Solving Satisfiability in Ground Logic with Equality by Efficient Conversion to Propositional Logic

Igor Gammer and Eyal Amir

Department of Computer Science
University of Illinois, Urbana-Champaign
{igammer2,eyal}@uiuc.edu

Abstract *Ground Logic with Equality* ($GL^{=}$) is a subset of First-Order Logic (FOL) in which functions or quantifiers are excluded, but equality is preserved. We argue about $GL^{=}$'s unique position (in terms of expressiveness and ease of decidability) between FOL and Propositional Logic (PL). We aim to solve satisfiability (SAT) problems formulated in $GL^{=}$ by converting them into PL using a satisfiability-preserving conversion algorithms, and running a general SAT solver on the resulting PL Knowledge Base (KB). We introduce two conversion algorithms, with the latter utilizing the former as a subroutine, and prove their correctness - that is, that the translation preserves satisfiability. The main contribution of this work is in utilizing input fragmentation to yield PL KBs that are smaller than possible prior to our work, thus resulting in the ability to solve $GL^{=}$ SAT problems faster than was possible before.

1 Introduction

Satisfiability in PL is well-studied in Computer Science because of its theoretical significance and the multitude of practical applications that it has. A significant number of real-world problems can be modeled in or translated to propositional knowledge bases (KBs), such that solution to SAT on the PL KB can be easily translated to a solution of the original problem. As a consequence, modern-day PL SAT solvers operate very efficiently in many practical cases. Although the general problem of satisfiability in PL is NP-complete in the number of variables, the best practical SAT solvers applied to moderately-sized problems can terminate in minutes and hours, not years.

However, PL is not a sufficiently complex language for some problems. It is generally well understood that every representation language faces a trade-off between expressive power (which determines the subset of real-world applications that can be reasonably modeled in that language) and complexity (which determines, among other, the running time of the algorithms performing inference in that language). For example, PL has no notion of equality, a fundamental notion which is required in many applications, such as game playing (and specifically game playing with partial information such as Kriegspiel[1]), where one needs to assert and reason that two positions or game pieces are equal.

A natural solution is to use a language with more expressive power, and the usual choice for many applications is FOL. The price of much higher expressive power, however, is complexity of algorithms; even though reasoning algorithms for FOL exist ([2] is one example), they are less efficient than PL SAT solvers. Even assuming the best progress in improving the algorithms, theoretical results challenge for FOL inference as FOL, unlike PL, is only semi-decidable.

$GL^=$ can be seen as a compromise between complexity and expressive power. Many problems that require equality do not, however, need the full expressiveness of FOL and can be well-expressed in $GL^=$. Having strictly more expressive power than PL (any PL KB can be easily translated to an equivalent KB in $GL^=$ by choosing a constant for each proposition in L of PL; choosing no predicates; and fixing $\{true, false\}$ as the domain) but still being sufficiently simple to permit efficient inference, $GL^=$ establishes itself at a favorable level of formalism.

In this paper, we explore the inference in $GL^=$, and particularly solving SAT problems. Rather than introducing a brand-new SAT solving routine in $GL^=$, we will investigate algorithms for translating arbitrary KBs in $GL^=$ into "equivalent" KBs in PL, where equivalence is defined as preserving satisfiability. That is, the resulting KB in PL will be satisfiable if and only if the original KB in $GL^=$ was satisfiable. Having an efficient algorithm for performing such translation easily leads us to solving SAT in $GL^=$ by utilizing highly efficient, off-the-shelf PL SAT solvers. This approach has an added benefit of automatically improving as SAT solvers improve, and thus can be seen as gaining power "for free" as the field of SAT solving progresses (whereas a new $GL^=$ SAT algorithm would have to be manually improved to include any new ideas the SAT solving community introduces).

To accomplish the conversion, we first describe a "naive" encoding of $GL^=$ KBs into PL. The encoding is naive only in the sense of being natural, and thus not very efficient; its correctness, however, is proved. The last section describes the main contribution of this paper: an advanced encoding algorithm that employs divide-and-conquer approach while using the naive encoding as a subprocedure. Our proposed algorithm has an added advantage of being independent of the actual encoding subprocedure used, so long as that subprocedure is satisfiabilty-equivalent (in the sense defined later). As such, our algorithm will benefit from any improvement to the underlying procedure without any explicit change being necessary.

