RESOLUTION THEOREM PROVING: A LOGICAL POINT OF VIEW

ALEXANDER LEITSCH

§1. Introduction. Logical calculi were invented to model mathematical thinking and to formalize mathematical arguments. The calculi of Boole [8] and of Frege [15] can be considered as the first *mathematical* models of logical inference. Their work paved the way for the discipline of metamathematics, where mathematical reasoning itself is the object of mathematical investigation. The early calculi, the so-called Hilbert type- and Gentzen-type calculi [25], [17] developed in the 20th century served the main purpose to *analyze* and to *reconstruct* mathematical proofs and to investigate *provability*. A practical use of these calculi, i.e. using them for solving actual problems (e.g. for proving theorems in "real" mathematics), was not intended and even did not make sense.

But the idea of a logical calculus as a *problem solver* is in fact much older than the origin of propositional and predicate logic in the 19th century. Indeed this idea can be traced back to G.W. Leibniz with his brave vision of a *calculus ratiocinator* [29], a calculus which would allow solution of arbitrary problems by purely mechanical computation, once they have been represented in a special formalism. Today we know that, even for restricted languages, this dream of a complete mechanization is not realizable – not even in principle (we just refer to the famous results of Gödel [20] and Turing [39]). That does not imply that we have to reject the idea altogether. Still it makes sense to search for a *lean version* of the calculus ratiocinator. Concerning the logical language, the ideal candidate is first-order logic; it is axiomatizable (and thus semidecidable), well-understood and sufficiently expressive to represent relevant mathematical structures. By Church's result [10] we know that there is no decision procedure for the validity problem of first-order logic; thus there is no procedure which is 1. capable of verifying the validity of all valid formulas and 2. terminating on all formulas. So, even in first-order logic, we have to be content with the *verification* of problems. The only thing we can hope for is a calculus which offers a basis for efficient *proof search*.

It is not surprising that the invention of the computer lead to a revival of Leibniz's dream. Indeed one of the first enterprises in the field we are calling Artificial Intelligence today was the problem of *automated theorem proving*. The very first systems were the geometry prover of Gelernter [16] and Gilmore's first-order prover [18]. Gilmore's approach was based on Herbrands's theorem and used a "projection" of predicate logic to propositional logic. By Herbrand's theorem, a universal formula F is unsatisfiable iff there exists a finite conjunction F' consisting of instances of the matrix which is unsatisfiable too. This gives a simple reduction of predicate logic to propositional logic which looks quite

natural from the point of view of logical complexity. But *computationally* the method proved to be highly ineffective. In fact there are two main sources of inefficiency: 1. the size of F' and 2. the inference on F' itself. While the efficiency of 2. was considerably increased by the method of Davis and Putnam [11], the search for F' and also the large size of F' remained serious obstacles even in proving most simple first-order theorems. Obviously there was a need for a natural technique employed by humans in the finding of proofs, which did not receive much attention in mathematical logic so far. This technique is unification and the invention of the unification principle by J.A. Robinson [37] in the early sixties brought the decisive breakthrough and marks the very beginning of automated deduction. Indeed the consequent use of the unification principle lead to a substantially new type of calculus. Robinson's resolution principle, a combination of most general unification and atomic cut on conjunctive normal forms, opens the way for a large variety of computational calculi and still forms the logical basis of the most efficient theorem proving programs.

But even on the basis of the unification principle theorem proving programs turned out too weak to automatically handle substantial or even unsolved problems of real mathematics. One of the few exceptions is the solution of Robbin's problem by W. McCune [35]. This problem is purely equational and its solution requires special techniques of equational reasoning which will not be presented in this paper. On the other hand, the resolution principle had a strong impact on computer science and became the decisive tool in the development of logic programming. Other interesting applications of resolution can be found in mathematical logic itself. Two of them, the decision problem of first-order logic and cut-elimination, will be presented in this paper. Our aim is to illustrate that the resolution calculus is more than just a principle for a (partial) automatization of logical inference. In some sense resolution is the assembler language of deduction, which – by its simplicity – makes it possible to recognize new features of deductions and to develop a new *understanding* of formal proofs and of inference in general.

§2. The Logical Basis of Resolution. In contrast to most logic calculi for first-order logic resolution does not work on the full syntax of predicate logic but on *normal forms*. These normal forms, in fact clausal normal forms, have the advantage to admit simpler inference systems and thus a more efficient way of proof search. Roughly speaking the clausal form is a *logic free* form, where the whole logical structure of the original formula is coded on the level of atomic formulas. A proof of a first-order formula A via resolution is based on the refutation of a normal form of $\neg A$, the general principle being proof by contradiction on normal forms. So we distinguish two phases in a proof of a sentence A:

- Transform $\neg A$ to a clause form $\mathcal{C}$.
- Apply resolution to $\mathcal{C}$.

2.1. Normal Forms. In constructing normal forms we have to take care that the following conditions are fulfilled: 1. soundness, 2. efficiency and 3. preservation of structure. Point 1 is the most important one, though we will see

that the concept of soundness need not be the usual one. Without taking care of point 2 the enterprise of computational inference does not really makes sense. Note that a sound transformation to normal forms would be to replace every unsatisfiable formula by falsum and every satisfiable one by itself; then proof by contradiction becomes trivial as falsum would be the only normal form for unsatisfiable formulas. The pathological aspect in such a transformation clearly is that it involves the whole complexity of theorem proving itself, i.e. to detect the unsatisfiability of a formula. Thus we have to take care that the *computation* of normal forms is simple, at best within polynomial time. Moreover it should be decidable within polynomial time whether a given formula is in normal form or not. Point 3 in the least trivial because is not completely clear which structure we mean and why it has to be preserved. We will address this point later and particularly in Section 3. Before we give a formal definition of a transformation to clause form we illustrate the main steps in an example:

EXAMPLE 2.1. Let A be the formula

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

The first step consists in transforming $\neg A$ to a formula F_1 which does not contain $\rightarrow$ and where negation only occurs in front of atoms. To this aim we apply the transformations

$$(\mathcal{A} \rightarrow \mathcal{B}) \Rightarrow (\neg\mathcal{A} \vee \mathcal{B}),$$
$$\neg(\mathcal{A} \rightarrow \mathcal{B}) \Rightarrow (A \wedge \neg B),$$
$$\neg(\exists v)\mathcal{A} \Rightarrow (\forall v)\neg\mathcal{A}.$$

and obtain the formula

$$F_1: \ [(\forall x)(\exists y)P(x,y) \wedge (\exists u)(\forall v)(\neg P(u,v) \vee Q(v))] \wedge (\forall z)\neg Q(z).$$

Clearly all the transformations preserve logical equivalence and therefore F_1 is logically equivalent to $\neg A$.

In the next step we eliminate the existential quantifiers by a technique which is generally called "skolemization". Roughly spoken, we delete an existential quantifier and replace its variable by a functional term containing all variables of universal quantifiers which are "above" the existential one. By this technique we replace $(\forall x)(\exists y)P(x,y)$ by $(\forall x)P(x,f(x))$ and $(\exists u)(\forall v)(\neg P(u,v) \vee Q(v))$ by $(\forall v)(\neg P(a,v) \vee Q(v))$; so we obtain

$$F_2: (\forall x)P(x,f(x)) \wedge (\forall v)(\neg P(a,v) \vee Q(v)) \wedge (\forall z)\neg Q(z).$$

Note that F_2 is not logically equivalent to F_1, the formulas are merely equivalent with respect to satisfiability (in this case they are both unsatisfiable)!

In F_2 the quantifiers do not carry much "information" anymore; in particular they can be shifted in front and arbitrarily commuted without destroying the logical equivalence to F_2. Thus we simply drop the quantifiers and obtain

$$F_3: P(x,f(x)) \wedge (\neg P(a,v) \vee Q(v)) \wedge \neg Q(z).$$

F_3 is quantifier free and any universal closure of F_3 is logically equivalent to F_2. In fact F_3 is already in conjunctive normal form, i.e. it consists of a conjunction of disjunctions where the disjunctions are composed of literals (i.e. of negated

and unnegated atoms). We can easily represent this formula without any logical connective using a representation by sequents: we replace any disjunction containing the positive literals $A_1, \ldots, A_n$ and the negative literals $B_1, \ldots, B_m$ by the expression

$$B_1, \ldots, B_m \vdash A_1, \ldots, A_n.$$

These logic free expressions are called *clauses*. In particular F_3 yields the clauses

$$C_1 = \vdash P(x, f(x)),$$
$$C_2 = P(a, v) \vdash Q(v),$$
$$C_3 = Q(z) \vdash .$$

F_3 as a whole can eventually be represented by the set of clauses $\mathcal{C}$: $\{C_1, C_2, C_3\}$. $\mathcal{C}$ is called a *clause form* of the original formula $\neg A$. ♯

The example above shows the typical three stages of a *standard* normal form transformation:

- Transformation to negation normal form,
- transformation to Skolem form,
- transformation to conjunctive normal form.

Below we give formal definitions of the various (intermediary) normal forms obtained in the transformation to clause form:

DEFINITION 2.1. Let A a closed first-order formula. A is in negation normal form (NNF) if it fulfils the following conditions:

1. the propositional connectives of A are in $\{\wedge, \vee, \neg\}$,
2. $\neg$ occurs only in front of atoms.

Informally expressed, a formula is in NNF if only atoms are of negative polarity. An obvious way to transform a formula into an equivalent formula in NNF consists in 1. replacing $\rightarrow$ by $\vee$, 2. apply the rules of de Morgan, and 3. shift negations over quantifiers and dualize them, 4. eliminating multiple negations.

After transformation to NNF existential quantifiers are "really" existential in the semantic sense. For technical reasons we assume that the formulas are *rectified*, i.e. every variable is quantified at most once in the formula. In the next step the existential quantifiers of the formula are eliminated and a purely universal formula is produced. If (Qx) is a quantifier in A ((Qx) is uniquely determined as A is rectified) we write $A_{-(Qx)}$ for the formula A after omission of the occurrence of (Qx).

DEFINITION 2.2. Let A be a formula in negation normal form. We define an operator *sk* in the following way:
If A does not contain existential quantifiers then $sk(A) = A$. Now assume that A contains existential quantifiers and that $(\exists x)$ is the first one. We distinguish two cases:

- If $(\exists x)$ is not in the scope of universal quantifiers then

  $$sk(A) = sk(A_{-(\exists x)}\{x \leftarrow a\})$$

  where a is a constant symbol not occurring in A.

- If $(\exists x)$ is in the scope of the universal quantifiers $(\forall y_1), \ldots, (\forall y_m)$ then

$$sk(A) = sk(A_{-(\exists x)}\{x \leftarrow f(y_1, \ldots, y_m)\})$$

where f is an m-ary function symbol not occurring in A.

$sk(A)$ is called the *Skolem form* of A. ♯

Remark: Note that the recursive definition of sk is well-founded: if there are existential quantifiers in the formulas then the argument of sk on the right hand side has always one existential quantifier less. For illustration let A be $(\forall y)(\exists z)(\exists x)P(y, z, x)$. Then

$$\begin{aligned} sk(A) &= sk((\forall y)(\exists x)P(y, f(y), x) \\ &= sk((\forall y)P(y, f(y), g(y)) \\ &= (\forall y)P(y, f(y), g(y)). \end{aligned}$$

♯

As we already pointed out in Example 2.1 sk does usually not preserve logical equivalence. Nevertheless the transformation is *weakly sound* i.e. it preserves sat-equivalence:

THEOREM 2.1. *Let A be a formula in negation normal form. Then $sk(A) \sim_{sat} A$, i.e. $sk(A)$ is satisfiable iff A is satisfiable.*

PROOF. We only give the idea of the proof; a full formal proof can be found in [30].

By definition of sk it is easy to see that $sk(A) \rightarrow A$ is valid. Thus the satisfiability of $sk(A)$ implies that of A. For the other direction assume that A is satisfiable and that $\mathcal{M}$ is a model of A. Then a model $\mathcal{M}'$ for $sk(A)$ is obtained by extending $\mathcal{M}$ by an appropriate interpretation of the new function symbol. ⊣

In particular A is refutable iff $sk(A)$ is refutable. Thus, in some sense, we are not proving $\neg A$ itself, but rather $\neg sk(A)$. But such problem reductions are quite natural and standard in the practice of mathematical proofs.

Once we have constructed a Skolem form it remains to transform the formula to conjunctive normal form. Again there exists a straightforward method using the rules of distributivity. Although this method is quite simple it can lead to an exponential blow up of the formula size and, much worse, to an even nonelementary increase of proof length. We will come back to this problem in Section 3 where we present an alternative way of constructing clause forms.

DEFINITION 2.3. Let $A_1, \ldots, A_n, B_1, \ldots, B_m$ be atom formulas. Then the expression $C\colon A_1, \ldots, A_n \vdash B_1, \ldots, B_m$ is called a *clause*. C represents the formula $F\colon \neg A_1 \vee \cdots \vee \neg A_n \vee B_1 \vee \cdots \vee B_m$. The empty clause $\vdash$ represents falsum. Assume that the clauses C_i represent the formulas F_i for $i = 1, \ldots, k$; then $\{C_1, \ldots, C_n\}$ represents the universal closure of $F_1 \wedge \cdots \wedge F_k$. ♯

To sum up, in proving A by contradiction we reduce $\neg A$ to a set of clauses $\mathcal{C}$ representing the universal closure of a conjunctive normal form obtained via the transformation steps described above. These sets $\mathcal{C}$ are the very raw material for resolution.

2.2. The Resolution Principle. In some sense the resolution principle is the simplest first-order calculus which is possible at all. It is based on one single rule which combines substitution and atomic cut. Its most typical feature is a *binary* substitution rule computing a minimal (i.e. most general) substitution which makes two atoms equal.

EXAMPLE 2.2. Let A be the formula

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

of Example 2.1. We have shown that the clausal form of $\neg A$ is

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

We refute $\mathcal{C}$ by using two rules 1. substitution and 2. the atomic cut rule

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

But the substitution rule is tied to the cut and serves as preparatory step. The following derivation ϕ illustrates this property:

$$\dfrac{\dfrac{\dfrac{\vdash P(x,f(x))}{\vdash P(a,f(a))}\ S \quad \dfrac{P(a,v) \vdash Q(v)}{P(a,f(a)) \vdash Q(f(a))}\ S}{\vdash Q(f(a))}\ cut \quad \dfrac{Q(z) \vdash}{Q(f(a)) \vdash}\ S}{\vdash}\ cut$$

ϕ is indeed a refutation of the formula represented by $\mathcal{C}$. The substitution rule is sound by the validity of the substitution axioms and by the fact that a clause represents the universal closure of a disjunction. The cut rule is sound because it "represents" the rule

$$\frac{B \vee A \quad \neg A \vee C}{B \vee C}$$

♯

In general there are infinitely many substitutions unifying two atoms, but only the *most general ones* are actually needed. The observation of this effect and the invention of the *unification principle* by J.A. Robinson (see [37]) is a landmark in the history of computational logic and automated deduction.

