Proof Systems for Effectively Propositional Logic

Juan Antonio Navarro[1] and Andrei Voronkov[2]

[1] Max Planck Institute for Software Systems
[2] The University of Manchester

Abstract. We consider proof systems for effectively propositional logic. First, we show that propositional resolution for effectively propositional logic may have exponentially longer refutations than resolution for this logic. This shows that methods based on ground instantiation may be weaker than non-ground methods. Second, we introduce a generalisation rule for effectively propositional logic and show that resolution for this logic may have exponentially longer proofs than resolution with generalisation. We also discuss some related questions, such as sort assignments for generalisation.

1 Introduction

Effectively propositional logic (in the sequel we call it simply EPR) is a fragment of first-order logic which can be effectively translated into propositional logic. Formulae in EPR are, essentially, those formulae in the Bernays-Schönfinkel class. It has recently been shown that several real life applications such as bounded model checking [11] and planning [12] can be naturally and succinctly encoded as EPR formulae. Effectively propositional benchmarks in the TPTP library [18] also include problems from diverse areas such as algebra, natural language processing, verification and puzzles.

When skolemised, EPR formulae contain no function symbols and this have a finite Herbrand Universe, which allows one to translate them to propositional logic using *grounding*: substitutions of constants for variables of the formula.

Grounding and the subsequent use of SAT solvers remains one of the most succesful approaches to checking the satisfiability of EPR formulae. The purpose of this paper is to compare several proof systems for EPR formulae, including those based on grounding and SAT. Although our results are formulated in terms of proof lengths, which is not always the most interesting criterion in practice, they give some insight on why proof systems for EPR can be more powerful than propositional proof systems. We believe that the insight gained from our results and their proofs may find their ways in practical proof systems for EPR formulae. In particular, we propose a new inference rule, called *generalisation*, which allows one to lift ground (propositional) reasoning to the non-propositional level.

Our first result is that on EPR formulae resolution can be exponentially more efficient than any propositional proof system working on a set of ground instances of EPR formulae. Although this is not surprising, and even perhaps expected, we are not aware of any previous formal proof of this result in the

literature. We prove this result by giving a family of formulae which have first-order resolution refutations of quadratic size, but whose propositional refutations are all exponential in size.

Our second result is that resolution with generalisation can have exponentially shorter proofs than resolution. Moreover, we define variants of the generalisation rule based on sort inference methods which allow one to generalise from a smaller set of clauses. We then show that generalisation using sorts is sound. We also show that some standard sort inference algorithms, such as the one implemented in the PARADOX model finder [3], are compatible with generalisation.

2 Preliminaries

A clause is called an *EPR clause* if it has no function symbols of arity > 0. That is, EPR clauses may contain variables and constants but not function symbols. In this paper we will consider only EPR clauses. An expression (for example, a clause) is called *ground* if it contains no variables. An *instance* of a clause C is any clause $C\sigma$ obtained from C by applying a substitution σ.

Given a set of clauses S, we denote by S^* the set of all ground instances of clauses in S:

$$S^* \stackrel{\text{def}}{=} \{S\sigma \mid S\sigma \text{ is ground and } S\sigma \text{ contains only constants from } S\} .$$

As usual, if S contains no constant symbols, then an arbitrary constant is added in order to compute S^*. We consider logic without equality, which is not a real restriction for EPR formulae since equality can be axiomatised using EPR formulae. Note that a collection S of ground EPR clauses can also be considered as a collection of propositional clauses by considering every ground atom A ocurriong in S as a distinct propositional variable.

To improve readability, we will sometimes write clauses as implications, for example $p \to q$ instead of $\neg p \vee q$.

2.1 Propositional vs first-order resolution

We prove in this section that resolution on EPR formulae can be exponentially more efficient than propositional proof systems using grounding. First, we should explain what we mean by the latter. By a propositional proof system for EPR clauses we mean any proof system that, given a set S of clauses, builds a subset of S^* and applies a propositional proof system to this subset. Note that if a propositional proof system finds an unsatisfiable subset of S^*, then S is unsatisfiable too.

We will now proceed to show that there is a family of sets of clauses S_i with respective resolution refutations Γ_i, such that the shortest propositional refutation of S_i is exponentially larger than Γ_i.

Let i be a positive integer. Consider the language having two constant symbols 0 and 1, and a single predicate symbol s of arity i. Denote by $\bar{0}$, $\bar{1}$, $\bar{x}$, etc.

sequences of constants 0, 1 and variables, respectively, whose length will be clear from the context. The set S_i consists of the following: the clause

$$s(\bar{0}) \ , \tag{1}$$

the clause

$$\neg s(\bar{1}) \ , \tag{2}$$

and i clauses of the form:

$$s(\bar{x}, 0, \bar{1}) \rightarrow s(\bar{x}, 1, \bar{0}) \ , \tag{3}$$

where the length of $\bar{1}$ ranges from $0, \ldots, i-1$. Note that the size of S_i is $O(i^2)$.