The rest of the paper is organized as follows. In Section 2, we give a brief overview of fundamentals associated with the topic at hand. Section 3 proposes a simple yet effective conversion algorithm, proves its correctness, and then analyzes its efficiency. Section 4, which form the core of this paper, introduces the more efficient algorithm mentioned above, which functions by fragmenting the input and using the previously formulated algorithm on the fragments. We prove that this algorithm is correct - that is, even though some information appears to be missing from the encoding (which fact is chiefly responsible for the lower encoding size and thus higher SAT solving efficiency), satisfiability is preserved.

Finally, Section 6 presents an overview of related work, examines our future plans, and provides a summary for this paper.

2 Fundamentals

2.1 Ground Logic with Equality

Ground Logic with Equality ($GL^=$) is a subset of FOL which excludes quantifiers and functions, but keeps constants and predicates, as well as equality. The only other element of FOL - variables - is not specifically excluded, but becomes synonymous with constants due to the exclusion of quantifiers. No additional restrictions are placed on the formulas or KBs.

In this section, we give formal definitions necessary to establish a framework within which the discussion of the rest of this paper will occur. All of these definitions are adapted from the corresponding definitions of FOL, with changes required to accommodate the lack of functions and quantifiers. In particular, it is worthwhile to mention the basic terms of FOL which are missing from $GL^=$.

Besides variables, three other terms of FOL are no longer applicable: free occurrence, sentences and terms. In FOL, a variable is said to occur free if it is not bound by a quantifier; since we don't have quantifiers, we don't distinguish between free and bound occurrences. A sentence is a special case of formula which contains no free variables; in $GL^=$, every formula is a sentence. Finally, a term of FOL is an entity which is interpreted as an element in the universe - that is, either a ground term (variable or constant), or a function symbol applied to terms. Because we do not have functions (or variables), our terms will always be constant symbols, and thus do not deserve a definition of their own.

We will use the standard definitions from formal logic, with their standard meanings. The same terms applied to $GL^=$ will be assumed to hold naturally derived meaning as well. We will talk about *languages*, *formulas* (and sets of formulas, which we will denote as *Knowledge Bases (KBs)*), *interpretations* and *models*, as well as *satisfiability* and *unsatisfiability*. The most important definitions are reproduced below.

Definition 1 (Syntax). *A formula in $GL^=$ with language L is obtained in one of the six following ways:*

a. $x = y$, *where* x *and* y *are constants.*
b. $P^k x_1 ... x_k$, *where* P^k *is a* k*-ary predicate in* L, *and for each* i, x_i *is a constant in* L.
c. $F_1 \vee F_2$, *where* F_1 *and* F_2 *are formulas.*
d. $F_1 \wedge F_2$, *where* F_1 *and* F_2 *are formulas.*
e. $\neg F$, *where* F *is a formula.*
f. (F), *where* F *is a formula.*

Definition 2 (Interpretation). *An interpretation in $GL^=$ with language L is a pair (A, β), where*

a. *A is an arbitrary set, also referred to as* domain *or* universe*.*
b. *β is a map (also referred to as an* assignment*) on L, defined as follows:*
   (a) *For each constant x from L, β assigns an element of A.*
   (b) *For each k-ary predicate P from L, β assigns a subset of A^k.*

Definition 3 (Semantics). *Let F be a formula in $GL^=$ with language L, and let I = (A, β) be an interpretation on L. Then F is said to* ***hold*** *(alternatively, to* ***be true****) under I if and only if:*

a. *$x = y$: $\beta(x) = \beta(y)$; that is, the assignment β maps x and y into the same element of the domain A.*
b. *$P^k x_1 ... x_k$: $(\beta(x_1), ..., \beta(x_k)) \in \beta(P^k)$; that is, the k-tuple obtained by mapping each of the predicate arguments is an element of the subset (of all k-tuples of the domain) to which the assignment maps the predicate.*
c. *The remaining cases are obvious and are omitted here for space.*

If F does not hold, it is also said to ***be false****.*

3 Naive Conversion

In this section, we explore the naive conversion of a $GL^=$ KB into a PL KB. Note that "naiveness" applies only to efficiency; the correctness of the conversion is preserved. We first describe the algorithm and then state and prove the correctness theorem.