We may compare substitutions and expressions with regard to their "generality", i.e., whether they can be obtained from other substitutions (expressions) by instantiation.

DEFINITION 2.4 (generality). Let E_1 and E_2 be expressions. We say that E_1 is *more* (or equally) *general* than E_2 (notation $E_1 \leq_s E_2$) if there exists a substitution σ such that $E_1\sigma = E_2$. Let σ, τ be two substitutions. We define $\sigma \leq_s \tau$ (σ is more general than τ) if there exists a substitution ϑ such that $\sigma\vartheta = \tau$.

We can go back now to the problem of unifying atoms, literals, and terms (i.e., expressions) by substitutions.

EXAMPLE 2.3. Consider the atoms

$$A_1 = P(x, f(y), f(y)) \text{ and}$$
$$A_2 = P(x', y', f(x')).$$

Let t be an arbitrary term different from x and x'. Then the substitution $\sigma_t\colon \{x \leftarrow t, y \leftarrow t, x' \leftarrow t, y' \leftarrow f(t)\}$ fulfils

$$\begin{aligned} P(x, f(y), f(y))\sigma_t &= P(x', y', f(x'))\sigma_t \\ &= P(t, f(t), f(t)). \end{aligned}$$

The same property holds for the substitution

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

Clearly $\sigma \leq_s \sigma_t$ by $\sigma\{x \leftarrow t\} = \sigma_t$. ♯

The example above motivates the following definition.

DEFINITION 2.5 (unifier). Let E be a nonempty set of expressions. A substitution σ is called a *unifier* of E if for all $e, e' \in E$ $e\sigma = e'\sigma$. σ is called a *most general unifier* (m.g.u.) of E if for every unifier θ of E we have $\sigma \leq_s \theta$.

Remark: A unifier of a two-element set $\{e, e'\}$ is frequently called a unifier of e and e'. If the set of expressions E consists of one element only then every substitution is a unifier of E, and the identical substitution ϵ is the m.g.u. of E. ♯

Still two questions arise:

1. Is there always an m.g.u. of a unifiable set of expressions?
2. Are m.g.u.s effectively computable?

Fortunately the answer to both questions is *yes* as we will point out below. Note that computing a unifier of a set of atoms $\{A_1, \ldots, A_n\}$, i.e. a substitution θ with $A_1\theta = \ldots = A_n\theta$, can easily be reduced to the problem of unifying two atoms and even of unifying two terms. E.g. any unifier θ of $\{P(x), P(y), P(f(z))\}$ is also a unifier of the two terms $g(x, y, f(z))$ and $g(x, x, x)$ and vice versa.

We call any algorithm computing an m.g.u. in case of unifiability and deciding unifiability a *unification algorithm*. As the performance of theorem provers strongly depends on the efficiency of unification algorithms, many sophisticated algorithms for computing m.g.u.s have been developed so far. Besides the original algorithm in Robinson's paper [37] we just mention the algorithm of Martelli and Montanari [33]. In this algorithm, unification is considered as the problem of finding a *most general solution* for a system of term equations. We also follow this line and present a simple rule based approach like in [2].

EXAMPLE 2.4. Let

$$A_1 : P(g(x), f(x, z)) \text{ and}$$
$$A_2 : P(g(g(u)), v).$$

The problem of unifying A_1 and A_2 can be reduced to solving the system of equations

$$\mathcal{E}_1\colon \{g(x) \doteq g(g(u)),\ v \doteq f(x, z)\},$$

i.e. we are searching for a substitution θ s.t.

$$g(x)\theta = g(g(u))\theta,\ v\theta = f(x,z)\theta.$$

(called a solution) s.t. for all other solutions η we have $\theta \leq_s \eta$.

Clearly the equations cannot be "read" as a substitution directly. But we observe that θ is a solution of $\mathcal{E}_1$ iff θ is a solution of $\mathcal{E}_2$ for

$$\mathcal{E}_2 = \{x \doteq g(u),\ v \doteq f(x,z)\}.$$

$\mathcal{E}_1$ and $\mathcal{E}_2$ are *equivalent* because substitutions are homomorphisms on terms. Thus we are allowed to *decompose* the equation $g(x) \doteq g(g(u))$ into $x \doteq g(u)$.

The system $\mathcal{E}_2$ can be "interpreted" as the substitution $\theta\colon \{x \leftarrow g(u), v \leftarrow f(x,z)\}$; but θ is not a unifier. Indeed $A_1\theta \neq A_2\theta$. So we apply another transformation and interpret the equation $x \doteq g(u)$ as a substitution on the system $\mathcal{E}_2$. This gives us the equivalent system

$$\mathcal{E}_3\colon \{x \doteq g(u),\ v \doteq f(g(u),z)\}.$$

In $\mathcal{E}_3$ every equation is of the form $x \doteq t$, where x is a variable and t is a term, and the variables on the lefthandside of the equations occur only once in the whole system. What we have obtained is a system in *solved form*. Indeed, if we read the equations as substitution

$$\sigma :\ \{x \leftarrow g(u),\ v \leftarrow f(g(u),z)\}$$

then σ is indeed a most general solution of $\mathcal{E}_3$ and thus of $\mathcal{E}_1$. Clearly σ is also an m.g.u. of A_1 and A_2. ♯

Thus finding a most general unifier means to find a most general solution of a system of term equations. And solving such a system of equations means to transform it into an equivalent solved form. Clearly we must also define criteria for the unsolvability of such systems and to ensure that our transformations on the systems terminate.

DEFINITION 2.6. A system of term equations is a finite set of the form

$$\mathcal{E}\colon \{s_1 \doteq t_1, \ldots, s_n \doteq t_n\}$$

where s_i, t_i are terms for $i = 1, \ldots, n$. A substitution θ is called a *solution* of $\mathcal{E}$ if for all $i = 1, \ldots, n\colon s_i\theta = t_i\theta$. A solution σ is called a *most general solution* of $\mathcal{E}$ if for all solutions θ of $\mathcal{E}$ $\sigma \leq_s \theta$. Two systems $\mathcal{E}$ and $\mathcal{E}'$ are called *equivalent* if they have the same set of solutions.

We say that $\mathcal{E}$ is in *solved form* if

- $\{s_1, \ldots, s_n\} \subseteq V$ and
- every s_i occurs only (and thus exactly) once in $\mathcal{E}$ (for $i = 1, \ldots, n$).

By definition a solved form represents a substitution. It is easy to see that for a solved form

$$\mathcal{E}\colon\ \{x_1 \doteq t_1, \ldots, x_n \doteq t_n\}$$

the substitution $\sigma\colon \{x_1 \leftarrow t_1, \ldots, x_n \leftarrow t_n\}$ is a most general solution of $\mathcal{E}$. σ is also called the *substitution defined by* $\mathcal{E}$.

Equality is reflexive; thus any equation of the form $s \doteq s$ is redundant. By the symmetry of equality $s \doteq t$ is equivalent to $t \doteq s$. As we are interested in

interpreting some equations as substitutions, equations of the form $t \doteq x$, for a term t which is not a variable, are transformed to $x \doteq t$. The observation above leads to the following two transformation rules:

(trivial): If the system $\mathcal{E}$ contains an equation $s \doteq s$ then replace $\mathcal{E}$ by $\mathcal{E} \setminus \{s \doteq s\}$.

(orient): If $\mathcal{E}$ contains an equation of the form $t \doteq v$, where v is a variable and t is not, then replace $\mathcal{E}$ by $(\mathcal{E} \setminus \{t \doteq v\}) \cup \{v \doteq t\}$.

There are systems which are unsolvable and thus cannot be transformed into solved form. As we have to avoid nonterminating procedures we have to detect some typical cases of unsolvability. If we recognize the unsolvability of a system S we reduce it to $\perp$ (where $\perp$ can be interpreted as a fixed unsolvable system).

DEFINITION 2.7 (failure rules). Let $\mathcal{E}$ be a system of term equations and $s \doteq t$ be an equation in $\mathcal{E}$.

(symbol clash): If s and t are functional terms having different leading symbols then $\mathcal{E}$ is replaced by $\perp$.

(occurs check): If s is a variable s.t. $s \neq t$ and s occurs in t then replace $\mathcal{E}$ by $\perp$. ♯

EXAMPLE 2.5. Let

$$\mathcal{E}\colon\ \{x \doteq y, x \doteq g(y)\}.$$

$\mathcal{E}$ is unsolvable: there exists no substitution θ s.t. $x\theta = y\theta$ and $x\theta = g(y)\theta$; for otherwise $y\theta = g(y)\theta = g(y\theta)$, which is impossible. However, neither (symbol clash) nor (occurs check) is actually applicable. But if we apply $x \doteq y$ as substitution $\{x \leftarrow y\}$ to the other equation we obtain the equivalent system

$$\mathcal{E}'\colon\ \{x \doteq y, y \doteq g(y)\},$$

which can be reduced to $\perp$ via (occurs check). ♯

The example above shows that we need additional rules for deciding unifiability and for computing most general unifiers. These rules, decomposition and replacement, are in fact the most important ones.

(decomposition): Let $s \doteq t$ be an equation in $\mathcal{E}$ s.t. there exists a function symbol $f \in \mathrm{FS}_n$ and terms $s_1, \ldots, s_n, t_1, \ldots, t_n$ with

$$s = f(s_1, \ldots, s_n),$$
$$t = f(t_1, \ldots, t_n).$$

Then replace $\mathcal{E}$ by the system

$$\mathcal{E}'\ :\ (\mathcal{E} \setminus \{s \doteq t\}) \cup \{s_1 \doteq t_1, \ldots, s_n \doteq t_n\}.$$

(replacement): Let $s \doteq t$ be an equation in $\mathcal{E}$ s.t.

(1) s is a variable and

(2) s does not occur in t and

(3) s occurs in $\mathcal{E} \setminus \{s \doteq t\}$.

Then replace $\mathcal{E}$ by the system

$$\mathcal{E}'\ :\ (\mathcal{E} \setminus \{s \doteq t\})\{s \leftarrow t\} \cup \{s \doteq t\}.$$

DEFINITION 2.8. Let $\Re$ be the set of rules {(trivial), (orient), (symbol clash), (occurs check), (decomposition), (replacement)}. If $\mathcal{E}$ is transformed to $\mathcal{E}'$ via a rule in $\mathcal{R}$ we write $\mathcal{E} \succ_{\Re} \mathcal{E}'$. For the reflexive and transitive closure of the relation $\succ_{\Re}$ we write $\succ_{\Re}^*$. A system $\mathcal{E}$ is called *irreducible* if none of the rules in $\Re$ is applicable to $\mathcal{E}$. $\perp$ is irreducible by definition. ♯

According to the definitions above, unification means nothing else than reduction of a system to an irreducible form (where an arbitrary order of applications of rules in $\Re$ is admitted).

EXAMPLE 2.6. We take the system $\mathcal{E}$ from Example 2.4 where

$$\mathcal{E} = \{g(x) \doteq g(g(u)),\ v \doteq f(x,z)\}.$$

Then $\mathcal{E} \succ_{\Re} \mathcal{E}'$ (via (decomposition)) where

$$\mathcal{E}' = \{x \doteq g(u),\ v \doteq f(x,z)\}.$$

Now (replacement) is applicable and $\mathcal{E}' \succ_{\Re} \mathcal{E}''$, where

$$\mathcal{E}'' = \{x \doteq g(u),\ v \doteq f(g(u),z)\}.$$

$\mathcal{E}''$ is irreducible and in solved form. ♯

Remark: Note that every solved form is irreducible. On the other hand every irreducible form is either a solved form or $\perp$.

THEOREM 2.2 (unification theorem). *$\Re$ is a unification system, i.e*

- *$\Re$ always terminates.*
- *If the system $\mathcal{E}$ is solvable then $\mathcal{E} \succ_{\Re}^* \mathcal{E}'$ s.t. $\mathcal{E}'$ is in solved form and the substitution defined by $\mathcal{E}'$ is a most general solution of $\mathcal{E}$.*

PROOF. (sketch)
For a full proof see [2]. The most involved part is termination.

If we know that the system is terminating our task is easier: all rules in $\Re$ preserve the equivalence of systems. So let $\mathcal{E}'$ be an irreducible system obtained from $\mathcal{E}$. If $\mathcal{E}' = \perp$ then $\mathcal{E}'$ and $\mathcal{E}$ itself are unsolvable; else $\mathcal{E}'$ must be in solved form (for otherwise it would not be irreducible). Thus the substitution defined by $\mathcal{E}'$ is a most general solution of $\mathcal{E}'$ and of $\mathcal{E}$ itself. ⊣

We have seen in Example 2.2 how an unsatisfiable set of clauses can be refuted via unification and cut. The example below shows that unification cannot be restricted to atoms of different clauses but must also be applied within clauses.

EXAMPLE 2.7. Let $\mathcal{C} = \{C_1, C_2\}$ for

$$C_1 = \vdash P(x), P(y),$$
$$C_2 = P(u), P(v) \vdash .$$

Clearly $\mathcal{C}$ is unsatisfiable, but we will see that without an additional technique $\vdash$ cannot be derived.

Even if we permute $P(x)$ and $P(y)$ within C_1 and $P(u), P(v)$ within C_2 and try all unifications between atoms in C_1 and C_2 we obtain a variant of the clause C_3: $P(x) \vdash P(y)$. But, modulo variable renaming, C_1 with C_3 gives C_1 and C_2 with C_3 gives C_2. We see that cutting out single atoms does not suffice to

derive contradiction. What we need is unification within a clause followed by a contraction rule.