Lemma 1. *The shortest propositional refutation of S_i has a size of $O(2^i)$.*

Proof. Note that every ground atom in the language of S_i is of the form $s(\bar{b})$, where $\bar{b}$ is a sequence of bits of length i. We can consider this sequence of bits as a non-negative integer between 0 and $2^i - 1$ written in binary notation. For a number n such that $0 \leq n < 2^i$ let us denote by $\underline{n}$ the sequence of i bits denoting this number. Using this notation we can observe that S_i^* consists of the following clauses: $s(\underline{0})$, $\neg s(\underline{2^i - 1})$, and all clauses of the form $s(\underline{n}) \rightarrow s(\underline{n+1})$ for $n = 0, \ldots, 2^i - 1$.

Moreover, it is not hard to argue that every proper subset of S_i^* is satisfiable, so every propositional refutation of S_i should be at least as large as S_i^*. It remains to note that the number of literals in S_i^* is $O(2^i)$. □

Lemma 2. *S_i has a resolution refutation of the size $O(i^2)$.*

Proof. Let us build, for every $k = 1, \ldots, i-1$ a resolution derivation Π_k from S_i of the clause

$$s(\bar{x}, \bar{0}) \rightarrow s(\bar{x}, \bar{1}), \tag{4}$$

where k is the length of $\bar{0}$. This derivation will be built by induction on k. When $k = 1$, (4) is of the form (3). For $k > 1$, we will take the derivation Π_{k-1} and add several clauses to it to obtain Π_k. We know that Π_{k-1} derives a clause

$$s(\bar{x}, y, \bar{0}) \rightarrow s(\bar{x}, y, \bar{1}) \ . \tag{5}$$

where the length of $\bar{0}$ is k. From this and (3) we derive by a resolution inference the clause

$$s(\bar{x}, 0, \bar{0}) \rightarrow s(\bar{x}, 1, \bar{0}) \ .$$

From this and (5) we derive by a resolution inference the clause

$$s(\bar{x}, 0, \bar{0}) \rightarrow s(\bar{x}, 1, \bar{1}) \ .$$

and we are done. It is easy to see that the number of clauses in Π_k not ocurring in S is $3(k-1)$, so the size of Π_k is $O(i \cdot k)$.

Note that Π_i derives

$$s(\bar{0}) \rightarrow s(\bar{1})$$

and has the size $O(i^2)$. From this clause, (1) and (2) we can derive the empty clause in 2 steps, so there exists a resolution refutation of S_i having the size $O(i^2)$. □

Lemmas 1 and 2 yield the following theorem

Theorem 1. *There is a sequence of sets of clauses $S_1, S_2, \ldots$ of increasing size such that each S_i has a refutation of a size quadratic in i, while the shortest propositional refutation of S_i has a size exponential in i.*

Interestingly, as the results of the recent CASC competition show [17], resolution has so far been found not very competitive on EPR formulae. In the following section we introduce another inference rule, especially designed for effectively propositional formulae and intended to complement the resolution approach.

3 The generalisation inference rule

Suppose that S is a set of EPR clauses whose constants are in the set $c_1, \ldots, c_n$. Let $A[x]$ be a quantifier-free formula with a free variable x written in the same language as S. Let us note that the formula

$$\forall x(A[c_1] \wedge \cdots \wedge A[c_n] \rightarrow A[x]) \tag{6}$$

is valid in all Herbrand models of S. This, and the fact that $A[x]$ is quantifier-free, implies that adding (6) to S does not change the set of Herbrand models. This gives us an idea of a *generalization rule*: an inference rule that allows one to derive $A[x]$ from $A[c_1], \ldots, A[c_n]$. We will introduce a more general form of this rule inspired by grounding systems based on sort inference [see e.g. 16]. These systems may derive that the set of instantiations for some variable in s may be restricted to a subset of all constants. Likewise, we can make generalisation from a subset of constants.

Let us give some definitions. We call a *predicate position* a pair (p, i) where p is a predicate symbol and i is a positive integer less than or equal to the arity of p. When we denote positions we will write $p.i$ instead of (p, i). We call a *sort* a set of constant symbols, and a *sort assignment* a function that maps each predicate position to a sort. A ground formula F is said to be *compatible with a sort assignment* A if for every atomic subformula $p(t_1, \ldots, t_n)$ of F, we have $t_i \in A(p.i)$. For a quantifier-free formula F, denote by $F|_A$ the set of all ground instances of F compatible with A.

Definition 1. *Given a sort assignment A, the inference rule of* generalisation with respect to A *is*

$$\frac{C_1 \vee p(\bar{t}_1, c_1, \bar{u}_1) \quad \cdots \quad C_n \vee p(\bar{t}_n, c_n, \bar{u}_n)}{C_1\sigma \vee \cdots \vee C_n\sigma \vee p(\bar{t}_1, y, \bar{u}_1)\sigma}\ \mathsf{Gen}_A$$

where y is a fresh variable, the length of $\bar{t}_i$ is k, $A(p.k{+}1) = \{c_1, \ldots, c_n\}$ and σ is the most general simultaneous unifier of the set of tuples $\{(\bar{t}_1, \bar{u}_1), \ldots, (\bar{t}_n, \bar{u}_n)\}$.