3.1 Algorithm

The basic premise of the conversion is to associate a proposition with every *instance* of a predicate or equality. This in particular means that the original KB determines the language of the resulting PL KB. The increased expressive power of $GL^=$ compared to PL is responsible for this increase in complexity.

Our final goal is to find an efficient (that is, generating as small an output as possible) conversion that preserves satisfiability. We define this precisely.

Definition 4 (Conversion Algorithm). *A conversion algorithm is one that accepts as input a KB in $GL^=$ and outputs a KB in PL. Alternatively, a conversion function is one with domain of all KBs in $GL^=$ and codomain of all KBs in PL.*

Definition 5 (Satisfiability Equivalence for Formulas and Algorithms). *We define satisfiability equivalence for both formulas and algorithms.*

a. *Given a $GL^=$ KB Φ and a PL KB Φ′, we call Φ and Φ′* ***satisfiability equivalent****, or* ***SAT-EQ****, if Φ and Φ′ are either both satisfiable, or both unsatisfiable.*

b. *A conversion algorithm A with the property that, for every GL$^=$ KB Φ, whenever A(Φ) = Φ', then Φ and Φ' are satisfiability equivalent, is satisfiability equivalent.*

We define the translation of a single formula first.

Definition 6 (Naive Translation). *Let ϕ be a formula in GL$^=$ with language L. The Naive Translation of ϕ, NT(ϕ), is a formula ϕ' in PL with language L$'$ defined on the structure of ϕ as follows:*

a. *$x = y$: A new proposition $\overline{EXY}$, which is created and added to L$'$ if it is not already there.*
b. *$P^k x_1 ... x_k$: A new proposition $\overline{PX_1...X_k}$, which is created and added to L$'$ if it is not already there.*
c. *$F_1 \vee F_2$: NT(F_1) $\vee$ NT(F_2).*
d. *$F_1 \wedge F_2$: NT(F_1) $\wedge$ NT(F_2).*
e. *$\neg F$: $\neg$NT(F).*
f. *(F): (NT(F)).*

As defined above, L$'$ contains a unique proposition for each instance of equality and for each instance of predicate in ϕ.

If the original formula does not contain equality, this translation is indeed a satisfiability-equivalent algorithm. However, as soon as equality is introduced, this property no longer holds. To illustrate this, consider a simple counterexample with a GL$^=$ KB whose language contains one binary predicate and three constants, consisting of three formulas: $\{Pxy, \neg Pxz, y = z\}$. Clearly, this KB is UNSAT (any interpretation of this KB would have to map y and z to the same element in its domain A, and then the pairs (x, y) and (x, z) would have to be mapped to the same pair in A^2; but that pair cannot both belong and not belong to a subset of A^2 to which the interpretation maps P). However, Naive Translation of this KB produces a PL KB containing three propositions $\{\overline{PXY}, \neg\overline{PXZ}, \overline{EYZ}\}$ *with no additional constraints* between them. This KB is satisfiable, for example by an assignment $\{\overline{PXY} \rightarrow true, \overline{PXZ} \rightarrow false, \overline{EYZ} \rightarrow true\}$.

In order to preserve satisfiability, we need to encode additional constraints intepreting equality in propositional logic. In general, there are three ways of dealing with equality in inference ([3]): adding additional formulas, using special rules (such as demodulation and paramodulation), and modifying the inference rule to be informed of equality (such as by introducing superposition into rule calculus). While the last two ways are useful for general inference, it is not immediately clear how to implement either of them for conversion into PL. Therefore, we will explicitly add formulas that capture the precise meaning of equality.

Definition 7 (Equality Semantics). *Let L be a GL$^=$ language. Then the Equality Semantics (ES) of L is a set of GL$^=$ formulas defined as the union of:*

a. *For every constant x in L: $x = x$.*
b. *For every pair of constants x, y in L: $x = y \rightarrow y = x$.*

c. *For every triple of constants x, y, and z in L:* $(x = y \wedge y = z) \rightarrow (x = z)$
d. *For every k-ary predicate P^k and for every $2k$-tuple of variables (a_1, ..., a_k, b_1, ..., b_k) in L:*

$$((a_1 = b_1) \wedge ... \wedge (a_k = b_k)) \rightarrow (P^k a_1 ... a_k \leftrightarrow P^k b_1 ... b_k) \tag{1}$$

Note that the formulas which capture the notion of equality do not depend on the contents of our KB, but only on the language. Nor are these formulas part of the KB (although adding them will not affect satisfiability); instead, they will be translated to PL using the mechanism defined above (NT) and added to the PL KB.