Let us apply the substitution $\lambda\colon \{x \leftarrow y\}$ to C_1; then we obtain the clause

$$\vdash P(y), P(y).$$

But $\vdash P(y), P(y)$ represents $(\forall y)(P(y) \vee P(y))$ which is logically equivalent to $(\forall y)P(y)$. Thus we may apply contraction within $\vdash P(y), P(y)$ and obtain $C_1'\colon \vdash P(y)$. Similarly we obtain by $\mu\colon \{u \leftarrow v\}$ the clause $C_2'\colon P(v) \vdash$ from C_2. Now by unifying $P(y), P(v)$ via the m.g.u. $\sigma\colon \{y \leftarrow v\}$ we eventually obtain $C_1''\colon \vdash P(v)$ and $C_2''\colon P(v) \vdash$ which resolve to $\vdash$ via atomic cut. C_1' and C_2' are called factors of C_1 and of C_2 respectively. The steps are illustrated in the derivation below (S stands for substitution as preparation for the cut and F for factoring):

$$\dfrac{\dfrac{\dfrac{\vdash P(x), P(y)}{\vdash P(y)}\,F}{\vdash P(v)}\,S \qquad \dfrac{P(u), P(v) \vdash}{P(v) \vdash}\,F}{\vdash}\,cut$$

♯

DEFINITION 2.9 (factor). Let

$$C : \Gamma \vdash \Delta_1, A_1, \ldots, \Delta_n, A_n, \Delta_{n+1}$$

($C\colon \Delta_1, A_1, \ldots, \Delta_n, A_n, \Delta_{n+1} \vdash \Gamma$) be a clause where $n \geq 1$ and the Δ_i are (possibly empty) sequences of atoms. Moreover let σ be a most general unifier of $\{A_1, \ldots, A_n\}$. Then

$$C' : \Gamma\sigma \vdash \Delta_1\sigma, \ldots, \Delta_n\sigma, A_n\sigma$$

($C'\colon A_n\sigma, \Delta_1\sigma, \ldots, \Delta_n\sigma \vdash \Gamma\sigma$) is called a *factor* of C. ♯

Remark: If the clause does not contain variables then a factor is obtained just by contraction of identical atoms. If in Definition 2.9 $n = 1$ then σ is the identical substitution and the atom A_1 is put to the extreme right, respectively left, of the clause. Thus we are using factoring also to position the atoms at the appropriate places for the following cut. ♯

DEFINITION 2.10 (binary resolvent). Let $C_1\colon \Gamma_1 \vdash \Delta_1, A$ and $C_2\colon B, \Gamma_2 \vdash \Delta_2$ be two clauses which are variable disjoint and A, B be unifiable with m.g.u. σ. Then the clause

$$\Gamma_1\sigma, \Gamma_2\sigma \vdash \Delta_1\sigma, \Delta_2\sigma$$

is called a *binary resolvent* of C_1 and C_2.

After these preparations we are ready for the definition of the general resolution rule.

DEFINITION 2.11 (resolution). Let C_1, C_2 be two clauses and C_1', C_2' be variable disjoint variants of factors of C_1, C_2 and C be a binary resolvent of C_1' and C_2'. Then C is called a (general) *resolvent* of C_1 and C_2.

By Definition 2.11 it is possible to define infinitely many resolvents of two clauses, as there are infinitely many possible variable renamings. We can avoid this effect by a standard renaming of the resolvent by variables $\{x_1, \ldots, x_n\}$. Using such a standard renaming there are indeed only *finitely many* resolvents of two clauses.

EXAMPLE 2.8. Let

$$C_1 = R(x, y) \vdash P(x), P(y),$$
$$C_2 = Q(x), P(f(x)) \vdash S(x, x).$$

Then, under a standard renaming, the following clauses are resolvents of C_1 and C_2:

$$C_3 = R(x_1, f(x_2)), Q(x_2) \vdash P(x_1), S(x_2, x_2),$$
$$C_4 = R(f(x_1), x_2), Q(x_1) \vdash P(x_2), S(x_1, x_1),$$
$$C_5 = R(f(x_1), f(x_1)), Q(x_1) \vdash S(x_1, x_1). \sharp$$

The following deduction principle on clause logic is characterized by using only one single inference rule, namely resolution.

DEFINITION 2.12 (resolution deduction). A resolution deduction (R-deduction) of a clause C from a set of clauses $\mathcal{C}$ is a sequence of clauses $C_1, \ldots, C_n$ with the following properties:

1. $C_n = C$.
2. For every i with $i \in \{1, \ldots, n\}$ we have: either C_i is a clause in $\mathcal{C}$ or C_i is a resolvent of two clauses C_j, C_k with $j, k < i$.

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

EXAMPLE 2.9. Let

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

Then

$$\phi\colon\ \vdash P(x, f(x));\ P(a, v) \vdash Q(v);\ \vdash Q(f(a));\ Q(z) \vdash;\ \vdash$$

is an R-refutation of $\mathcal{C}$. In tree format ϕ is of the form

$$\dfrac{\dfrac{\vdash P(x, f(x)) \quad P(a, v) \vdash Q(v)}{\vdash Q(f(a))}\, R \qquad Q(z) \vdash}{\vdash}\, R$$

Modulo the clause form transformation ϕ can be considered as a proof by contradiction of the formula

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

Resolution is a complete inference principle on clause logic and, together with normal form transformations, a refutationally complete inference system for first-order logic.

THEOREM 2.3. *Resolution is* complete, *i.e. if $\mathcal{C}$ is an unsatisfiable set of clauses then there exists an R-refutation of $\mathcal{C}$.*

PROOF. We merely give the main line of the proof; a detailed proof can be found in [30].

By Herbrand's theorem a set of clauses $\mathcal{C}$ is unsatisfiable iff there exists a finite set $\mathcal{C}'$ of ground (i.e. variable-free) clauses, obtained by instantiation of clauses in $\mathcal{C}$, which is unsatisfiable. Thus the unsatisfiability of the first-order problem $\mathcal{C}$ is reduced to that of the propositional problem $\mathcal{C}'$.

Then it is proved that resolution is complete on sets of ground clauses. As a consequence there exists an R-refutation ϕ' of $\mathcal{C}'$.

Finally the refutation ϕ' is *lifted* to a refutation ϕ of $\mathcal{C}$. Lifting is a technique replacing a resolvent C' of instances C_1', C_2' of clauses C_1, C_2 by a resolvent C of C_1, C_2 s.t. C' is an instance of C.

⊣

The lifting principle is the most significant feature of first-order resolution. Instead of producing infinitely many resolvents of instances of two clauses it is enough to use the general resolvents which are based on most general unification only. We give a simple example illustrating this principle.

EXAMPLE 2.10. Let

$$\mathcal{C} = \{\vdash P(x);\ P(u) \vdash Q(u);\ Q(f(z)) \vdash\}.$$

Let t be an arbitrary ground term over the signature $\{f, a\}$ and

$$\mathcal{C}' = \{\vdash P(f(t));\ P(f(t)) \vdash Q(f(t));\ Q(f(t)) \vdash\}.$$

Then

$$\phi_t\colon\ \vdash P(f(t));\ P(f(t)) \vdash Q(f(t));\ \vdash Q(f(t));\ Q(f(t)) \vdash;\ \vdash$$

is an R-refutation of $\mathcal{C}'$. But

$$\phi_t = \phi\{x \leftarrow f(t), u \leftarrow f(t), z \leftarrow t\}$$

for

$$\phi = \vdash P(x);\ P(u) \vdash Q(u);\ \vdash Q(u);\ Q(f(t)) \vdash;\ \vdash .$$

But ϕ is an R-refutation of $\mathcal{C}$. ♯

§3. Resolution and LK.

3.1. The calculus LK. Traditional logic calculi work on the full syntax of logic and not on normal forms. This has the advantage that the logical structure of the theorems is preserved and the semantical meaning is well-presented in the different steps of a derivation. The resolution calculus, in contrast, works on simple normal forms which favor efficient inference, but on the other hand its derivations look ugly and "unnatural". Moreover we even risk to lose structural information which is useful in defining good and/or short proofs. We will show in this section that there is a remedy for this defect of resolution, namely structural clause form transformation. This technique allows us to define an "interface" to standard logic calculi and to *store* the structure of full first-order logic without sacrificing efficiency. As "traditional" logic calculus we choose Gentzen's **LK** because it is the most elegant and "semantic" calculus for classical first-order logic. The elements **LK** is working with are sequents:

DEFINITION 3.1 (sequent). A sequent is an expression of the form $\Gamma \vdash \Delta$ where Γ and Δ are finite multisets of PL-formulas (i.e. two sequents $\Gamma_1 \vdash \Delta_1$ and $\Gamma_2 \vdash \Delta_2$ are considered equal if the multisets represented by Γ_1 and by Γ_2 are equal and those represented by Δ_1, Δ_2 are also equal). ♯

Note that clauses are sequents containing only atomic formulas.

DEFINITION 3.2 (the calculus **LK**). The initial sequents are $A \vdash A$ for first-order formulas A. In the rules of **LK** we always mark the auxiliary formulas (i.e. the formulas in the premise(s) used for the inference) and the principal (i.e. the inferred) formula using different marking symbols. Thus, in our definition, $\wedge$-introduction to the right takes the form

$$\frac{\Gamma_1 \vdash A^+, \Delta \qquad \Gamma_2 \vdash \Delta_2, B^+}{\Gamma_1, \Gamma_2 \vdash \Delta_1, A \wedge B^*, \Delta_2} \wedge : r$$

We usually avoid markings by putting the auxiliary formulas at the leftmost position in the antecedent of sequents and in the rightmost position in the consequent of sequents. The principal formula mostly is identifiable by the context. Thus the rule above will be written as

$$\frac{\Gamma_1 \vdash \Delta_1, A \qquad \Gamma_2 \vdash \Delta_2, B}{\Gamma_1, \Gamma_2 \vdash \Delta_1, \Delta_2, A \wedge B} \wedge : r$$

The version of **LK** we are using here is that in [4] and slightly deviates from Gentzen's original version [17]. The differences however are without importance to the results presented in this section. Readers who are interested in a detailed definition of **LK** are referred to [17] or to [38] ♯

The main result of Gentzen's famous paper [17] was the cut-elimination theorem. It shows that, in arbitrary **LK**-proofs, the cut rule

$$\frac{\Gamma_1 \vdash \Delta_1, A \quad A, \Gamma_2 \vdash \Delta_2}{\Gamma_1, \Gamma_2 \vdash \Delta_1, \Delta_2} \; cut$$

can be eliminated; the result is a proof with the *subformula property*, i. e., the whole proof is made of the syntactic material of the end sequent. By this property Gentzen's cut-free **LK** can be used as a basis for proof search and automated deduction when combined with the unification principle; the corresponding calculus is the *tableaux-calculus* [22]. For illustration we give a simple cut-free **LK**-proof of the sequent

$$P(a), (\forall x)(P(x) \to P(f(x))) \vdash P(f^2(a))$$

which (semantically) stands for the formula

$$[P(a) \wedge (\forall x)(P(x) \to P(f(x)))] \to P(f^2(a)).$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{P(f(a)) \vdash P(f(a)) \quad P(f^2(a)) \vdash P(f^2(a))}{P(a) \vdash P(a) \quad P(f(a)), P(f(a)) \to P(f^2(a)) \vdash P(f^2(a))} \to : l}{P(f(a)) \to P(f^2(a)), P(a) \to P(f(a))), P(a) \vdash P(f^2(a))} \to : l}{(\forall x)(P(x) \to P(f(x))), P(a) \to P(f(a))), P(a) \vdash P(f^2(a))} \forall : l}{(\forall x)(P(x) \to P(f(x))), (\forall x)(P(x) \to P(f(x))), P(a) \vdash P(f^2(a))} \forall : l}{(\forall x)(P(x) \to P(f(x))), P(a) \vdash P(f^2(a))} c : l$$

3.2. Structural Clause Form. The key to the *simulation* of cut-free **LK** by resolution is the *structural normal form* transformation. In contrast to the standard normal form transformation presented in Subsection 2.1 the structural one is based on introductions of new predicate symbols. Let F be the formula to be transformed into clause form. The idea is the following one: For any subformula A in F create a new predicate symbol P_A and "define" P_A by A:

Suppose, for illustration, that $A = A_1 \circ A_2$ for $\circ \in \{\wedge, \vee, \to\}$ and that X is the set of variables free in A. Furthermore let Y be the set of variables free in A_1 and Z the set of variables free in A_2. Then clearly $X = Y \cup Z$. Define vectors of variables $\bar{x}, \bar{y}, \bar{z}$ from the sets X, Y, Z. Then the *defining formula* for P_A is:

$$(\forall \bar{x})[P_A(\bar{x}) \leftrightarrow (P_{A_1}(\bar{y}) \circ P_{A_2}(\bar{z}))].$$

Similarly the defining formula for $A = \neg B$ is $(\forall \bar{x})(P_A(\bar{x}) \leftrightarrow \neg P_B(\bar{x}))$ and for $A = (Qy)B$ it is

$$(\forall \bar{x})(P_A(\bar{x}) \leftrightarrow (Qy)P_B(\bar{x}, y)).$$

Note that the defining formulas just represent a mathematical tool which is widely used in mathematics, namely *extension by definition.* The extension formulas directly yield the structural clause form.

DEFINITION 3.3. Let F be a closed formula and $\mathcal{F}$ be the set of all defining formulas obtained from F. Then the *structural clause form* of F is the set $\gamma_{\text{struc}}(F)$ defined by

$$\gamma_{\text{struc}}(F) = \bigcup\{\gamma_0(B) | B \in \mathcal{F}\} \cup \{\vdash A_F\}$$

where γ_0 is the operator of the standard clause form transformation and in all defining formulas $\gamma_0(G \leftrightarrow H)$ is defined by $\gamma_0((G \to H) \wedge (H \to G))$.

Like γ_0 also γ_{struc} is satisfiability preserving and thus weakly correct (a formal proof can be found in [12]). For practical reasons one can avoid to construct defining formulas for atoms. These would have the form $(\forall \bar{x})(P_Q(\bar{x}) \leftrightarrow Q(\bar{x}))$. It is easy to see that this results only in a renaming of the atoms themselves making the corresponding clauses redundant. The structural normal form subjected to this improvement is denoted by $\gamma'_{\text{struc}}(F)$.

EXAMPLE 3.1. Let $F = (\forall x)P(x) \vee (P(a) \wedge Q(b))$.
The defining formulas (omitting the atomic level) are the following ones:

$$E_1 = A_F \leftrightarrow (A_1 \vee A_2),$$
$$E_2 = A_1 \leftrightarrow (\forall x)P(x),$$
$$E_3 = A_2 \leftrightarrow (P(a) \wedge Q(b)).$$

We now compute the standard clause forms $\mathcal{C}_i$: $\gamma_0(E_i)$ and obtain

$$\mathcal{C}_1 = \{A_F \vdash A_1, A_2;\ A_1 \vdash A_F;\ A_2 \vdash A_F\},$$
$$\mathcal{C}_2 = \{A_1 \vdash P(x);\ P(c) \vdash A_1\},$$
$$\mathcal{C}_3 = \{A_2 \vdash P(a);\ A_2 \vdash Q(b);\ P(a), Q(b) \vdash A_2\}.$$

Now

$$\gamma'_{\text{struc}}(F) = \{\vdash A_F\} \cup \mathcal{C}_1 \cup \mathcal{C}_2 \cup \mathcal{C}_3.$$

Note that in computing $\mathcal{C}_2$ we have to apply skolemization and thus obtain the new constant symbol c. ♯

In Example 3.1 the structural clause form consists of nine clauses, while $\gamma_0(F) = \{\vdash P(x), P(a);\ \vdash P(x), Q(b)\}$ and thus consists of two clauses only. It might appear that structural transformation is much more expensive, but this is deceptive: the worst-case complexity of γ_{struc} is quadratic (in propositional logic even linear), but that of γ_0 is exponential (even for propositional problems). Moreover γ_{struc} preserves the structure of the formula and may lead to much shorter proofs. In fact it is proven in [3] that there exists a sequence of clause sets $\mathcal{C}_n$ fulfilling the following conditions:

- $\gamma_{\text{struc}}(\mathcal{C}_n)$ has resolution refutations of length $\leq 2^{2^{dn}}$,
- all resolution refutations of $\gamma_0(\mathcal{C}_n)$ are longer than $s(n-1)$, where s is a nonelementary function defined by $s(0) = 1$, $s(n+1) = 2^{s(n)}$.