If both inference rules, resolution and generalization, are allowed then the system is (trivially) complete; this follows since resolution alone is already complete. The discussion in the beginning of this section also suggests that it is also sound when the sort function always returns the set of all constants.

One of the first results that we want to show, is that when resolution is combined with generalisation, then the obtained inference system is still, as resolution alone, refutationally sound and complete. For this we begin by introducing the following set of clauses which will be useful to simulate generalisation using only resolution.

Definition 2. *Given a sort assignment A, we define the set of* generalisation clauses*, denoted by $[\![A]\!]$, as the set containing all the clauses of the form*

$$p(\bar{x}, c_1, \bar{z}) \wedge \cdots \wedge p(\bar{x}, c_n, \bar{z}) \rightarrow p(\bar{x}, y, \bar{z}) \ , \tag{7}$$

where k is the length of $\bar{x}$ and $A(p.k+1) = \{c_1, \ldots, c_n\}$.

Notice that all clauses in $[\![A]\!]|_A$ are tautologies, since the variable y would have to be mapped to a constant c_i of its appropriate sort.

Lemma 3. *If there is a refutation of a set of clauses S using resolution and generalisation with respect to a sort assignment A, then there is a refutation of $S \cup [\![A]\!]$ using only resolution.*

Proof. The result easily follows by noticing that generalisation inference steps can be simulated by resolving the n clauses of the form $C_i \vee s(\bar{x}, c_i, \bar{z})\sigma_i$ with the corresponding clause in the set $[\![A]\!]$ from Definition 2. □

Let us now discuss sort inference functions that result in sort assignments preserving satisfiability.

Definition 3. *A* sort inference function *Ξ is a function that yields a sort assignment given a set of clauses as input. Moreover, we say that Ξ is*

- valid *if, for any set of clauses S, the sets S and $S|_A$ are equisatisfiable; and*
- stable *if, for any set of clauses S, $\Xi(S \cup [\![A]\!]) = A$.*

where $A = \Xi(S)$.

The first condition, validity, states that when checking the satisfiability of S by using instantiation-based methods, the generation of ground instances can be restricted to those which are compatible with A. The condition of stability asserts that the sort inference procedure is not affected when the set of generalisation clauses $[\![A]\!]$ is added to a formula.

The following theorem proves that resolution can be extended with generalisation preserving soundness.

Theorem 2. *Let S be a set of clauses, Ξ be a valid and stable sort inference function, and $A = \Xi(S)$. If there is a refutation of S using resolution and generalisation with respect to A, then there is a refutation of S using resolution only.*

Proof. We will first show that the set of clauses S is equisatisfiable with $S \cup [\![A]\!]$. Since the sort inference function is stable, we know that $\Xi(S) = \Xi(S \cup [\![A]\!]) = A$. Moreover, since the sort inference is valid, the sets $S \cup [\![A]\!]$ and $(S \cup [\![A]\!])|_A = S|_A \cup [\![A]\!]|_A$ are equisatisfiable. But recall that $[\![A]\!]|_A$ contains only tautologies and, therefore, $S|_A \cup [\![A]\!]|_A$ and $S|_A$ are equisatisfiable. Finally, again by the validity of the sort inference function, we get that $S|_A$ and S are equisatisfiable.

Now, to prove the theorem statement, suppose that there is a refutation of S using resolution and generalisation with respect to A. Then, by Lemma 3 and since resolution is sound, $S \cup [\![A]\!]$ is unsatisfiable. But then, from the previous paragraph, it follows that S is unsatisfiable and, since resolution is refutation complete, there is a refutation of S using resolution only. □

From this, our main result now follows as a simple corollary.

Corollary 1. *An inference rule system based on resolution and generalisation, with respect to a valid and stable sort inference function, is both refutationally sound and complete.*

Proof. Soundness follows by Theorem 1, while completeness is directly inherited from the completeness of resolution. □

From this result it follows that, any valid and stable sort inference function is suitable to be combined with generalisation in order to produce a sound and complete inference system. The question on how to obtain such kind of sort inference functions, however, remains open. This is the matter of the following section.

3.1 Sort inference for generalisation

In this section we will explore some possibilities in order to generate sort information that can be combined with the generalisation inference rule. A first option, though not very interesting, is to assign the trivial sort assignment to all sets of clauses.

Definition 4. *Given an effectively propositional language with a domain $\mathcal{D}$ of constant symbols, the* trivial sort assignment *is the function that maps every predicate position to $\mathcal{D}$.*

That is, it uses the domain of the logic itself as the sort for all variables and positions in predicates. This procedure is clearly stable since, irrelevant to the particular set of input clauses, the trivial sort assignment is always used. Moreover, by Herbrand's theorem this procedure is also valid and therefore a suitable candidate to be used together with generalisation.

