 16} \\ &= (C_1 - A_1)\theta_1 \cup (C_2 - A_2)\theta_2 \quad \text{by disjointness} \\ &= C'. \end{aligned}$$

□

Lemma 10.5. (Lifting lemma) *Let S be an $\mathcal{L}$-formula. Let S' be the set of all ground instances of S in the Herbrand universe of $\mathcal{L}$. If T' is a resolution proof $S' \vdash_{\mathcal{R}} C'$ of a clause C' from S', then there is a clause C and a resolution proof T for $S \vdash_{\mathcal{R}} C$, and a substitution θ with $T' = T\theta$.*

Proof. This follows by induction on the depth of proofs, using Lemma 10.4 at the induction step. Note that several renamings of the same clause C may occur in T, so that θ can give several ground instances of C in T'. □

Theorem 10.6. (Completeness of resolution) *S unsatisfiable $\Longrightarrow \square \in \mathcal{R}(S)$.*

Proof. Let S' be as in Lemma 10.5. By Herbrand's Theorem 8.2, if S is unsatisfiable then so is S'. By the completeness of propositional resolution (Theorem 3.6) $\square$ is derivable from S'. By the Lifting Lemma, $\square \in \mathcal{R}(S)$. □

Exercise 10.7. Consider the following sentences:

(1) $\forall x \big(\alpha(x) \to \beta(x)\big) \to \exists y \big(\gamma(y) \wedge \neg\beta(y)\big)$

(2) $\forall x \big(\gamma(x) \to \beta(x)\big)$

(3) $\exists x \big(\alpha(x) \wedge \neg\beta(x)\big)$

(a) Use Skolemization to put the above sentences in clausal normal form.
(b) Use resolution to prove that (1), (2) $\models$ (3). That is, add the negation of (3) to (1) and (2) and derive $\square$.

11 Linear resolution

We already discussed linear resolution proofs in Section 4, but we only proved completeness for the Horn fragment (Theorem 5.2). In this section we prove the full completeness of linear resolution by refining the previous proofs.

Definition 11.1. A subset $U \subseteq S$ is called a *set of support* for S if $S - U$ is satisfiable. (Namely, if S is unsatisfiable, U supports this fact.) A linear resolution proof has *support* U if it starts with $C_0 \in U$. An unsatisfiable set S is *minimally unsatisfiable* if every proper subset of it is satisfiable, that is, if $\{C\}$ is a set of support for every $C \in S$.

Lemma 11.2. *If S is unsatisfiable then there is a minimally unsatisfiable $S' \subseteq S$. If U is a set of support for S, $U \cap S'$ is one for S'.*

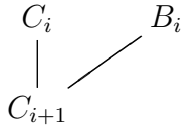
Proof. The first statement follows from compactness. For the second, if $S' \subseteq S$ is minimally unsatisfiable, and $U \cap S' = \emptyset$, then $S' \subseteq S - U$ is satisfiable, a contradiction. Hence $S' - (U \cap S')$ is a proper subset of S', and therefore satisfiable, so $U \cap S'$ is a set of support for S'. □

Theorem 11.3. (Completeness of linear resolution) *If S is unsatisfiable and U is a set of support for S, then there is a linear refutation with support U.*

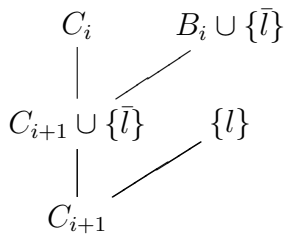
Proof. By Herbrand's Theorem 8.2, if S is unsatisfiable then so is the set S' of all ground instances of S, and the set U' of all ground instances of U is a set of support for S'. From the Lifting Lemma 10.5 it is immediate that a linear proof of $\Box$ from S' with support U' lifts to a linear proof from S with support U. So it remains to prove completeness of linear resolution for the propositional case. To this end, suppose that S is an unsatisfiable set of propositional formulas with set of support U. Without loss of generality S is finite (by compactness) and minimal (Lemma 11.2). The proof is by induction on the *excess literal number* $E(S)$, which is the number of occurrences of literals minus the number of clauses in S. We prove that there is a linear refutation proof starting with any given $C \in S$.

First note that $E(S)$ is negative if and only if there are more clauses than literals. Since every clause has at least one literal, this holds precisely when $\Box \in S$, so in this case there is nothing to prove. Hence suppose that $E(S) \geqslant 0$. There are two cases.

Case 1. The given start of the proof $C \in S$ is a unit clause $\{l\}$. Then there is $C' \in S$ containing $\bar{l}$, since otherwise a satisfying assignment for $S - C$ can be extended to S by setting l to true. Also, $l \notin C'$ because otherwise C' would be a tautology, contradicting the minimality of S. Hence $C' - \{\bar{l}\} \in S^l$, where S^l is the set defined before in Section 3. If $C' = \{\bar{l}\}$ then we are obviously done, so suppose that C' contains more literals. By Lemma 3.4, S^l is unsatisfiable. Clauses removed from S in forming S^l have at least one literal (namely l), so their removal cannot increase the excess literal number. Since C' loses $\bar{l}$ in transition to S^l we have $E(S^l) < E(S)$. As is obvious from its definition, S^l is also minimally unsatisfiable, namely every satisfying assignment of a subset of S^l gives one of S by making l true. Now the induction hypothesis gives a linear proof T of $\Box$ from S^l starting with $C' - \{\bar{l}\}$. We inductively define a linear proof of $\Box$ from S as follows. If the configuration



occurs in T then $B_i = C_j$ for some $j < i$ or $B_i \in S^l$. In the first case the induction tells us how to transform C_j, and we do the same with B_i. If $B_i \in S$ then we do not have to transform it. If $B_i \in S^l - S$ then we modify the above configuration as follows:



Case 2. The clause C contains literals other than l. In this case consider $S^{\bar{l}}$. As above, it is minimally unsatisfiable and $E(S^{\bar{l}}) < E(S)$. By induction hypothesis there is a linear deduction of $\Box$ from $S^{\bar{l}}$ starting with $C - \{l\} \in S^{\bar{l}}$. (Note that by minimality $\bar{l} \notin C$.) Add l to every center clause and any side clause in $S^{\bar{l}}$ but not S to obtain a linear proof of $\{l\}$ from S starting with C. Now consider $S' = S - \{C\} \cup \{\{l\}\}$. S' is unsatisfiable because any assignment satisfying it satisfies S. Since C contains more than one literal, $E(S') < E(S)$. Since $\Box \notin S$ also $\Box \notin S'$, hence for any $S'' \subseteq S'$ we have $E(S'') \leqslant E(S')$. Taking $S'' \subseteq S'$ minimally unsatisfiable, by induction we obtain a linear proof of $\Box$ from $S'' \subseteq S \cup \{\{l\}\}$ starting with $\{l\}$. (Note that S'' must contain $\{l\}$ because $S' - \{\{l\}\} = S - \{C\}$ is satisfiable.) Combining this with the proof of $\{l\}$ from S gives the desired proof. $\Box$

References

[1] S. R. Buss, *An introduction to proof theory*, in: S. R. Buss (ed.), *Handbook of proof theory*, Elsevier (1998) 1–78.

[2] A. Nerode and R. A. Shore, *Logic for applications*, 2nd edition, Springer 1997.