Note that $s(n)$ grows faster than any fixed iteration of the exponential function. The result above is obtained by using the complexity of cut-elimination (which is nonelementary) and the simulation of cut-free **LK** (and of **LK** with analytic cut) shown in the next subsection.

In [13] Egly and Rath proved that structural clause form transformation is not only superior in theory but also in practice. They gave a thorough experimental comparison with the standard transformation: the version of the prover using structural transformation did not only behave better in average but could handle problems unsolvable with the standard version. This shows that *preservation of logical structure* in normal form computations pays out also in practice.

3.3. Simulation of LK by Resolution. With γ_{struc} we have an operator which does not only map a formula into a set of clauses but also encodes all of its subformulas by new labels represented by new predicate symbols. This makes it possible to simulate inferences in **LK** by resolution deductions. For simplicity we only present the propositional case, for the full first-order simulation see [12] or [3].

In the first step we map sequents into clauses; all we have to do is to use atomic labels for formulas like in structural clause transformation. So a sequent of the form

$$F_1, \ldots, F_m \vdash G_1, \ldots, G_n$$

is represented by the clause

$$A_{F_1}, \ldots, A_{F_m} \vdash A_{G_1}, \ldots, A_{G_n}.$$

Now assume that we have a cut-free proof ψ of the sequent $F \vdash$ (which proves that F is unsatisfiable). Then we construct $\gamma_{\text{struc}}(F)$. The simulation of logical

introduction rules then is performed via resolution using the clauses encoding these introductions. We show the procedure for $\wedge : r$ and $\neg : l$ which suffices to illustrate the nature of this transformation.

Let π_1, π_2 **LK**-proofs and π be a subderivation of ψ of the form:

$$\frac{\begin{array}{cc}(\pi_1) & (\pi_2)\\ S_1 \colon\ \Gamma_1 \vdash \Delta_1, G & S_2 \colon\ \Gamma_2 \vdash \Delta_2, H\end{array}}{S_3 \colon\ \Gamma_1, \Gamma_2 \vdash \Delta_1, \Delta_2, G \wedge H}\ \wedge : r$$

Now let $C_1 \colon \Lambda_1 \vdash \Theta_1, A_G$ and $C_2 \colon \Lambda_2 \vdash \Theta_2, A_H$ be the clauses corresponding to S_1, S_2. Assume inductively that resolution derivations λ_1 of C_1 and λ_2 of C_2 already exist. We construct a resolution derivation of C, the clause representing S_3. By definition of $\gamma_{\text{struc}}(F)$ and by the fact that the formula $G \wedge H$ is a subformula of F (note that we have the subformula property!) the set $\gamma_{\text{struc}}(F)$ contains the clauses

$$A_G, A_H \vdash A_{G \wedge H};\ A_{G \wedge H} \vdash A_G;\ A_{G \wedge H} \vdash A_H.$$

Using these clauses we can construct the following resolution derivation λ:

$$\frac{\begin{array}{cc}(\lambda_2) & \dfrac{(\lambda_1)\quad \Lambda_1 \vdash \Theta_1, A_G \quad A_G, A_H \vdash A_{G \wedge H}}{A_H, \Lambda_1 \vdash \Theta_1, A_{G \wedge H}}\ R\\ \Lambda_2 \vdash \Theta_2, A_H & \end{array}}{\Lambda_2, \Lambda_1 \vdash \Theta_1, \Theta_2, A_{G \wedge H}}\ R$$

which is a resolution derivation of a clause representing S_3. So λ simulates π.

Now we illustrate the case of the negation introduction to the left. Let π be a subproof of ψ of the form

$$\frac{\begin{array}{c}(\pi')\\ S' \colon\ \Gamma \vdash \Delta, G\end{array}}{S \colon\ \neg G, \Gamma \vdash \Delta}\ \neg : l$$

and λ' be a resolution proof of the clause $\Lambda \vdash \Theta, A_G$ representing S'. As $\neg G$ is a subformula of F the set $\gamma_{\text{struc}}(F)$ contains the clauses $\vdash A_G, A_{\neg G}$ and $A_G, A_{\neg G} \vdash$. But then the following resolution derivation λ

$$\frac{\begin{array}{c}(\lambda)\\ \Lambda \vdash \Theta, A_G \quad A_G, A_{\neg G} \vdash\end{array}}{A_{\neg G}, \Lambda \vdash \Theta}\ R$$

yields a representation of S.

Similar transformations can be constructed for the other connectives and inferences. Finally we obtain a resolution derivation ρ simulating ψ. ρ is a resolution derivation of the clause $A_F \vdash$. The final step uses the clause $\vdash A_F$ which is in $\gamma_{\text{struc}}(F)$ by definition and eventually yields the resolution refutation

$$\frac{\begin{array}{c}(\rho)\\ A_F \vdash \quad \vdash A_F\end{array}}{\vdash}\ R$$

which is a refutation of $\gamma_{\text{struc}}(F) \cup \mathcal{D}$ where $\mathcal{D}$ is a set of tautological clauses of the form $A_F \vdash A_F$ representing the initial sequents of ψ. Note that every resolution refutation can be transformed into another (even shorter) one which is tautology-free (see e.g. [30]); therefore we can get rid of the set $\mathcal{D}$ and eventually obtain a

refutation ρ' of $\gamma_{\text{struc}}(F)$ alone. Note that the whole simulation is *polynomial*: the length of the resolution proof (counting the number of occurrences of formulas in a proof) is linear in the length of the **LK**-proof; the symbolic length can be quadratic due to the introduction of Skolem terms.

With γ_{struc} we can not only simulate the cut-free **LK** but also **LK** with *analytic* cut (a cut is called analytic if the cut formula occurs as subformula in the end sequent). Indeed if A is the cut formula in an inference of an **LK**-proof ψ of $F \vdash$ and G is subformula of F we have an atom $A_G(\bar{x})$ and the defining formula for G; thus the cut with G can be simulated by resolution with $A_G(\bar{t})$. If we add arbitrary formulas G and construct $\gamma_{\text{struc}}(G)$ we can even polynomially simulate full **LK**. This stronger form of structural transformation is called the *extension principle*; it was introduced (for first-order logic) by E. Eder [12]. Thus resolution + extension is capable of simulating virtually every logic calculus in an easy way: once we have simulated **LK** we can use Gentzen's transformations in [17] to simulate natural deduction and Hilbert type calculi.

§4. Resolution Refinements. As resolution is a calculus for proof search, a main direction of improvement was (since the very beginning of resolution theorem proving) the reduction of possible resolution deductions under preservation of completeness. Generally we call any restriction of resolution deduction a *resolution refinement.* Restrictions of the resolution principle did not only improve the efficiency of theorem provers but also lead to the development of logic programming and decision procedures; the latter application area will be treated in Section 5. In many applications of resolution we are not interested in the deductions themselves but rather in the deductive closure (i.e. in the set of derivable clauses). This aspect of deduction is best described by *resolution operators.*

DEFINITION 4.1 (resolution operator). Let $\mathcal{C}$ be a set of clauses and $\mathcal{R}es(\mathcal{C})$ denote the set of all resolvents definable from $\mathcal{C}$ and subjected to a normalization under a standard renaming of variables. Then we define the operator R, the *operator of unrestricted resolution* and its deductive closure R^* by

$$R(\mathcal{C}) = \mathcal{C} \cup \mathcal{R}es(\mathcal{C}), \quad R^0(\mathcal{C}) = \mathcal{C},$$
$$R^{i+1}(\mathcal{C}) = R(R^i(\mathcal{C})), \quad R^*(\mathcal{C}) = \bigcup_{i \in \mathbb{N}} R^i(\mathcal{C}). \sharp$$

Note that, by the completeness of the resolution principle, we have $\vdash \in R^*(\mathcal{C})$ for unsatisfiable sets of clauses $\mathcal{C}$. By the undecidability of clause logic $R^*(\mathcal{C})$ must be infinite for some satisfiable sets of clauses $\mathcal{C}$.

Refinement operators can be defined in a similar way; we only have to replace $\mathcal{R}es$ by another operator.

DEFINITION 4.2 (refinement operator). A *resolution refinement operator* R_x is a mapping from sets of clauses to sets of clauses defined in the following way: There exists a (one-step) operator ϱ_x s.t.

- $R_x(\mathcal{C}) = \mathcal{C} \cup \varrho_x(\mathcal{C})$,
- ϱ_x is recursive,
- $\varrho_x(\mathcal{C})$ is a finite subset of $R^*(\mathcal{C})$.

Again the deductive closure is defined as

$$R_x^0(\mathcal{C}) = \mathcal{C},\ R_x^{i+1}(\mathcal{C}) = R_x(R_x^i(\mathcal{C})),$$
$$R_x^*(\mathcal{C}) = \bigcup_{i\in\mathbb{N}} R_x^i(\mathcal{C}).\ \sharp$$

The requirement $\varrho_x(\mathcal{C}) \subseteq R^*(\mathcal{C})$, instead of $\rho_x(\mathcal{C}) \subseteq \mathcal{R}es(\mathcal{C})$, serves the purpose to encompass methods of macro-inference (several resolution steps may be considered as primitive inference step). Such a principle is hyperresolution which will be described in this section. Not all refinements can be formulated within the framework of operators; indeed linear refinements are defined via the restriction of the resolution tree, which cannot be formalized by operators (for details see [30]).

4.1. Ordering Refinements. The use of ordering refinements is based on the idea to keep resolvents "small". This does not only lead to smaller clauses during deduction but also to an improved termination behavior of operators. The most common ordering principle of this type is *atom ordering*; in this restriction no resolvents are admitted which contain atoms more complex than the resolved one.

DEFINITION 4.3. An A-ordering (atom ordering) $\prec$ is a binary relation on atoms with the following properties:

A.1 $\prec$ is irreflexive,
A.2 $\prec$ is transitive,
A.3 For all atoms A, B and for all substitutions θ: $A \prec B$ implies $A\theta \prec B\theta$. $\sharp$

The property A.3 guarantees the lifting property for A-orderings which is vital to completeness.

EXAMPLE 4.1. Let A and B be two arbitrary atoms. We define the *depth ordering* $\prec_d$ by

$A \prec_d B$ iff
d1. $\tau(A) < \tau(B)$,
d2. $\mathrm{var}(A) \subseteq \mathrm{var}(B)$ and $\tau_{\max}(x, A) < \tau_{\max}(x, B)$ for $x \in \mathrm{var}(A)$.

where τ is the term depth and $\tau_{\max}(x, A)$ the maximal depth of the variable x in A.

It is easy to see that the depth ordering $\prec_d$ fulfils A.1,A.2,A.3 and thus is indeed an A-ordering. Note that by condition d1. alone this would not be the case:

Assume that $P(x) \prec_d P(f(a))$. Then by A.3 (using $\theta = \{x \leftarrow f(a)\}$) we have $P(x)\theta \prec_d P(f(a))\theta$, i.e. $P(f(a)) \prec_d P(f(a))$. But the last relation contradicts A.1. Generally $A \prec B$ is impossible for unifiable atoms A and B.

Some further examples for the $\prec_d$-relation:
$P(x, y) \prec_d P(f(x), f(y))$, $P(x, x) \prec_d Q(g(x, y))$,
$Q(x) \not\prec_d Q(f(a))$: d2. is violated,
$Q(f(a)) \not\prec_d Q(x)$: d1. is violated. $\sharp$

Another A-ordering which will turn out useful in the next section is comparing rather the size than the depth of terms.

EXAMPLE 4.2. We define an ordering by first comparing functional terms:

two functional terms are called *similar* if $s = g(r_1, \ldots, r_n)$ and $t = f(w_1, \ldots, w_n)$ and $\{r_1, \ldots, r_n\} = \{w_1, \ldots, w_n\}$. Similar terms may have different top symbols but their arguments are equal under permutations.

A functional term s *dominates* a functional term t if t has less arguments than s or more formally

- $s = f(r_1, \ldots, r_n)$, $t = g(w_1, \ldots, w_m)$,
- $n > m$,
- $\{w_1, \ldots, w_m\} \subseteq \{r_1, \ldots, r_n\}$.

The concepts above can be used in defining the term ordering $\prec_1$, where $s \prec_1 t$ iff

- s properly occurs in t or
- t dominates s or
- t contains a proper subterm which dominates s or is similar to s.

Some examples for $\prec_1$:
$f(x,y) \prec_1 h(x,y,z)$: the second term dominates the first one.
$f(x,y) \not\prec_1 g(x,y)$: the terms are similar.
$g(y,x) \prec_1 g(f(x,y),x)$: the second term contains a proper subterm which is similar to the first term.

The ordering $\prec_1$ can easily be extended to an atom ordering $\prec_2$ defined by:
$A \prec_2 B$ iff there exists an argument t of B such that for all arguments s of A we have $s \prec_1 t$.

Some examples for $\prec_2$:
$P(x,y) \prec_2 Q(f(x,y))$: by $x \prec_1 f(x,y)$ and $y \prec_1 f(x,y)$.
$P(x,z) \not\prec_2 Q(f(x,y))$: by $z \not\prec_1 f(x,y)$. ♯

We define the A-ordering refinement via $\varrho_\prec(\mathcal{C})$, which describes the set of all $\prec$-resolvents in $\mathcal{C}$. The *resolved atom* of a resolution is the atom cut out by resolution (after application of the m.g.u.s).

DEFINITION 4.4 (ordered resolution). Let $\mathcal{C}$ be a set of clauses and $\prec$ be an A-ordering. We define $C \in \varrho_\prec(\mathcal{C})$ iff $C \in \mathcal{R}es(\mathcal{C})$ and for no atom B in C: $A \prec B$, where A is the resolved atom of the corresponding resolution.