In the following Section 3.2, we will see how even this simple approach can already represent a significant advantage over using the resolution inference rule alone. However, particularly on problems from applications, it is very likely that more specialised sort inference functions are able to give even better results in practise.

An example of a sort inference function is the method proposed by Claessen and Sörensson [3] and implemented in PARADOX in the context of grounding-based model finding.

Algorithm 1 (Basic sort inference). Start with a sort assignment giving unrelated empty sorts to each predicate position.

Processing one clause at a time, and as a union-find algorithm: for each variable in the clause, merge the sorts assigned to all predicate positions where that variable occurs; and, for each constant symbol, add the constant symbol to the sort assigned to the predicate position where it appears.

Finally, add a dummy constant symbol to any sort that still remained empty at the end of this procedure.

It is not hard to argue, as it is done by Claessen and Sörensson [3], that this sort inference function is valid –in the sense of our Definition 3– and that, moreover, is not affected when adding the set of clauses $[\![A]\!]$ to S. So, from Theorem 2, it follows that we already have a sort inference method that, without any further modifications, can be directly used to empower generalisation inferences.

It is to be expected, that if one obtains a sort inference method by extending some available technique to restrict the number of generated instances in a grounding approach, then the obtained method will most likely be valid. This follows since such methods actually work by replacing the satisfiability testing for an effectively propositional formula, to checking instead an equisatisfiable set of propositional instances. This equisatisfiability is, precisely, what the property of validity asks for.

Unfortunately, however, not any ground restriction method can be so easily integrated with generalisation. Using the idea of positional linking, also called structural constraints by Schulz [16] and implemented in EGROUND, one can easily define the following sort inference function.

Algorithm 2 (Positional sort inference). Given a set of clauses S compute, for every signed predicate $s.i$, the set $T_{s.i}$ as the set of all terms that appear in a literal with a signed predicate s at position i, i.e. if $s(t_1, \ldots, t_n)$ appears in S then $t_i \in T_{s.i}$. Then let $C_{s.i} = T_{s.i}$ if all terms in $T_{s.i}$ are constants, and $C_{s.i} = \mathcal{D}$ otherwise.

Now, for each predicate position $p.i$, the *positional sort inference* is defined as the function that maps each such set S, to the sort assignment $A_{p.i} = C_{p.i} \cap C_{\neg p.i}$.

This sort inference function is clearly also valid. It works by the observation that literals which are not compatible with the generated sort assignment would be pure, i.e. they only appear in one of the two possible phases, and so they can be discarded. However, this sort inference function is not stable, as the following example shows.

Example 1. Consider the following satisfiable set of clauses S:

$$p(a)$$
$$\neg p(x) \vee q(x)$$
$$\neg q(b)$$

The positional sort would have $A_{p.1} = \{a\}$, $A_{q.1} = \{b\}$, and the restricted set $S|_A$ is simply $\{p(a), \neg q(b)\}$. Note that, however, adding the clause

$$\neg p(a) \vee p(x)$$

which is part of $[\![A]\!]$, would cause $A_{p.1} = \mathcal{D} = \{a, b\}$ —because now a variable appears in $p(x)$ on both positive and negative phases— making the sort inference function unstable and rendering the set of clauses $S \cup [\![A]\!]$ unsatisfiable.

In this section we have shown how a sort inference method, as proposed by Claessen and Sörensson [3], can be used together with the generalisation inference rule in order to make it more easily applicable in practice. In the following we give, in the form of a theoretical result, some evidence on why combining generalisation with resolution is likely to produce a powerful reasoning system.

3.2 Generalisation vs resolution

In this section, and in order to further motivate the use of the generalisation inference rule in combination with resolution, we show a family of formulae which, similar to the one given in Section 2.1, shows that refutations can become exponentially shorter when combining resolution with the generalisation inference rule. For doing so we will show an example of a series of unsatisfiable sets of clauses $S_1, \ldots, S_n$ such that the length of shortest resolution refutation of S_n is exponential in n, while using both generalisation and resolution it is possible to find a refutation of size quadratic in n. In the following we will use x_i to represent variables, b_i and c_i for constant symbols, as well as s_i and t_i for arbitrary terms.