Definition 8 (Naive Conversion). *Let Φ be a KB in GL$^=$ with language L. The Naive Conversion of Φ, NC(Φ), is a KB Φ' in PL defined as:*

$$\Phi' := NT(\Phi) \cup NT(ES(L)) \tag{2}$$

It is instructive to revisit the earlier counterexample and ensure that it no longer holds, which we will not do for space reasons.

3.2 Correctness

We now state and prove the main result of this section: the conversion defined above preserves satisfiability. The proof will be an immediate application of two analogous results. Given an interpretation in either GL$^=$ or PL, it is possible to construct an interpretation in PL (resp. GL$^=$) for which an important property holds: if the original interpretation was a model for any KB in GL$^=$ (resp. PL), then the constructed interpretation will also be a model for a KB in PL (resp. GL$^=$) related to the original interpretation by NC.

Theorem 9 (Naive Conversion is SAT-EQ). *Let Φ be a KB in GL$^=$ with language L, and let Φ' be NC(Φ). Then Φ is satisfiable if and only if Φ' is satisfiable.* ***That is, NC is SAT-EQ.***

Proof. [1] The proof has two parts, each of which is further subdivided into two sections. In first part, we first prove that SAT $\Phi \rightarrow$ SAT Φ', and in the second part we prove the converse. Within each part, our proof will proceed as follows: Given that a KB is SAT, there must exist an interpretation under which it holds. We then (a) construct an interpretation for the other KB and (b) show that the other KB holds under that interpretation. For space reasons, we only show the construction in each case.

[**SAT $\Phi \rightarrow$ SAT Φ'**] Assume that Φ is satisfiable. Then there exists an interpretation $I := (A, \beta)$ under which Φ holds. We will construct the PL assignment β' and show that Φ' holds under this assignment. We define β' as follows:

[1] Due to space concerns, we omit some proofs, and only show the brief outlines of the others. Full versions of all proofs of this paper are available from the authors.

a. For every proposition $\overline{EXY}$ of the first kind, β' assigns *true* if $\beta(x) = \beta(y)$, and $false$ otherwise.
b. For every proposition $\overline{PX_1...X_k}$ of the second kind, β' assigns *true* if $(\beta(x_1), ..., \beta(x_k)) \in \beta(P)$, and $false$ otherwise.

It is now possible to show that Φ' holds under β'; the precise argument is omitted for space.

[SAT Φ' → SAT Φ] Assume that Φ' is satisfiable. Then there exists an assignment β' under which Φ' holds. We will construct the GL$^=$ interpretation I = (A, β) and show that Φ holds under this interpretation.

To define A, we need an intermediate result that will later help in our proof.

Lemma 10. *Let L_{const} be the subset of L containing precisely all constants of L. Let R be a binary relation defined on L_{const} as follows: for any constants a and b from L_{const}, $R(a, b)$ holds if and only if $\beta'(\overline{EAB}) = true$. Then R is an equivalence relation.*

Proof. Omitted for space; a simple argument showing reflexivity, symmetry and transitivity directly suffices. □

As an equivalence relation, R partitions L_{const} into equivalence classes. Let N be the total number of those classes (where $1 \leq N \leq |L_{const}|$). We define the domain A to be {1, 2, ..., N}, the set of positive integers from 1 to N, inclusive. Further, we associate with every equivalence class a unique number between 1 and N in any deterministic way. We now define β on L_{const} as follows: for every constant a, β assigns to a the unique number corresponding to the (unique) equivalence class to which a belongs. Note that this is well-defined, because of the properties of equivalence classes (namely, that a belongs to precisely one equivalence class).

It remains to define β for predicates. Let P be a k-ary predicate of L. Then we define β as follows:

$$\beta(P) := \{(\beta(a_1), ..., \beta(a_k)) \mid \beta'(\overline{PA_1...A_k}) = true\} \tag{3}$$

That is, β defines P to contain precisely all those k-tuples of elements of A for which the corresponding predicate of L' is assigned *true* by β'.