Remark: Definition 4.4 restricts resolvents in a way that only maximal atoms in a clause may be resolved. But note that the ordering has to be considered after application of the m.g.u. (this ordering principle is called *aposteriori ordering*). ♯

EXAMPLE 4.3. Let $\prec_d$ be the ordering from Example 4.1 and

$$\mathcal{C} = \{\vdash P(a);\ P(x) \vdash R(f(x));\ R(y) \vdash R(f(y));\ R(f^2(a)) \vdash\}.$$

Then

$$R_{\prec_d}(\mathcal{C}) = \mathcal{C} \cup \{R(f(a)) \vdash,\ P(f(a)) \vdash\},$$
$$R^2_{\prec_d}(\mathcal{C}) = R_{\prec_d}(\mathcal{C}) \cup \{R(a) \vdash,\ P(a) \vdash\},$$
$$R^3_{\prec_d}(\mathcal{C}) = R^*_{\prec_d}(\mathcal{C}) = R^2_{\prec_d}(\mathcal{C}) \cup \{\vdash\}.$$

Note that there are more clauses in $R^*(\mathcal{C})$. E.g. $\vdash R(f(a)) \in \mathcal{R}es(\mathcal{C})$, but here $P(a)$ is the resolved atom and $P(a) \prec_d R(f(a))$, thus $\vdash R(f(a)) \notin \varrho_{\prec_d}(\mathcal{C})$. Similarly $P(x_1) \vdash R(f^2(x_1)) \in \mathcal{R}es(\mathcal{C}) - \varrho_{\prec_d}(\mathcal{C})$, as the resolved atom is $R(f(x_1))$ and $R(f(x_1)) \prec_d R(f^2(x_1))$. ♯

Kowalski and Hayes have shown in [28] that $R_\prec$ is complete for any A-ordering $\prec$, i.e. $\vdash \in R^*_\prec(\mathcal{C})$ whenever $\mathcal{C}$ is unsatisfiable. We will show in Section 5 that completeness and good termination properties make ordering refinements the ideal tool for resolution decision procedures.

4.2. Hyperresolution. While ordered resolution uses the complexity of atoms to restrict resolution, hyperresolution concentrates on the sign of clauses. We call a clause *positive* if it is of the form $\vdash A_1, \ldots, A_n$. Roughly speaking hyperresolution is the deduction principle where only positive clauses (and $\vdash$) are derivable; this is only possible if one-step resolution is replaced by many-step inferences. The following example motivates this principle of macro-inference:

EXAMPLE 4.4. Let $\mathcal{C}$ be the set of clauses $\{C_1, C_2, C_3, C_4\}$ for

$\{C_1 = \vdash P(a,b), C_2 = \vdash P(b,a), C_3 = P(x,y), P(y,z) \vdash P(x,z), C_4 = P(a,a) \vdash\}$.

Then in the following resolution refutation one of the resolving clauses is always positive:

$$\dfrac{\dfrac{\dfrac{\vdash P(a,b) \quad \dfrac{\vdash P(b,a) \quad P(x,y), P(y,z) \vdash P(x,z)}{P(x,b) \vdash P(x,a)}\, R}{\vdash P(a,a)}\, R \qquad P(a,a) \vdash}{\vdash}\, R$$

The clause $P(x,b) \vdash P(x,a)$ can be interpreted an intermediary result leading to the clause $\vdash P(a,a)$. So we say that $\vdash P(a,a)$ is a *hyperresolvent* of the clash sequence $(C_3; C_1, C_2)$. In this sense the only "macro-resolvents" are $\vdash P(a,a)$ and $\vdash$. Note that C_3 may not be resolved with C_4 as none of them is positive.

DEFINITION 4.5. Let C be a nonpositive clause and $D_1, \ldots, D_n$ be positive clauses; then $S\colon (C; D_1, \ldots, D_n)$ is called a clash sequence. Let $C_0 = C$ and $C_{i+1} \in \mathcal{R}es(\{C_i, D_{i+1}\})$ for $i = 1, \ldots, n-1$. If C_n is defined and positive then it is called a *hyperresolvent* of S. We define the set of all hyperresolvents from a set of clauses $\mathcal{C}$ as $\varrho_H(\mathcal{C})$. The corresponding resolution operator R_H is called the operator of hyperresolution.

Remark: The operator R_H plays an important role in logic programming. In particular R_H coincides with the operator T_P (P being a logic program) in Horn logic; T_P defines the least fixed point of a logic program in the declarative semantics [31]. ♯

Hyperresolution was shown complete by J.A. Robinson in [36]. The construction of hyperresolvents can be subjected to various additional restrictions, like restriction of factoring to positive clauses and strict ordering of nonpositive clauses (see [30]).

If we take $\mathcal{C}$ as in Example 4.4 then $R^*_H(\mathcal{C}) = \mathcal{C} \cup \{\vdash P(a,a), \vdash\}$ and all produced clauses contain at most one atom. Indeed, on *Horn logic*, R_H produces only positive unit clauses and (possibly) $\vdash$ (a clause is Horn if it is of the form

$A_1, \ldots, A_n \vdash$ or $A_1, \ldots, A_n \vdash B$ for $n \geq 0$). On satisfiable sets of Horn clauses R_H can be interpreted as a *model builder:* If $R_H(\mathcal{C}) = \mathcal{C}$ and $\vdash \notin \mathcal{C}$ then the positive clauses in $R_H(\mathcal{C})$ represent a (minimal) Herbrand model of $\mathcal{C}$ (for a proof see e.g. [30]).

EXAMPLE 4.5. Let

$$\mathcal{C} = \{\vdash P(a);\ P(x) \vdash P(f(x));\ P(b) \vdash\}.$$

Then $\mathcal{C}$ is satisfiable and

$$R_H^*(\mathcal{C}) = \mathcal{C} \cup \{\vdash P(f^n(a)) | n \geq 1\}.$$

Note that $R_H^*(\mathcal{C})$ is a fixed point of the operator R_H and $\vdash \notin R_H^*(\mathcal{C})$.

The set of positive clauses in $R_H^*(\mathcal{C})$ is just $\mathcal{A}$: $\{\vdash P(f^n(a)) | n \geq 0\}$. The corresponding atoms define a Herbrand model Γ where a ground atom A is true in Γ iff $\vdash A \in \mathcal{A}$. Note that $P(f^n(b))$ is false in Γ for all $n \geq 0$.

Unfortunately $R_H^*(\mathcal{C})$ is infinite and thus this model is not produced in finitely many steps. If we apply $R_{\prec_d}$ to $\mathcal{C}$ then it is easily verified that $R_{\prec_d}^*(\mathcal{C}) = \mathcal{C}$; this gives us the answer that $\mathcal{C}$ is satisfiable, but we do not have any information about models.

§5. Resolution and the Decision Problem.

5.1. The Decision Problem. The decision problem (or the "Entscheidungsproblem") of first-order logic can be traced back into the early years of the 20th century. Around 1920 Hilbert formulated the problem to find an algorithm which decides the validity of formulas in first-order predicate logic (see e.g. [24]). He called this decision problem the *fundamental problem of mathematical logic.*

Between 1920 and 1930 a positive solution of the decision problem seemed to be merely a question of mathematical invention. Indeed some progress was achieved soon as decidable subclasses of predicate logic were found. The decision algorithms provided for these classes were clearly effective in any intuitive sense of the word (note that before publication of Turing's landmark paper, no formal concept of algorithm was available). One of the first results (achieved even before the general problem was formulated by Hilbert) was the decidability of the monadic class [32] (i.e. the class of first order formulas containing only unary predicate– and no function symbols). In the same paper Löwenheim proved that dyadic logic (i.e. the subclass where all predicate symbols are binary) is a *reduction class*, i.e. a class of first-order formulas effectively encoding full predicate logic. In the time between world war I and world war II prominent logicians attacked this problem. The initial strategy (probably) was to enlarge the decidable classes and to "shrink" the reduction classes till they eventually meet at some point (the outcome would have been the decidability of first-order logic). But in 1936 A. Church succeeded to prove the undecidability of first-order logic and thus the *unsolvability* of the (general) decision problem [10]. An immediate consequence of Church's result was the undecidability of all reduction classes. Despite this negative result, the interest in the decision problem was kept alive, the focus shifting to the exploration of the borderline between decidable and undecidable classes.

In more recent research on the decision problem the satisfiability problem instead of the validity problem is investigated. Basically this is just a matter of taste as A is valid iff $\neg A$ is unsatisfiable.

Above we mentioned the monadic and the dyadic classes which are characterized by the arity of predicate symbols. Another type of syntax restriction concerns the quantifier prefix of prenex formulas. Some of the prenex classes shown decidable (i.e. the satisfiability problem of these classes was proved decidable) before publication of Church's result are: $\forall\exists^*$ (the *Ackermann class* [1]), $\forall\forall\exists^*$ (the *Gödel class* [21]), and $\exists^*\forall^*$ (the *Bernays–Schönfinkel class* [7]).

Slight changes in the prefixes above lead to undecidable classes, e.g. $\forall\exists\forall$ and $\forall\forall\forall\exists$ define prenex classes with undecidable satisfiability problems. For a thorough treatment of the decision problem as a whole see [9].

The methods applied to prove decidability of classes are at least as interesting as the classes themselves. In particular, the decidability of the Bernays–Schönfinkel class can be proved via the finite–model–property of this class (i.e. there exists a finite model iff there exists a model at all). A class enjoying this property is called *finitely controllable*. Most of the original proofs of decidability for the classes mentioned above were based on the finite–model–property. In fact the set of all first-order formulas having finite models is recursively enumerable. Thus in performing search for a refutation and for a finite model in parallel, we obtain a decision procedure. Clearly these model-theoretic methods were designed to *prove* decidability rather than to give efficient decision algorithms. In fact, the algorithms extracted from this method are based on exhaustive search and hardly are candidates for an even modest *calculus ratiocinator*. It turned out that the satisfiability problem of decidable classes can be handled by *proof theoretic* means on a larger scale. The most appropriate candidate for such a proof theoretic approach is *resolution*.

5.2. Resolution Decision procedures. Suppose that we start a theorem prover (i.e. a complete resolution refinement R_x) on a set of clauses $\mathcal{C}$ which may be satisfiable or unsatisfiable. Obviously there are three possibilities:

1. R_x terminates on $\mathcal{C}$ and refutes $\mathcal{C}$.
   Because R_x is correct and $\vdash \in R_x(\mathcal{C})$ we know that $\mathcal{C}$ is unsatisfiable.
2. R_x terminates on $\mathcal{C}$ without deriving $\vdash$.
   By the completeness of R_x $\mathcal{C}$ must be satisfiable.
3. R_x does not terminate on $\mathcal{C}$:
   In this case $R_x^*(\mathcal{C})$ (the set of all clauses derivable by R_x from $\mathcal{C}$) is infinite and $\vdash \notin R_x^*(\mathcal{C})$ (we assume that the production of new clauses is stopped as soon as $\vdash$ is derived). Like in case 2) $\mathcal{C}$ is satisfiable, but we cannot detect this property just by computing $R_x^*(\mathcal{C})$.

As clause logic – being a reduction class of first-order logic – is undecidable we know that for every complete refinement operator R_x there must exist a (finite) set of clauses $\mathcal{C}$ s.t. $R_x^*(\mathcal{C})$ is *infinite*. That means it is, in principle, impossible to avoid nontermination on all sets of clauses. Let us investigate this point in somewhat more detail:

Let F be a sentence of (first-order) predicate logic. Using a normal form transformation we can transform F into a sat–equivalent set of clauses $\mathcal{C}$. By

the arguments above R_x must be nonterminating on some finite, satisfiable sets of clauses; therefore case 3) mentioned above cannot be avoided in general. But avoiding case 3) for specific *subclasses* of clause logic is precisely the principle of *resolution decision procedures!* It leads to the following method for proving decidability of (the satisfiability problem of) a first-order class Γ:

a: Transform the formulas in Γ into their sat-equivalent clause forms (resulting in a clausal class Γ' corresponding to Γ).

b: Find a complete resolution refinement which terminates on Γ'.

This principle is quite general and can be applied with other calculi than resolution and other normal forms than clause form. In 1968 S.Y. Maslov proved the decidability of the so called K–class (a decision class properly containing the Gödel class) using a similar approach; it is based on the so called inverse method, which is a resolution–type method formulated within the framework of a sequent calculus [34].

In the same spirit as Maslov, but on the basis of the resolution calculus, Joyner showed in his thesis [26] that resolution theorem provers can be used as decision procedures for some classical prenex classes (e.g. the Ackermann– and the Gödel class). His idea to find complete resolution refinements R_x which terminate on clause classes corresponding to prenex classes, lead to the general methodology of resolution decision procedures developed in [14].

Even unrestricted resolution can be useful in proving the decidability of first-order classes. An example is Herbrand's class **H**C [23], where **H**C is the class of all first-order formulas of the form

$$(Q_1x_1)\cdots(Q_nx_n)(L_1 \wedge \cdots \wedge L_m)$$

for function-free literals $L_1, \ldots, L_m$.

THEOREM 5.1. *The satisfiability problem of* ***H****C is decidable.*

PROOF. Skolemization of formulas in **H**C directly yields a clausal class **H**C' consisting of finite sets of unit clauses. Now let

$$\mathcal{C} = \{\vdash A_1, \ldots, \vdash A_m, B_1 \vdash, \ldots, B_n \vdash\}$$

for some atoms $A_1, \ldots, A_m, B_1, \ldots, B_n$; then, clearly, $R^*(\mathcal{C}) = \mathcal{C}$ or $R^*(\mathcal{C}) = \mathcal{C} \cup \{\vdash\}$. Thus for all $\mathcal{C} \in$ **H**C' $R^*(\mathcal{C})$ is finite and, consequently, unrestricted resolution decides **H**C' and thus **H**C. ⊣

The proof of the theorem above shows the power of the unification principle which renders an originally complicated problem trivial.

However unrestricted resolution fails on very simple (satisfiable) sets of clauses:

EXAMPLE 5.1. Let $F = P(a) \wedge (\forall x)(P(x) \leftrightarrow \neg P(f(x)))$. Then F is satisfiable and (via the standard transformation) yields the set of clauses

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

As $\mathcal{C}$ is satisfiable $\vdash \notin R^*(\mathcal{C})$. Moreover, $\vdash P(x), P(f^{2n+1}(x)) \in R^*(\mathcal{C})$ for all $n \in \mathbb{N}$, thus $R^*(\mathcal{C})$ is infinite and resolution does not terminate on $\mathcal{C}$.