Definition 5. *Take a logic whose language has a set of constant symbols $\mathbb{B} = \{0, 1\}$ and let n be a non-negative number. For every i, with $0 \leq i \leq n$, there is a pair of predicate symbols p_i and q_i both of arity i.*

Now let S_n be the set of clauses that contains: the clause

$$p_0 \ , \tag{8}$$

$2n$ clauses, two for every $0 \leq i < n$, of the form

$$\begin{aligned} p_i(x_1, \ldots, x_i) &\rightarrow p_{i+1}(x_1, \ldots, x_i, 0) \ , \\ p_i(x_1, \ldots, x_i) &\rightarrow p_{i+1}(x_1, \ldots, x_i, 1) \ , \end{aligned} \tag{9}$$

the clause

$$p_n(x_1, \ldots, x_n) \rightarrow q_n(x_1, \ldots, x_n) \ , \tag{10}$$

n clauses, one for every $0 \leq i < n$, of the form

$$q_{i+1}(0, x_{\underline{i}}, \ldots, x_n) \wedge q_{i+1}(1, x_{\underline{i}}, \ldots, x_n) \rightarrow q_i(x_{\underline{i}}, \ldots, x_n) \ , \tag{11}$$

where $\underline{i} = n - i + 1$, and the clause

$$\neg q_0 \ . \tag{12}$$

Moreover, we will assume that generalisation inferences are applied with respect to the trivial sort assignment that simply maps every predicate position to the domain set $\mathbb{B} = \{0, 1\}$.

Intuitively, clauses of the form (9) encode the fact that if $p_i(b_1, \ldots, b_i)$ is true, then p_n should be true for all n-bit strings with a prefix of $b_1, \ldots, b_i$. A dual of this is encoded by clauses of the form (11): if $q_i(b_{\underline{i}}, \ldots, b_n)$ is false, then q_n should be false for some n-bit string with a suffix of $b_{\underline{i}}, \ldots, b_n$.

From clauses (8) and (9) we get that $p_n(b_1, \ldots, b_n)$ is true for all n-bit strings. Then from (10) that $q_n(b_1, \ldots, b_n)$ is also true for all n-bit strings and, therefore from (11), the atom q_0 should be true. But this causes a contradiction with (12), so the set S_n is unsatisfiable.

Indices in variables and terms have been chosen to enforce the *prefix* and *suffix* intuition of these predicate symbols. Formally, in the atoms $p_i(t_1, \ldots, t_i)$ and $q_i(t_{\underline{i}}, \ldots, t_n)$, we say that the position of the term t_i is the i-th *bit position*. Note that in all clauses of S_n, the variable x_i only appears at the i-th bit position of an atom.

Theorem 3. *There is a refutation of* S_n*, using both generalisation and resolution inference rules, which is of size quadratic in* n*.*

Proof. We start our refutation with the clause (8) which is the fact p_0. Observe now that it is possible to extend a proof of

$$p_i(x_1, \ldots, x_i) \tag{13}$$

to a proof of

$$p_{i+1}(x_1, \ldots, x_i, x_{i+1}) \tag{14}$$

by adding a constant number of steps.

To do this, first apply a generalisation inference on the pair of clauses (9) to obtain

$$p_i(x_1, \ldots, x_i) \to p_{i+1}(x_1, \ldots, x_i, x_{i+1}) \ ,$$

and then resolve this with (13) to obtain (14).

After n iterations of this procedure we get a proof of $p_n(x_1, \ldots, x_n)$ whose length is linear in n. Now, resolve this with (10) to obtain $q_n(x_1, \ldots, x_n)$. Observe now that we can extend a proof of

$$q_{i+1}(x_{\underline{i}-1}, x_{\underline{i}}, \ldots, x_n) \tag{15}$$

to a proof of

$$q_i(x_{\underline{i}}, \ldots, x_n) \tag{16}$$

by adding a constant number of steps.

To do this, simply resolve (15) with (11) to obtain

$$q_{i+1}(1, x_{\underline{i}}, \ldots, x_n) \to q_i(x_{\underline{i}}, \ldots, x_n) \ ,$$

and again with (15) to finally obtain (16).

After n of such iterations we end with a proof of q_0 which is also of length linear in n. Finally resolving q_0 with (12) we obtain a refutation of S_n. Since the size of each clause in the refutation is also linear in n, the size of the refutation is quadratic in n. □

Our main theorem of this section states that, using resolution alone, even the shortest refutation is of length at least exponential in n. The idea of the proof is not very difficult, but proving some of the necessary lemmas gets rather involved. We will therefore give first a sketch of the proof, followed by a couple of lemmas without proofs, followed by a formal proof of the main theorem relying on those lemmas. The missing proofs, as well as some other aditional details, can be found in an appendix of the full version of this paper.[3]

Theorem 4. *A resolution refutation of S_n has a length of, at least, 2^n.*

Proof (Sketch of proof). To prove that any refutation of S_n has at least an exponential length, we introduce a function on sets of clauses that, in a way, measures the accumulated progress achieved step by step on a refutation.