This completes the construction of interpretation $I = (A, \beta)$ for L. It remains to show that I is a model of Φ; this is omitted for space. □

We conclude this subsection by noticing two results that we have used in the proof above, and extracting them to form separate lemmas. Both of these results will be useful further in discussing Craig's Interpolation applications. The correctness of both lemmas follows easily from the generality (independence of particular Φ and Φ', other than the relation $\Phi' = NC(\Phi)$ connecting them) of construction and proof of the respective part of Theorem 9; the proofs are omitted for space.

Lemma 11. *Let $I := (A, \beta)$ be a GL$^=$ interpretation. Then it is possible to construct a PL assignment β' such that, for any KB Ψ in GL$^=$, if I is a model of Ψ, then β' is a model of $NC(\Psi)$.*

Lemma 12. *Let β' be a PL assignment. Then it is possible to construct a $GL^=$ interpretation $I := (A, \beta)$ such that, for any KB Ψ in $GL^=$, if β' is a model of $NC(\Psi)$, then I is a model of Ψ.*

3.3 Analysis

The theorem's statement and proof together form the first complete method by which we can reach the goal of this paper: given a KB in $GL^=$, construct a SAT-EQ KB in PL. While the method is correct, it is not optimal. Recall that our motivation was the ability to solve SAT in $GL^=$ using off-the-shelves PL SAT solvers. Such solvers vary in efficiency, but invariably depend in the execution time on the number of propositions in the PL KB. Thus, a measure of the efficiency of any conversion algorithm can be presented by computing the number of PL propositions this algorithm generates, as a function of its input.

Definition 13. *Let C be a conversion algorithm. The size of C, denoted $Size(C)$, is a function which accepts the number of predicates m and the number of constants n, and outputs the number of propositions which C generates from an input having m predicates and n constants.*

Suppose that the input $GL^=$ KB contains n constants and m predicates. We wish to estimate $Size(NC)$; that is, rather than arriving at a precise number, we wish to obtain a complexity bound for the number of propositions generated. By examining the construction described above (with detailed treatment omitted for space), we can conclude that the upper bound on $Size(NC)$ is $n^2 + mn^k$. In a particular case of having only unary and binary predicates, this bound can be further simplified to $O(mn^2)$.

4 Advanced Conversion by Partitioning

The previous section presented NC, a method to encode arbitrary $GL^=$ KBs, and proved its correctness. Having analyzed the perfomance of NC, we will now attempt to devise a more efficient procedure which uses the presented method as a subroutine.

4.1 Motivation

While NC is correct, it generates a propositional encoding with size of $O(n^2 + mn^k)$. Since the size of a conversion algorithm is defined as the number of propositions it can generate, and since modern SAT solvers are designed so that their efficiency depends greatly on the number of propositions in the input, we would like to decrease the size of a conversion method while still ensuring its correctness. In the worst case, a SAT solver will take time exponential[2] in the number

[2] Provided, of course, that $P \neq NP$.

of propositions, and hence, for NC, exponential in both n and m and super-exponential in k[3]

The above, however, is insufficient motivation for devising a better method, which would use NC as a subprocedure, since we have so far only enumerated the deficiencies of NC. The key observation that allows a better method to be devised is that NC produces some extraneous propositions - those that are not necessary to propagate the semantics of satisfiability. We will attempt to remove as many as possible of those extraneous propositions while not violating the satisfiability equivalence of our conversion algorithm.

4.2 Description

The concept of dividing the input into parts and applying some transformation to those parts has long been among the standard tools for fighting complexity; see, for example, [5] and [6]. The principal notion is that, even if the underlying transformation is still exponential in the size of its input, applying that transformation to each of the partitions dramatically decreases that "size of its input". We will adapt a simple algorithm: Given an input $GL^{=}$ KB, instead of applying a conversion procedure[4] to that entire KB, we will instead partition that KB into two fragments, and apply the conversion procedure to each in turn. The result, then, will be the union of the obtained results.