Applying the A-ordering refinement $R_{\prec_d}$ (see Example 4.1) to the set of clauses $\mathcal{C}$ in Example 5.1 we just obtain $R^*_{\prec_d}(\mathcal{C}) = \mathcal{C} \cup \{P(x) \vdash P(x)\}$. So $\vdash \notin R^*_{\prec_d}(\mathcal{C})$ and, by the completeness of $R_{\prec_d}$, we conclude that $\mathcal{C}$ is satisfiable. ♯

$R_{\prec_d}$ does not only work in Example 5.1 but gives a decision procedure of some well-known first-order classes, in particular of the Ackermann class:

Let us consider a formula of the form

$$F\colon\ (\exists x_1)\cdots(\exists x_m)(\forall y)(\exists z_1)\cdots(\exists z_k)M(x_1,\ldots,x_m,z_1,\ldots,z_k,y)$$

where $k,m \geq 0$ and M is a function– and constant free matrix. By skolemizing F we obtain a formula

$$F'\colon\ (\forall y)M(c_1,\ldots,c_m,f_1(y),\ldots,f_k(y),y),$$

where $c_1,\ldots,c_m$ are (different) constant symbols and $f_1,\ldots,f_k$ are (different) one–place function symbols. In transforming the matrix of F' into conjunctive normal form (via the standard transformation) we obtain a set of clauses $\mathcal{C}$ fulfilling the following properties:

1) All clauses contain at most one variable,
2) all function symbols occurring in $\mathcal{C}$ are unary,
3) the term depth of all clauses C in $\mathcal{C}$ is ≤ 1.

In particular all sets of clauses obtained from the Ackermann class belong to the more general *one–variable class* introduced in the following definition:

DEFINITION 5.1. The class **VARI** (also called the one–variable class) is the set of all finite sets of clauses $\mathcal{C}$ fulfilling the following condition: All $C \in \mathcal{C}$ contain at most one variable.

We have seen that the clause forms of the formulas of the Ackermann class belong to **VARI**; on the other hand there exist sets of clauses in **VARI** which cannot be obtained by transforming Ackermann formulas into clause form. Clearly a decision procedure for **VARI** yields a decision procedure for Ackermann's class.

THEOREM 5.2. *The class **VARI** can be decided by the A–ordering $<_d$ or more exactly: $R^*_{\prec_d}(\mathcal{C})$ is finite for all $\mathcal{C} \in$ **VARI**.*

PROOF. In [30]. In fact a more general result is actually proven as **VARI** is not invariant under $R_{\prec_d}$. So **VARI** is extended to a class **K** which is invariant under $R_{\prec_d}$ (even under unrestricted resolution R). Then it is shown that $R_{\prec_d}$ terminates on **K**. ⊣

Remark: The termination of $R_{\prec_d}$ on **VARI** only yields its decidability because $R_{\prec_d}$ is complete! But the completeness of A-ordering refinements is a general result in automated deduction. Also the proof that $\prec_d$ is indeed an A-ordering is quite simple. Thus the the major complexity of the proof lies in showing termination. ♯

It is not just **VARI** and the Ackermann class which can be decided by ordered resolution. In fact there is a broad range of traditional and new classes which can be decided by some ordering refinement (see [14]). Another example is the monadic class which can be decided via the A-ordering $\prec_2$ defined in Example 4.2; we only have to apply $\mathcal{R}_{\prec_2}$ to the clausal representation **MON'** of the monadic class:

DEFINITION 5.2. Let **MON** be the monadic class. Then **MON'** is the class of all sets of clauses $\mathcal{C}$ obtained via the standard clause form transformation from **MON**.

THEOREM 5.3. *$R_{\prec_2}$ decides **MON'**, i.e. for all $\mathcal{C} \in$ **MON'** $R^*_{\prec_2}(\mathcal{C})$ is finite.*

PROOF. in [27] and [30]. ⊣

EXAMPLE 5.2. Let F be the monadic formula

$$(\exists x_1)(\exists x_2)(\forall y_1)(\forall y_2)(\exists x_3)(P(x_1) \wedge (\neg P(y_1) \vee P(x_3)) \wedge \neg P(x_2)).$$

Then the corresponding set of clauses is

$$\mathcal{C}\colon\ \{\vdash P(a);\ P(y_1) \vdash P(f(y_1, y_2));\ P(b) \vdash\}.$$

It is easy to see that $R^*_{\prec_2}(\mathcal{C}) = \mathcal{C}$, which gives a trivial proof of the satisfiability of $\mathcal{C}$. On the other hand unrestricted resolution does not terminate on $\mathcal{C}$. ♯

Note that *termination* of resolution decision procedures is the key for proving decidability of classes. But the termination of resolution refinements is much less dependent on the signature of problems than the applicability of model theoretic methods. Therefore resolution decision theory leads to several syntactic extensions of the traditional first-order decision classes [14]. Concerning the decision problem itself resolution refinements thus offer an elegant tool to prove decidability of first-order classes; therefore they can be considered as a *general theoretical methodology* in mathematical logic. Moreover they are useful for solving *concrete satisfiability problems* and are most essential to the efficiency of theorem proving programs.

§6. Cut-elimination by Resolution. Cut-elimination is one of the most important techniques of proof transformation. Roughly speaking, eliminating cuts from a proof generates a new proof without lemmas, which essentially consists of the syntactic material of the proven theorem; i.e. we obtain proofs fulfilling the *subformula property*. Traditionally cut-elimination served the purpose to show consistency of calculi and thus played a central role in metamathematics. In this traditional context the aim is to define just a *constructive method* for eliminating cuts, its actual use as an algorithm is of minor importance. But in more recent time J.Y. Girard demonstrated that cut-elimination on real mathematical proofs may produce valuable mathematical information [19]. In particular he showed, how a proof of van der Waerden's theorem using concepts of topology can be transformed into an elementary combinatorial proof by means of cut-elimination. Thus it makes sense to investigate cut-elimination for *single proofs*, in order to obtain additional mathematical information on a theorem. But then an *algorithmic* approach to cut-elimination becomes more interesting.

The standard method of cut-elimination is that of Gentzen defined in his famous "Hauptsatz" [17]. The method is essentially a nondeterministic algorithm extracted from his (constructive) proof. Its characteristic feature is a *stepwise reduction* of cut complexity. In this reduction the cut formulas are decomposed w.r.t. their outermost logical operator (leading to a decrease of the logical complexity). Moreover, the cut formulas to be eliminated must be rendered principal formulas of inferences by adequate proof transformations (leading to a reduction of the rank). Despite its elegance, Gentzen's method is algorithmically very

costly (of course we cannot blame Gentzen, as his aim was not to define an algorithm!). The reason is that the method is largely independent of the derivations and of the *inner* structure of the cut formulas.

The availability of resolution theorem proving and the fact that resolution is in some sense a "subcalculus" of **LK** (see Section 3) makes resolution a natural candidate in the investigation of cut-elimination. In this section we will informally present an algorithmic method of *cut-elimination by resolution*, an exact and exhaustive treatment can be found in [5]. The resolution method substantially differs from Gentzen's one. In the first step a set of clauses is generated from the derivations of the cut formulas. These sets of clauses are always unsatisfiable and thus have a resolution refutation. The construction of the resolution refutation is the second step of the procedure. Note that this step represents a direct application of automated theorem proving. The resolution refutation obtained from the theorem prover then serves as a *skeleton* of an **LK**-proof with only atomic cuts; this **LK**-proof is obtained by filling the skeleton with parts of the original **LK**-proof (actually with proof projections). The last step consists of the elimination of atomic cuts.

Although cut-elimination gave the original motivation to the development of the resolution method, the approach is far more general: indeed, the elimination of cuts appears as a special case of redundancy-elimination in **LK**-proofs. E.g. it suffices that the left cut formula logically implies the right one; they need not be syntactically equal. In fact the resolution method is largely a *semantic* one. Furthermore the method can be generalized to a method of occurrence elimination in **LK**-proofs which sheds more light on the role of redundancy in proofs.

In the first step we reduce cut-elimination to formula-elimination: that means we transform a proof φ with cuts into a cut-free proof ψ of an extended end-sequent; this transformation (unlike "real" cut-elimination) is harmless in the sense that the time complexity is linear in the size of φ.

DEFINITION 6.1. We define a mapping T_{cut} which transforms an **LK**-proof ψ of a sequent $S\colon \Gamma \vdash \Delta$ with cut formulas $A_1, \dots A_n$ into an **LK**-proof ψ^* of

$$\forall(A_1 \to A_1) \wedge \dots \wedge \forall(A_n \to A_n), \Gamma \vdash \Delta$$

in the following way: Take an uppermost cut and its derivation χ:

$$\dfrac{\begin{matrix}(\chi_1) \\ \Pi_1 \vdash \Lambda_1, A\end{matrix} \quad \begin{matrix}(\chi_2) \\ A, \Pi_2 \vdash \Lambda_2\end{matrix}}{\Pi_1, \Pi_2 \vdash \Lambda_1, \Lambda_2}\ cut$$

occurring in ψ and replace it by χ'

$$\dfrac{\begin{matrix}(\chi_1) \\ \Pi_1 \vdash \Lambda_1, A\end{matrix} \quad \begin{matrix}(\chi_2) \\ A, \Pi_2 \vdash \Lambda_2\end{matrix}}{A \to A, \Pi_1, \Pi_2 \vdash \Lambda_1, \Lambda_2} \to\colon l$$

Afterwards apply $\forall : l$-inferences to the end-sequent of χ' on the free variables in $A \to A$ resulting in a proof χ'' of $\forall(A \to A), \Pi_1, \Pi_2 \vdash \Lambda_1, \Lambda_2$. Iterate the procedure on the next uppermost cuts till all cuts are eliminated and keep all

other inferences unchanged. The result is a proof ψ' of the sequent S' :

$$\forall(A_1 \to A_1), \ldots \forall(A_n \to A_n), \Gamma \vdash \Delta.$$

Finally ψ^* is obtained by contractions and $\wedge : l$.

We call the new sequent S' : the *cut-extension* of S w.r.t. ψ. ♯

It is easy to see that $T_{cut}(\psi)$ is indeed a cut-free proof of the cut-extension of S w.r.t. ψ. The only nontrivial point is the preservation of the eigenvariable conditions.

After transformation of the proof ψ of S to $T_{cut}(\psi)$ of the cut-extension S' the problem of cut-elimination in ψ can be reduced to the construction of a cut-free proof of S from $T_{cut}(\psi)$. The new problem then consists in the elimination of the formula B: $\forall(A_1 \to A_1) \wedge \ldots \wedge \forall(A_n \to A_n)$ on the left-hand-side of the end-sequent. For technical reasons we assume that the end sequent of S is skolemized. Note that **LK**-proofs can be skolemized by a polynomial transformation defined in [4].

The first step in the formula-elimination procedure consists in the construction of a set of clauses. This set corresponds to a left occurrence of a (valid) formula in the end-sequent of an **LK**-proof. Roughly speaking we trace the derivation of the formula B (encoding the cut formulas of the original proof) back to the initial sequents. In the initial sequents we separate the parts which are ancestors of B and obtain a set of clauses $\mathcal{C}$ where each $C \in \mathcal{C}$ is of the form $\vdash$, $A \vdash$, $\vdash A$ or $A \vdash A$. Going down in the proof we look whether the corresponding inference works on ancestors of B or not. In the first case we have to subject the sets of clauses to union, in the second one to a product. The formal definition is given below:

DEFINITION 6.2. Let ψ be a cut-free proof of S and α be an occurrence of a formula in S. We define the set of *characteristic clauses* $\mathrm{CL}(\psi, \alpha)$ inductively: Let η be an occurrence of a sequent S' in ψ; by $anc(\eta, \alpha)$ we denote the subsequent S'' of S' which consists exactly of the formulas with occurrences being ancestors of the occurrence α in S. Let η be the occurrence of an initial sequent $A \vdash A$ in ψ and η_1 (η_2) be the left (right) occurrence of A in $A \vdash A$. If neither η_1 nor η_2 is an ancestor of α then $\mathcal{C}_\eta = \{\vdash\}$; If both η_1 and η_2 are ancestors of α then $\mathcal{C}_\eta = \emptyset$. Otherwise (exactly one of η_1, η_2 is ancestor of α) $\mathcal{C}_\eta = \{anc(\eta, \alpha)\}$, i.e. $\mathcal{C}_\eta = \{A \vdash\}$ if η_1 is ancestor of α and $\mathcal{C}_\eta = \{\vdash A\}$ if η_2 is ancestor of α.

Let us assume that the clause sets $\mathcal{C}_\lambda$ are already constructed for all sequent–occurrences λ in ψ with $depth(\lambda) \leq k$ (where the depth of an occurrence λ is the length of the path in the proof tree from the root to λ).

Now let λ be an occurrence with $depth(\lambda) = k+1$. We distinguish the following cases:

a: λ is the consequent of μ, i.e. a unary rule applied to μ gives λ. Here we simply define $\mathcal{C}_\lambda = \mathcal{C}_\mu$.

b: λ is the consequent of μ_1 and μ_2, i.e. a binary rule X applied to μ_1 and μ_2 gives λ.

b1: The auxiliary formulas of X are ancestors of α, i.e. the formulas occur in $anc(\mu_1, \alpha), anc(\mu_2, \alpha)$. Then $\mathcal{C}_\lambda = \mathcal{C}_{\mu_1} \cup \mathcal{C}_{\mu_2}$.

b2: The auxiliary formulas of X are not ancestors of α. In this case we define $\mathcal{C}_\lambda = \mathcal{C}_{\mu_1} \otimes \mathcal{C}_{\mu_2}$ where

$\{\bar{P}_1 \vdash \bar{Q}_1, \ldots \bar{P}_m \vdash \bar{Q}_m\} \otimes \{\bar{R}_1 \vdash \bar{T}_1, \ldots \bar{R}_n \vdash \bar{T}_n\} =$

$\{\bar{P}_i, \bar{R}_j \vdash \bar{Q}_i, \bar{T}_j \mid i \leq m, j \leq n\}$

Finally $\mathrm{CL}(\psi, \alpha)$ is set to $\mathcal{C}_\nu$ where ν is the occurrence of the end-sequent. Note that α is an occurrence in ν and its own ancestor. ♯

EXAMPLE 6.1. Let ψ be the proof (for u, v free variables, a a constant symbol)

$$\dfrac{\begin{matrix}\psi_1 & \psi_2\end{matrix}}{(\forall x)(P(x) \to Q(x)) \vdash (\exists y)(P(a) \to Q(y))}\ cut$$

where ψ_1 is the **LK**-proof:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{P(u)^* \vdash P(u) \quad Q(u) \vdash Q(u)^*}{P(u)^*, P(u) \to Q(u) \vdash Q(u)^*} \to : l}{P(u) \to Q(u) \vdash (P(u) \to Q(u))^*} \to : r}{P(u) \to Q(u) \vdash (\exists y)(P(u) \to Q(y))^*} \exists : r}{(\forall x)(P(x) \to Q(x)) \vdash (\exists y)(P(u) \to Q(y))^*} \forall : l}{(\forall x)(P(x) \to Q(x)) \vdash (\forall x)(\exists y)(P(x) \to Q(y))^*} \forall : r$$

and ψ_2 is:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{P(a) \vdash P(a)^* \quad Q(v)^* \vdash Q(v)}{P(a), (P(a) \to Q(v))^* \vdash Q(v)} \to : l}{(P(a) \to Q(v))^* \vdash P(a) \to Q(v)} \to : r}{(P(a) \to Q(v))^* \vdash (\exists y)(P(a) \to Q(y))} \exists : r}{(\exists y)(P(a) \to Q(y))^* \vdash (\exists y)(P(a) \to Q(y))} \exists : l}{(\forall x)(\exists y)(P(x) \to Q(y))^* \vdash (\exists y)(P(a) \to Q(y))} \forall : l$$

The ancestors of the cut formula in ψ_1 and ψ_2 are marked by $*$. From ψ we construct the cut-extension ψ', where A denotes the cut formula $(\forall x)(\exists y)(P(x) \to Q(y))$ of ψ:

$$\dfrac{\begin{matrix}\psi_1 & \psi_2\end{matrix}}{A \to A, (\forall x)(P(x) \to Q(x)) \vdash (\exists y)(P(a) \to Q(y))} \to : l$$

Let α be the occurrence of $A \to A$ in the end sequent S' of ψ'. We compute the characteristic clauses $\mathrm{CL}(\psi', \alpha)$:

From the $*$-marks in the proofs ψ_1 and ψ_2 (which indicate the ancestors of α) we first get the sets of clauses corresponding to the initial sequents:

$$\mathcal{C}_1 = \{P(u) \vdash\},\ \mathcal{C}_2 = \{\vdash Q(u)\},\ \mathcal{C}_3 = \{\vdash P(a)\},\ \mathcal{C}_4 = \{Q(v) \vdash\}.$$

The first inference in ψ_1 (it is $\to : l$) takes place on nonancestors of α – the auxiliary formulas of the inference are not marked by $*$. Consequently we apply $\otimes$ and obtain the set $\mathcal{C}_{1,2} = \{P(u) \vdash Q(u)\}$. The following inferences in ψ_1 are all unary and so we obtain

$$\mathrm{CL}(\psi_1, \alpha_1) = \{P(u) \vdash Q(u)\}$$

for α_1 being the occurrence of the ancestor of α in the end-sequent of ψ_1.