This *work* function, denoted by w, will map the set of clauses occurring in a partial proof Γ to the set of n-bit strings which, intuitively, have already been *consumed* while trying to build a refutation. This function should moreover satisfy $w(S_n) = \emptyset$, while the work of any refutation Γ is $w(\Gamma) = \mathbb{B}^n$. In order to prove the theorem statement it would then be enough to show that from one step to the next in the refutation, the work done increases in, at most, a single element. In other words all elements in $\mathbb{B}^n$ would have to be consumed one by one, yielding a proof of exponential length. □

In order to define such work function, first we identify the kind of clauses that might appear in a refutation of S_n. We observe that, indeed, only two kinds of clauses are possible (more details in the paper's full version):

- type I are clauses, such as (8-10), of the form

$$[l_1] \to h \tag{17}$$

  where the head h has a predicate symbol p_j, with $0 \leq j \leq n$, or q_n.
- type II are clauses, such as (11-12), of the form

$$l_1 \wedge \cdots \wedge l_m \to h \tag{18}$$

  where the head h is either $\bot$ or has a predicate symbol q_i with $i < n$. Moreover, we say that a literal l_i is *active* if its bit positions overlap with those of h, and *inactive* otherwise.

It is easy to check that resolving together two clauses of type I will yield another clause of type I, while resolving a clause of type II with one of either type will also produce a clause of type II.

[3] Available at: `http://www.mpi-sws.mpg.de/~jnavarro/papers.html`

Definition 6 (Work function). *For a clause of type II with head $q_i(t_{\underline{i}}, \ldots, t_n)$ we first define the* work of a literal l *as follows:*

$$w(l) = \begin{cases} \mathbb{B}^n & l = \bot, \\ \mathbb{B}^{n-j} \times \hat{t}_{\underline{j}} \times \cdots \times \hat{t}_n & l = q_j(t_{\underline{j}}, \ldots, t_n), \\ \hat{t}_1 \times \cdots \times \hat{t}_n & l = p_j(t_1, \ldots, t_j) \text{ for an active } l, \\ \emptyset & \text{if } l \text{ is inactive.} \end{cases}$$

where $\hat{t}$ is the set of all ground instances of the term t. We also let $w^c(l) = \mathbb{B}^n \setminus w(l)$. The work of the clause C *is then defined as*

$$w(C) = w^c(l_1) \cap \cdots \cap w^c(l_m) \cap w(h) \tag{19}$$

For clauses of type I we let $w(C) = \emptyset$. Finally, the work of a set of clauses is the union of the work of each clause in the set.

From this definition it is not difficult to check that, indeed, $w(S_n) = \emptyset$; while the work of the empty clause (which is of type II), and therefore of any refutation of S_n, is $\mathbb{B}^n$. A significant ammount of the proof details are actually spent in showing that (1) non-ground clauses have an empty work, and (2) the work function is invariant with respect to the application of substitutions. In the paper's full version we prove the following two lemmas.

Lemma 4. *Let C be any clause in a refutation of S_n. If C is non-ground, then the work $w(C) = \emptyset$.*

Lemma 5. *Let C be any clause in a refutation of S_n and let σ be a substitution. It then follows that $w(C) = w(C\sigma)$.*

The former lemma will allow us to quickly discard many resolution steps as points where the work could increase in a refutation. The later allows us to more easily analize the effect of the remaining inferences, since the substitution that may be required in order to unify and resolve a pair of clauses will not affect the work measure of each individual clause. Having these, we can now prove the following lemma which is the core of the main result.

Lemma 6. *Let C, C_1, and C_2 be clauses in a refutation of S_n such that C is obtained by resolving another pair of clauses C_1 and C_2 with a unifier σ. It then follows that $\delta = w(C) \setminus w(\{C_1, C_2\})$ has at most one element.*

Proof. If C is either non-ground or of type I, then we know, respectively by Lemma 4 and by definition, that $w(C) = \emptyset$ and the result is trivial. We threfore assume that C is a ground clause of type II.

Suppose that C is the result of resolving two clauses of types I and II.

$$\begin{aligned} C_1\sigma &: [l_1'] \to l_1 \\ C_2\sigma &: l_1 \wedge l_2 \wedge \cdots \wedge l_m \to h \\ C &: [l_1'] \wedge l_2 \wedge \cdots \wedge l_m \to h \end{aligned}$$

Take $\bar{b} \in \delta \subseteq w(C)$, in particular, $\bar{b} \in w^c(l_2) \cap \cdots \cap w^c(l_m) \cap w(h)$. However, since $\bar{b} \notin w(C_2)$ and, by Lemma 5, $\bar{b} \notin w(C_2\sigma)$, it must therefore be the case that $\bar{b} \notin w^c(l_1)$ or, equivalently, $\bar{b} \in w(l_1)$. This actually shows that $\delta \subseteq w(l_1)$. Observe, however, that since l_1 appears as the head of clause a clause of type I, its predicate symbol is either p_j or q_n and, since moreover the predicate is ground, $w(l_1)$ will contain at most a single element (c.f. Definition 6).

Suppose that otherwise C is the result of resolving two clauses of type II.

$$\begin{aligned} C_1\sigma &: l'_1 \wedge \cdots \wedge l'_{m'} \rightarrow l_1 \\ C_2\sigma &: l_1 \wedge l_2 \wedge \cdots \wedge l_m \rightarrow h \\ C &: l'_1 \wedge \cdots \wedge l'_{m'} \wedge l_2 \wedge \cdots \wedge l_m \rightarrow h \end{aligned}$$

Exactly as in the previous case we can prove for any $\bar{b} \in \delta$ that also $\bar{b} \in w(l_1)$. Now, however, from $\bar{b} \in w(C)$ we also get $\bar{b} \in w^c(l'_1) \cap \cdots \cap w^c(l'_{m'})$ and, from this, that $\bar{b} \in w(C_1\sigma) = w(C_1)$, contradicting the hypothesis that $\bar{b} \notin w(C_1)$. The conclusion is that, in this case, $\delta = \emptyset$. □

Our main theorem now follows as a simple corollary of the previous result.

Corollary 2. *A resolution refutation of S_n has a length of, at least, 2^n.*

Proof. Let $\Gamma = C_1, \ldots, C_m$ be a refutation of S_n, i.e. $C_m = \bot$. Now, for every $1 < i \leq m$ let $\delta_i = w(C_1, \ldots, C_i) \setminus w(C_1, \ldots, C_{i-1})$ be the work increment in the proof that is provided by the clause C_i. If C_i is an hypothesis in S_n, then by definition its work is empty and, therefore, $\delta_i = \emptyset$. Suppose that, otherwise, the clause C_i is the result of applying resolution among another pair of clauses C_j and C_k with $i > j$ and $i > k$; as a consequence of Lemma 6 we know that δ_i contains, at most, one element.

Since all elements in $w(\Gamma) = \mathbb{B}^n$ had to be incorporated one element at a time, it therefore follows that the length of the proof, m, is at least 2^n. □

4 Related work

Although quite recent, there has been a significant interest of the automated reasoning community on effectively propositional logic. The CADE ATP System Competition, since its JC instalment in 2001, holds a division for EPR problems [13]. Moreover, a number of theorem provers particularly geared towards this class of formulae have also been developed. These include, for example, EGROUND by Schulz [16], PARADOX by Claessen and Sörensson [3], DARWIN by Fuchs [5], and IPROVER by Korovin [8].

Systems such as EGROUND and PARADOX implement an instantiation approach where ground instances of the given input formula are generated and then tests for satisfiability are run by a SAT solver. Several ideas have been then proposed in order to limit the number of instances that have to be generated. The work of Schulz [16] discusses and compares some of the early approaches,

while Claessen and Sörensson [3] introduce the notion of *sort inference* in the context of *MACE-style* model finding..

In a closely related line of research, the notion of hyper-linking [9, 14] was also proposed in order to restrict the kind of inferences that need to be performed in instantiation based methods. Other related ideas can be found in the work of Hooker et al. [7]. Alternatively, Ganzinger and Korovin [6] propose the use of first-order reasoning (e.g. resolution) to drive the instantiation process; while Baumgartner and Tinelli [1] try to avoid direct instantiation by lifting the propositional DLL algorithm [4] to the first-order level.

Voronkov [20] proposed to use relational databade technology and equality constraints for implementing resolution on EPR formulae and shows that set-at-a-time resolution related operations, such as resolution and subsumption, can be implemented using database operations, for example, joins. A similar observation was later made by Tammet and Kadarpik [19].

Combinations of tableaux related techniques with propositional satisfiability checking have also been proposed by researchers such as Billon [2], Letz and Stenz [10].

5 Conclusion and Future Work

We proved several results showing potential of non-ground methods for proving EPR formulae. First, we proved that resolution can have exponentially shorter proofs than propositional procedures for EPR. Second, we proposed a generalisation rule and showed that resolution with generalisation can have exponentially shorter proofs than resolution.

We find the generalisation rule especially appealing for practical theorem proving. Implementation techniques and heuristics for applying this rule should be developed. Further, it is interesting to investigate soundness of stronger versions of generalisation, both theoretically and practically. In practice, one can try to use potentially unsound versions of this rule and then see if a refutation obtained by a potentially unsound version can be made into a valid refutation.

A crucial step in the efficiency of the generalisation rule is to find new sort inference techniques that restrict sorts as much as possible yet are still sound. We show how some existing sort inference techniques, such as those developed for grounding-based approaches, can be directly applied in this context; while some others, particularly based on linking restrictions, cannot be used in a sound way as easily.

Incidentally, the proofs of these two exponential gaps provide us with benchmark families that might be interesting to test with existing systems. We have shown that, when reasoning with a particular inference system, some unsatisfiable problems have rather short refutations, but are existing implementations of theorem provers able to find such short proofs? Or, which heuristics can we use in order to find these shorter proofs with higher probability?

Further directions for future work include the research on techniques for efficiently implementing and integrating the proposed generalisation inference

rule with other systems which already make use of resolution, such as iProver [8] or Vampire [15]. Alternatively, it might also prove fruitful to investigate possible extensions of this inference rule in order to make use of complete linking information which can perhaps better describe the underlying structure of the problem being solved.