The first inference in ψ_2 takes place on ancestors of α (the auxiliary formulas are $*$-ed) and we have to apply the $\cup$ on $\mathcal{C}_3, \mathcal{C}_4$. We obtain $\mathcal{C}_{3,4} = \{\vdash P(a), \quad Q(v) \vdash\}$. Like in ψ_1 all following inferences in ψ_2 are unary leaving the set of clauses unchanged. Let α_2 be the ancestor of α in the end-sequent of ψ_2. Then the corresponding set of clauses is

$$\mathrm{CL}(\psi_2, \alpha_2) = \{\vdash P(a), \quad Q(v) \vdash\}.$$

The last inference $\rightarrow: l$ in ψ' takes place on ancestors of α and we have to apply $\cup$ on $\mathcal{C}_{1,2}$ and $\mathcal{C}_{3,4}$. This eventually yields

$$\mathrm{CL}(\psi', \alpha) = \{P(u) \vdash Q(u), \quad \vdash P(a), \quad Q(v) \vdash\}. \; \sharp$$

It is easy to verify that the set of characteristic clauses $\mathrm{CL}(\psi', \alpha)$ constructed in the example above is unsatisfiable. This is not merely a coincidence, but a general principle expressed in the next proposition.

PROPOSITION 6.1. *Let ψ be a cut-free proof of the sequent S and α be a left-occurrence of a valid formula occurring in S. Then the set of clauses* $\mathrm{CL}(\psi, \alpha)$ *is unsatisfiable.*

PROOF. in [4]. Basically it is shown that $B \vdash$, for B occurring at α, (which is an unsatisfiable sequent) is derivable in **LK** from the initial axioms $\mathrm{CL}(\psi, \alpha)$. $\dashv$

Remark: The proof of Proposition 6.1 might suggest that the set of clauses $\mathrm{CL}(\psi, \alpha)$ is just a clausal normal form of the formula $\neg B$ corresponding to the sequent $B \vdash$; but this is not the case! As a simple counterexample consider the following derivation ψ:

$$\dfrac{\dfrac{Q(b) \vdash Q(b)}{\vdash Q(b) \rightarrow Q(b)} \rightarrow: r}{P(a) \rightarrow P(a) \vdash Q(b) \rightarrow Q(b)} \; w: l$$

The only initial sequent of ψ is $Q(b) \vdash Q(b)$. Neither the left- nor the right occurrence of $Q(b)$ in this sequent is an ancestor of the occurrence α of $P(a) \rightarrow P(a)$ in the end sequent. Thus the set of clauses $\mathcal{C}$ corresponding to the node of the initial sequent is $\{\vdash\}$. As there are only unary rules in ψ we finally obtain $\mathrm{CL}(\psi, \alpha) = \mathcal{C} = \{\vdash\}$. On the other hand no traditional transformation to normal form (like standard- or structural transformation) transforms the formula $\neg(P(a) \rightarrow P(a))$ into $\{\vdash\}$. In particular the standard transformation gives the set of clauses $\mathcal{D}$: $\{\vdash P(a), \; P(a) \vdash\}$. The example above illustrates that the set of clauses $\mathrm{CL}(\psi, \alpha)$ strongly depends on the derivation ψ and not only on the form of the formula on position α! We will see in the following presentation of the method that the construction of $\mathrm{CL}(\psi, \alpha)$ from the proof ψ plays a central role in the so-called proof projection (which cannot be performed on the basis of ordinary clause forms). $\sharp$

Now let $\mathrm{CL}(\psi, \alpha)$ be the (unsatisfiable) set of clauses extracted from the **LK**-proof of the extended sequent S. By the completeness of the resolution principle there exists a resolution refutation γ (in form of a tree) of the set of clauses $\mathrm{CL}(\psi, \alpha)$. γ can be transformed into a ground refutation of $\mathrm{CL}(\psi, \alpha)$:

PROPOSITION 6.2. *Let γ be a tree resolution refutation of a set of clauses $\mathcal{C}$. Then there exists a ground instance γ' of γ s.t. γ' is a tree resolution refutation of $\mathcal{C}'$ where $\mathcal{C}'$ is a set of ground instances from $\mathcal{C}$.*

PROOF. Let λ be the simultaneous most general unifier of all the resolutions in γ. Then $\gamma\lambda$ is also a resolution refutation where the resolution rule reduces to atomic cut and contractions (i.e. to a mix, see [17]). Let σ be an arbitrary ground substitution of the variables of $\gamma\lambda$; then $\gamma' : \gamma\lambda\sigma$ is the desired resolution refutation. ⊣

Remark: We call the refutation γ' defined above a ground refutation *corresponding to* γ. ♯

Now let γ' be a ground refutation corresponding to a resolution refutation γ of $\mathrm{CL}(\psi, \alpha)$. By our definition of resolution γ' can easily transformed to an **LK**-proof of $\vdash$ from $\mathcal{C}'$ with atomic cuts. Indeed, only additional contractions are necessary to simulate factoring. The resulting **LK**-proof γ' will serve as a *skeleton* of an **LK**-proof ϕ of $\Gamma \vdash \Delta$ with atomic cuts. Recall that S may be a cut-extension $B, \Gamma \vdash \Delta$ of $\Gamma \vdash \Delta$.

Thus ϕ corresponds (modulo the transformation T_{cut}) to a reduction of a proof with cuts to a proof with atomic cuts. The construction of ϕ from γ' is based on so called *projections* replacing the proof ψ of the cut-extension S by proofs $\psi[C]$ of $\bar{P}, \Gamma \vdash \Delta, \bar{Q}$ for clauses $C : \bar{P} \vdash \bar{Q}$ in $\mathcal{C}'$, where $\mathcal{C}'$ is the set of ground instances refuted by γ'. The existence of such projections of ψ w.r.t. clauses in $\mathcal{C}'$, guaranteed by the lemma below, is the most important property of the cut-elimination method based on resolution.

LEMMA 6.1. *Let ψ be a cut-free proof of a sequent $S : B, \Gamma \vdash \Delta$ s.t. $\Gamma \vdash \Delta$ is skolemized, B is valid and α is the occurrence of B in S. Let $C : \bar{P} \vdash \bar{Q}$ be a clause in $\mathrm{CL}(\psi, \alpha)$. Then there exists a cut-free proof $\psi[C]$ of $\bar{P}, \Gamma \vdash \Delta, \bar{Q}$ with $l(\psi[C]) \leq l(\psi)$ (where l denotes the length of the proof, i.e. the number of nodes occurring in the derivation tree).*

PROOF. We only give a proof sketch; a full formal proof can be found in [5].

The proof goes by induction on the depth of inference nodes in ψ. In constructing $\psi[C]$ we skip all inferences in ψ leading to the extension formula B. In the other inferences with auxiliary formulas which are not ancestors of B we select two clauses from the corresponding set of clauses and construct the corresponding projections via the induction hypothesis. We concentrate on a binary inference rule X and the following proof χ:

$$\dfrac{\begin{array}{cc}(\rho_1) & (\rho_2)\\ (\mu_1)\ \Gamma_1 \vdash \Delta_1 & (\mu_2)\ \Gamma_2 \vdash \Delta_2\end{array}}{(\lambda)\ \Gamma_1', \Gamma_2' \vdash \Delta_1', \Delta_2'}\ X$$

where μ_1, μ_2, λ are the nodes in the proof tree ψ and $\bar{P} \vdash \bar{Q}$ is a clause in $\mathcal{C}_\lambda$. We assume that the auxiliary formulas of X are not ancestors of α and that the subsequents of $\Gamma_i \vdash \Delta_i$ defined by formulas which are not ancestors of B are $\Pi_i \vdash \Lambda_i$ for $i = 1, 2$. Then, by Definition 6.2, we have $\mathcal{C}_\lambda = \mathcal{C}_{\mu_1} \otimes \mathcal{C}_{\mu_2}$. Therefore there are clauses $\bar{P}_1 \vdash \bar{Q}_1 \in \mathcal{C}_{\mu_1}$ and $\bar{P}_2 \vdash \bar{Q}_2 \in \mathcal{C}_{\mu_2}$ s.t.

$$\bar{P} \vdash \bar{Q} = \bar{P}_1, \bar{P}_2 \vdash \bar{Q}_1, \bar{Q}_2.$$

By induction hypothesis we obtain proofs ρ_1' of $\bar{P}_1, \Pi_1 \vdash \Lambda_1, \bar{Q}_1$ and ρ_2' of $\bar{P}_2, \Pi_2 \vdash \Lambda_2, \bar{Q}_2$ with $l(\rho_1') \leq l(\rho_1)$ and $l(\rho_2') \leq l(\rho_2)$. Then the projection corresponding to the node λ is χ':

$$\frac{\begin{array}{cc}(\rho_1') & (\rho_2') \\ \bar{P}_1, \Pi_1 \vdash \Lambda_1, \bar{Q}_1 & \bar{P}_2, \Pi_2 \vdash \Lambda_2, \bar{Q}_2\end{array}}{\bar{P}_1, \bar{P}_2, \Pi_1', \Pi_2' \vdash \Lambda_1', \Lambda_2', \bar{Q}_1, \bar{Q}_2}\ X$$

Clearly $l(\chi') \leq l(\chi)$. ⊣

Remark: For the projections $\psi[C]$ we need the set of clause $\mathrm{CL}(\psi, \alpha)$ as defined in Definition 6.2. Here it is important that $\mathcal{C}: \mathrm{CL}(\psi, \alpha)$ is constructed *from the proof* ψ itself! Thus $\mathcal{C}$ is not an ordinary clause form of $\neg B$ constructed from the syntax of B, but a clause form belonging to the derivation of $B \vdash$ within ψ. ♯

Once we we have constructed the projections $\psi[C]$ we can "insert" them into the resolution refutation γ. The formal procedure is defined below:

DEFINITION 6.3. Let ψ be a cut-free proof of S: $B, \Gamma \vdash \Delta$ s.t. B is valid, $\Gamma \vdash \Delta$ closed and skolemized and α the occurrence of B in S. Let γ' a ground refutation corresponding to a resolution refutation γ of $\mathrm{CL}(\psi, \alpha)$ s.t. $\gamma' = \gamma\sigma$. We define an **LK**-proof $\gamma'[\psi]$ inductively:

Let N be a leaf node in γ labelled with a clause $C\sigma$ for $C \in \mathrm{CL}(\psi, \alpha)$ and let $C\sigma = \bar{P} \vdash \bar{Q}$. To N we assign the proof $\omega_N: \psi[C]\sigma$, where $\psi[C]$ is the projection of ψ to C as defined in Lemma 6.1 By definition ω_N is a cut-free proof of the sequent $\bar{P}, \Gamma\sigma \vdash \Delta\sigma, \bar{Q}$. By assumption S is closed and thus ω_N is a cut-free proof of $\bar{P}, \Gamma \vdash \Delta, \bar{Q}$.

Assume that N is a node in γ labelled with C and with parent nodes N_1 labelled with C_1 and N_2 labelled with C_2. Then, by definition of a resolution derivation, C is a (ground) resolvent of C_1 and C_2. Therefore $C_1 = \bar{P} \vdash \bar{Q}, A^r$, $C_2 = A^s, \bar{R} \vdash \bar{T}$ and $C = \bar{P}, \bar{R} \vdash \bar{Q}, \bar{T}$ for multisets of atoms $\bar{P}, \bar{Q}, \bar{R}, \bar{T}$ and an atom A occurring r-times in C_1 and s-times in C_2

Let ω_{N_1} and ω_{N_2} be the **LK**-proofs corresponding to N_1 and N_2, respectively. Assume that ω_{N_1} is a proof of $\bar{P}, \Gamma^k \vdash \Delta^k, \bar{Q}, A^r$ and ω_{N_2} of $A^s, \bar{R}, \Gamma^l \vdash \Delta^l, \bar{T}$ for $k, l \in \mathbb{N}$. Then ω_N, the **LK**-proof corresponding to N, is defined as

$$\frac{\dfrac{\begin{array}{c}(\omega_{N_1}) \\ \bar{P}, \Gamma^k \vdash \Delta^k, \bar{Q}, A^r\end{array}}{\bar{P}, \Gamma^k \vdash \Delta^k, \bar{Q}, A}\ c: r^* \qquad \dfrac{\begin{array}{c}(\omega_{N_2}) \\ A^s, \bar{R}, \Gamma^l \vdash \Delta^l, \bar{T}\end{array}}{A, \bar{R}, \Gamma^l \vdash \Delta^l, \bar{T}}\ c: l^*}{\bar{P}, \bar{R}, \Gamma^{k+l} \vdash \Delta^{k+l}, \bar{Q}, \bar{T}}\ cut$$

Let N_r be the root node of γ'; then $\gamma'[\psi]$ is defined as ω_{N_r}. ♯

If ψ is a cut-free proof of $B, \Gamma \vdash \Delta$ then $\gamma'[\psi]$ in Definition 6.3 is a proof with atomic cuts of $\Gamma, \ldots, \Gamma \vdash \Delta, \ldots, \Delta$ (note that the clause belonging to the root node of γ' is $\vdash$). Only additional contractions are necessary for getting a proof $\hat{\gamma}[\psi]$ with atomic cuts of $\Gamma \vdash \Delta$ itself. It remains only to eliminate the atomic cuts; to this aim any cut-elimination procedure (e.g. this in [17]) does the job. The length of the proofs with atomic cuts is essentially defined by the length of γ'.