Bibliography

[1] Peter Baumgartner and Cesare Tinelli. The model evolution calculus. In Franz Baader, editor, *CADE-19: Proceedings of the 19th International Conference on Automated Deduction*, number 2741 in Lecture Notes in Artificial Intelligence, pages 350–364. Springer, 2003.

[2] Jean-Paul Billon. The disconnection method. A confluent integration of unification in the analytic framework. In *TABLEAUX'06: Proceedings of the International Conference on Automated Reasoning with Analytic Tableaux and Related Methods*, number 1071 in Lecture Notes in Artificial Intelligence, pages 110–126, Terrasini, Italy, May 1996. Springer.

[3] Koen Claessen and Niklas Sörensson. New techniques that improve MACE-style model finding. In *MODEL'03: Proceedings of the Workshop on Model Computation*, 2003.

[4] Martin Davis, George Logemann, and Dondald Loveland. A machine program for theorem-proving. *Communications of the ACM*, 5:394–397, 1962.

[5] Alexander Fuchs. Darwin: A theorem prover for the model evolution calculus. Master's thesis, University of Koblenz-Landau, 2004.

[6] Harald Ganzinger and Konstantin Korovin. New directions in instantiation-based theorem proving. In *LICS'03: Proceedings of the 18th Annual IEEE Symposium on Logic in Computer Science*, page 55, Ottawa, Canada, June 2003. IEEE Computer Society. ISBN 0-7695-1884-2.

[7] John N. Hooker, G. Rago, V. Chandru, and A. Shrivastava. Partial instantiation methods for inference in first order logic. *Journal of Automated Reasoning*, 28(3):371–396, 2002.

[8] Konstantin Korovin. Implementing an instantiation-based theorem prover for first-order logic. In C. Benzmueller, B. Fischer, and Geoff Sutcliffe, editors, *IWIL'06: Proceedings of the 6th International Workshop on the Implementation of Logics, held at the 13th International Conference on Logic for Programming, Artificial Intelligence and Reasoning (LPAR'06)*, volume 212 of *CEUR Workshop Proceedings*, Phnom Penh, Cambodia, 2006.
See also: `http://www.cs.man.ac.uk/~korovink/iprover/`.

[9] Shie-Jue Lee and David A. Plaisted. Eliminating duplication with the hyper-linking strategy. *Journal of Automated Reasoning*, 9(1):25–42, 1992.

[10] Reinhold Letz and Gernot Stenz. DCTP: A disconnection calculus theorem prover. In *IJCAR'01: Proceedings of the First International Joint Conference on Automated Reasoning*, number 2083 in Lecture Notes in Computer Science, pages 381–385, Siena, Italy, June 2001. Springer.

[11] Juan Antonio Navarro Pérez and Andrei Voronkov. Encodings of bounded LTL model checking in effectively propositional logic. In *CADE-21: Proceedings of the 21st International Conference on Automated Deduction*, number 4603 in Lecture Notes in Artificial Intelligence, pages 346–361, Bremen, Germany, 2007. Springer.

[12] Juan Antonio Navarro Pérez and Andrei Voronkov. Planning with effectively propositional logic. In *Collection of Papers Dedicated to Harald Ganzinger's Memory*. 2007. To appear.

[13] Francis Jeffry Pelletier, Geoff Sutcliffe, and Christian B. Suttner. The development of CASC. *AI Communications*, 15(2):79–90, 2002. ISSN 0921-7126.

[14] David A. Plaisted and Yunshan Zhu. Ordered semantic hyper-linking. *Journal of Automated Reasoning*, 25(3):167–217, 2000.

[15] Alexandre Riazanov and Andrei Voronkov. The design and implementation of Vampire. *AI Communications*, 15(2):91–110, 2002. ISSN 0921-7126.

[16] Stephan Schulz. A comparison of different techniques for grounding near-propositional CNF formulae. In Susan Haller and Gene Simmons, editors, *FLAIRS'02: Proceedings of the Fifteenth International Florida Artificial Intelligence Research Society Conference*, pages 72–76. AAAI Press, 2002.

[17] Geoff Sutcliffe. The CADE-21 ATP system competition website, 2007. URL `http://www.cs.miami.edu/~tptp/CASC/21/`.

[18] Geoff Sutcliffe and Christian B. Suttner. The TPTP problem library: CNF release v1.2.1. *Journal of Automated Reasoning*, 21(2):177–203, 1998.

[19] Tanel Tammet and Vello Kadarpik. Combining an inference engine with database: A rule server. In Michael Schroeder and Gerd Wagner, editors, *RuleML*, volume 2876 of *Lecture Notes in Computer Science*, pages 136–149, 2003.

[20] Andrei Voronkov. Merging relational database technology with the constraint technology. In *Perspectives of System Informatics. Andrei Ershov Second International Memorial Conference (Preliminary Proceedings)*, pages 189–195, Novosibirsk, Akademgorodok, Russia, June 1996.