THEOREM 6.1. *Let ψ be a cut-free proof of a closed sequent $S\colon B, \Gamma \vdash \Delta$, where B is a valid formula occurring at α in S and $\Gamma \vdash \Delta$ is skolemized. Furthermore let γ' be a ground refutation which corresponds to a resolution refutation of* $\mathrm{CL}(\psi, \alpha)$ *and* $\|\gamma\| = \max\{\|C\| \mid C \text{ in } \gamma\}$. *Then there exists a proof $\hat{\gamma}[\psi]$ of $\Gamma \vdash \Delta$ with atomic cuts and $l(\hat{\gamma}[\psi]) \leq 2 \cdot l(\psi) l(\gamma)(2\|\gamma\| + 1)$.*

PROOF. see [5] ⊣

To illustrate the whole procedure described above we continue with Example 6.1.

EXAMPLE 6.2. Let ψ' be the proof of the sequent

$$S\colon\ A \to A, (\forall x)(P(x) \to Q(x)) \vdash (\exists y)(P(a) \to Q(y))$$

as defined in Example 6.1. We have shown that

$$\mathrm{CL}(\psi', \alpha) = \{P(u) \vdash Q(u),\ \ \vdash P(a),\ \ Q(v) \vdash\}$$

where α is the occurrence of $A \to A$ in S.

We first define the projections of ψ' w.r.t. clauses in $\mathrm{CL}(\psi', \alpha)$:

We start with $\psi'[C_1]$, the projection of ψ' to $C_1\colon P(u) \vdash Q(u)$:
The problem can be reduced to the construction of $\psi_1[C_1]$ because of

$$\mathrm{CL}(\psi_1, \alpha_1) = \{P(u) \vdash Q(u)\}.$$

By definition of ψ_1 and of the projection, $\psi_1[C_1]$ is a proof of

$$P(u), (\forall x)(P(x) \to Q(x)) \vdash Q(u).$$

The last inference in ψ' applies to ancestors of α and thus $\psi'[C_1]$ is defined as

$$\dfrac{\begin{matrix}(\psi_1[C_1])\\ P(u), (\forall x)(P(x) \to Q(x)) \vdash Q(u)\end{matrix}}{P(u), (\forall x)(P(x) \to Q(x)) \vdash (\exists y)(P(a) \to Q(y)), Q(u)}\ w : r$$

We proceed "inductively" and construct $\psi_1[C_1]$:

$$\dfrac{\dfrac{P(u) \vdash P(u) \quad Q(u) \vdash Q(u)}{P(u), P(u) \to Q(u) \vdash Q(u)}\ \to : l}{P(u), (\forall x)(P(x) \to Q(x)) \vdash Q(u)}\ \forall : l$$

Putting the parts together we eventually obtain $\psi'[C_1]$:

$$\dfrac{\dfrac{\dfrac{P(u) \vdash P(u) \quad Q(u) \vdash Q(u)}{P(u), P(u) \to Q(u) \vdash Q(u)}\ \to : l}{P(u), (\forall x)(P(x) \to Q(x)) \vdash Q(u)}\ \forall : l}{P(u), (\forall x)(P(x) \to Q(x)) \vdash (\exists y)(P(a) \to Q(y)), Q(u)}\ w : r$$

For $C_2 = \vdash P(a)$ we obtain the projection $\psi'[C_2]$:

$$\dfrac{\dfrac{\dfrac{\dfrac{P(a) \vdash P(a)}{P(a) \vdash P(a), Q(v)}\ w : r}{\vdash P(a) \to Q(v), P(a)}\ \to : r}{\vdash (\exists y)(P(a) \to Q(y)), P(a)}\ \exists : l}{(\forall x)(P(x) \to Q(x)) \vdash (\exists y)(P(a) \to Q(y)), P(a)}\ w : l$$

In the next step we take a resolution refutation γ of $\mathrm{CL}(\psi, \alpha)$, construct a ground instance $\gamma\sigma$ via a ground substitution σ and insert appropriate instances of $\psi[C]\sigma$ into $\gamma\sigma$. The result is a proof with (only) atomic cuts of a sequent S' in which the occurrence α is eliminated.

Recall that

$$\mathrm{CL}(\psi', \alpha) = \{C_1 \colon P(u) \vdash Q(u),\ C_2 \colon \vdash P(a),\ C_3 \colon Q(u) \vdash\}.$$

First we define a resolution refutation δ of $\mathrm{CL}(\psi', \alpha)$:

$$\dfrac{\dfrac{\vdash P(a) \quad P(u) \vdash Q(u)}{\vdash Q(a)}\, R \qquad Q(v) \vdash}{\vdash}\, R$$

and a corresponding ground refutation γ:

$$\dfrac{\dfrac{\vdash P(a) \quad P(a) \vdash Q(a)}{\vdash Q(a)}\, R \qquad Q(a) \vdash}{\vdash}\, R$$

The ground substitution defining the ground refutation is

$$\sigma = \{u \leftarrow a, v \leftarrow a\}.$$

Let $\chi_1 = \psi'[C_1]\sigma$, $\chi_2 = \psi'[C_2]\sigma$ and $\chi_3 = \psi'[C_3]\sigma$. For a more compact representation let us write B for $(\forall x)(P(x) \to Q(x))$ and C for $(\exists y)(P(a) \to Q(y))$.

Then $\hat{\gamma}[\psi']$ is of the form

$$\dfrac{\dfrac{\dfrac{\overset{(\chi_2)}{B \vdash C, P(a)} \quad \overset{(\chi_1)}{P(a), B \vdash C, Q(a)}}{B, B \vdash C, C, Q(a)}\, cut \qquad \overset{(\chi_3)}{Q(a), B \vdash C}}{B, B, B \vdash C, C, C}\, cut}{B \vdash C}\ \text{contractions}$$

$\hat{\gamma}[\psi']$ can be considered as the result of a transformation eliminating the occurrence of $A \to A$ in S. ψ' was defined as $T_{cut}(\psi)$ where ψ is a proof of $B \vdash C$. Therefore $\hat{\gamma}[\psi']$ is a proof of the same end-sequent with only atomic cuts. ♯

To put things together we obtain a procedure for occurrence-elimination, which can be transformed into a cut-elimination procedure via T_{cut}. We call this procedure OCERES (*OCcurence-Elimination by RESolution*) and display the main steps below:

procedure OCERES(ψ):

input: A (skolemized) proof ψ, a left-occurrence α of a valid formula in the end-sequent S of ψ.
output: A cut-free proof χ of the end-sequent S without the formula occurring at α:

1. Compute $\mathrm{CL}(\psi, \alpha)$.
2. Compute a ground resolution refutation γ of $\mathrm{CL}(\psi, \alpha)$.
3. Compute $\phi : \hat{\gamma}[\psi]$.
4. Eliminate the atomic cuts in ϕ.

Then the cut-elimination procedure itself is simply defined as

$$\mathrm{CERES}(\psi) = \mathrm{OCERES}(T_{cut}(\psi)).$$

Remark: As the worst-case complexity of cut-elimination is nonelementary (i.e. its time complexity cannot be bounded by a fixed iteration of the exponential function) we cannot expect CERES to be simply a fast algorithm. However it is shown in [5] that, for some sequences of **LK**-proofs, CERES gives a nonelementary speed-up of Gentzen's procedure. This speed-up is based on a "redundancy" at the atomic level of the proofs which can be detected in the construction of the set of clauses $\mathrm{CL}(\psi, \alpha)$, but not by Gentzen's procedure. By the availability of efficient refinements and search strategies for resolution CERES also performs quite well in experiments (see [6]). ♯

REFERENCES

[1] W. Ackermann, *Über die Erfüllbarkeit gewisser Zählausdrücke*, ***Mathematische Annalen***, vol. 100 (1928), pp. 638–649.

[2] F. Baader and W. Snyder, *Unification theory*, ***Handbook of automated reasoning*** (A. Robinson and A. Voronkov, editors), vol. I, Elsevier Science, 2001, pp. 445–532.

[3] M. Baaz, C. Fermüller, and A. Leitsch, *A non-elementary speed-up in proof length by structural clause form transformation*, ***Proc. lics***, 1994, pp. 213–219.

[4] M. Baaz and A. Leitsch, *Cut normal forms and proof complexity*, ***Annals of Pure and Applied Logic***, vol. 97 (1999), pp. 127–177.

[5] ———, *Cut-elimination and redundancy-elimination by resolution*, ***J. Symbolic Computation***, vol. 29 (2000), pp. 149–176.

[6] M. Baaz, A. Leitsch, and G. Moser, *System description: Cutres 01, cut elimination by resolution*, ***Conference on automated deduction, cade-16***, Lecture Notes in Artificial Intelligence, Springer, 1999, pp. 212–216.

[7] W. Bernays and M. Schönfinkel, *Zum entscheidungsproblem der mathematischen logik*, ***Mathematische Annalen***, vol. 99 (1928), pp. 342–372.

[8] G. Boole, ***An investigation on the laws of thought***, Dover Publications, 1958.

[9] E. Börger, E. Grädel, and Y. Gurevich, ***The Classical Decision Problem***, Perspectives in Mathematical Logic, Springer, 1997.

[10] A. Church, *A note on the entscheidungsproblem*, ***The Journal of Symbolic Logic***, vol. 1 (1936), pp. 40–44.

[11] Martin Davis and Hilary Putnam, *A Computing Procedure for Quantification Theory*, ***Journal of the ACM***, vol. 7 (1960), no. 3, pp. 201–215.

[12] E. Eder, ***Relative complexities of first-order calculi***, Vieweg, 1992.

[13] U. Egly and T. Rath, *On the practical value of different definitional translations to normal form*, ***Proc. of cade-13***, Lecture Notes in Artificial Intelligence, vol. 1104, Springer, 1996, pp. 403–417.

[14] C. Fermüller, A. Leitsch, T. Tammet, and N. Zamov, ***Resolution methods for the decision problem***, Lecture Notes in Artificial Intelligence, vol. 679, Springer, 1993.

[15] G. Frege, *Begriffsschrift, eine der arithmetischen nachgebildete Formelsprache des reinen Denkens*, ***From Frege to Gödel: A sourcebook in mathematical logic 1879–1931*** (J. van Heijenoort, editor), Harvard University Press, 1967.

[16] H. Gelernter, *Realization of a geometry theorem proving machine*, ***Proceedings of the international conference on information processing***, June 1959, pp. 273–282.

[17] G. Gentzen, *Untersuchungen über das logische schließen*, ***Mathematische Zeitschrift***, vol. 39 (1934), pp. 405–431.

[18] P.C. Gilmore, *A proof method for quantification theory: its justification and realization*, ***IBM J. Res. Dev.***, (1960), pp. 28–35.

[19] Y. GIRARD, ***Proof theory and logical complexity***, Studies in Proof Theory, Bibliopolis, 1987.

[20] K. GÖDEL, *über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme*, ***Monatshefte für Mathematik und Physik***, vol. 38 (1931), pp. 175–198.

[21] ———, *Ein Spezialfall des Entscheidungsproblems der theoretischen Logik*, ***Ergebnisse Mathematischer Kolloquien***, vol. 2 (1932), pp. 27–28.

[22] R. HÄHNLE, *Tableaux and related methods*, ***Handbook of Automated Reasoning*** (A. Robinson and A. Voronkov, editors), vol. 1, Elsevier, 2001, pp. 101–178.

[23] J. HERBRAND, *Sur le probleème fondamental de la logique mathématique*, ***Sprawozdania z posiedzen Towarzysta Naukowego Warszawskiego, Wydzial III***, vol. 24 (1931), pp. 12–56.

[24] D. HILBERT and W. ACKERMANN, ***Grundzüge der theoretischen Logik***, Springer, Berlin, 1928.

[25] D. HILBERT and P. BERNAYS, ***Grundlagen der Mathematik***, Springer, 1970.

[26] W. JOYNER, *Automated theorem proving and the decision problem*, ***Ph.D. thesis***, Harvard University, 1973.

[27] W.H. JOYNER, *Resolution strategies as decision procedures*, ***Journal of the ACM***, vol. 23 (1976), pp. 398–417.

[28] R. KOWALSKI and P. HAYES, *Semantic trees in automatic theorem proving*, ***Machine intelligence*** (B. Meltzer and D. Michie, editors), vol. 7, American Elsevier, 1969, pp. 87–101.

[29] G. W. LEIBNIZ, *Calculus ratiocinator*, ***Sämtliche schriften und briefe*** (Preussische Akademie der Wissenschaften, editor), Reichel, Darmstadt, 1923.

[30] A. LEITSCH, ***The resolution calculus***, Springer. Texts in Theoretical Computer Science, 1997.

[31] JOHN W. LLOYD, ***Foundations of logic programming***, second ed., Springer, 1987.

[32] L. LÖWENHEIM, *Über Möglichkeiten im Relativkalkül*, ***Mathematische Annalen***, vol. 68 (1915), pp. 169–207.

[33] A. MARTELLI and U. MONTANARI, *Unification in linear time and space: a structured presentation*, ***Technical Report B76-16***, Istituto di elaborazione della informazione, 1976.

[34] S.Y. MASLOV, *The inverse method for establishing deducibility for logical calculi*, ***Proc. Steklov Inst. Math.***, vol. 98 (1968), pp. 25–96.

[35] W. MCCUNE, *Solution of the Robbins problem*, ***Journal of Automated Reasoning***, vol. 19 (1997), no. 3, pp. 263–276.

[36] J. A. ROBINSON, *Automatic deduction with hyperresolution*, ***Intern. Journal of Computer Math.***, vol. 1 (1965), pp. 227–234.

[37] ———, *A machine-oriented logic based on the resolution principle*, ***J. Assoc. Comput. Mach.***, vol. 12 (1965), pp. 23–41.

[38] G. TAKEUTI, ***Proof theory***, second ed., North-Holland, 1987.

[39] A. TURING, *On computable numbers with an application to the Entscheidungsproblem*, ***Proc. of the London Math. Soc.***, vol. Ser. 2 (1936/37), no. 42, pp. 230–265.

ALEXANDER LEITSCH,
TU-VIENNA,
FAVORITENSTRASSE 9-11,
1040 VIENNA, AUSTRIA
E-mail: leitsch@logic.at