The primary potential concern is the possible loss of information resulting from severing the connections between entities which are assigned to different partitions. Intuitively, it might seem that such an algorithm will not provide for a satisfiability-equivalent conversion, because some information will be lost - specifically, it might be that while the input KB itself was unsatisfiable, the two segments are satisfiable by themselves, and that their translations will also be satisfiable, so that the union of the translations will fail to preserve the unsatisfiability of the original KB. However, this reasoning fails to account for the extra information generated by NC - specifically, that generates by equivalence semantics processing. Indeed, in what follows we will show that the mere fragmentation, without any extra reasoning (other than the conversion procedure) being performed on either of the fragments, is sufficient to maintain satisfiability equivalences.

---

[3] The latter is not as bad as it may seem, because typically k is bounded by a very small number (in fact, the case of k bounded by 2 - that is, the case of having only unary and binary predicates - is of importance by itself. For the case of k bounded by 1, and thus having only unary predicates, see [4]). However, the former may act as a severely limiting factor, since either n or m, or both, can be large in practical applications.

[4] We will be using NC (the only conversion procedure we have so far presented), but notice that this algorithm does not depend on the choice of a particular conversion procedure. Indeed, this is an explicit strenght of this algorithm (and of the divide-and-conquer algorithms in general), because any improvements to the underlying conversion procedure will be benefited from without any need to change our algorithm, that is, "for free")

We conclude this subsection by formalizing our algorithm and stating (but not yet proving) the associated correctness theorem.

Our precise algorithm for converting a GL$^=$ KB will thus be as follows:

a. Given a GL$^=$ KB Φ:
b. Separate Φ into two fragments, $NC(\Phi_1)$ and $NC(\Phi_1)$[5].
c. Run the conversion algorithm on the first fragment.
d. Run the conversion algorithm on the second fragment.
e. Join the results.

Thus, instead of computing $NC(\Phi)$, we will be computing $NC(\Phi_1) \cup NC(\Phi_2)$. We need, of course, to show that this is satisfiability-equivalent. We thus formulate the result, which we hold to be the most important single contribution of this paper, and which we will prove in the following subsection.

Theorem 14. *Let Φ be a GL$^=$ KB partitioned into Φ_1 and Φ_2. Let Φ' be a PL KB obtained as follows:*

$$\Phi' := NC(\Phi_1) \cup NC(\Phi_2) \tag{4}$$

Then, Φ and Φ' are satisfiability-equivalent (Definition 5). Equivalently, a conversion algorithm that applies (4) is satisfiability-equivalent.

We will develop the proof in the following subsections, after having introduced the tools that we will use.

4.3 Craig's Interpolation

Craig's theorem, first published in [7], is a classical result that forms one of the bases of many "divide-and-conquer" approaches in FOL and its fragments. The original result has been extended several times; however, the basic version will be satisfactory for our purposes. The key property, which will be useful for us in reducing the size of the conversion (our entire reason for even attempting to find a conversion method different from the original NC), is that this message will have a potentially greatly reduced language compared to that of each fragment. The Craig's interpolants, thus, will have only those predicates and constants which appear in both fragments.

We state the form we will use here for future reference.

Proposition 15. *Let Φ_1 and Φ_2 be two GL$^=$ KBs. Then there exists a GL$^=$ KB Φ, referred to as a **Craig's interpolant**, with the following properties:*

a. $\Phi_1 \models \Phi$.
b. If Φ_1 is inconsistent with Φ_2, then Φ is also inconsistent with Φ_2.
c. The language of Φ is the intersection of the languages of Φ_1 and Φ_2. That is, Φ contains only those predicates and only those constants that appear in both Φ_1 and Φ_2.

[5] We leave the precise way of doing this unspecified for now. In the case of two fragments, which is being considered here, any fragmentation is sufficient.

Proof. Craig's Interpolation for FOL is discussed in great detail in [7]. The only concern here is that, since the original result is formulated for FOL, we need to ensure that the interpolants of $\mathrm{GL}^{=}$ formulas can be expressed in $\mathrm{GL}^{=}$; that is, that the interpolants will not contain quantifiers so long as the input itself does not. This is addressed in [8]. □

4.4 Craig's Interpolation and Partitioning

We will state and prove some auxiliary results first. The following result, which postulates that NC respects entailment, forms the core of our argument.

Lemma 16. *Let Φ_1 and Φ_2 be $GL^{=}$ KBs. If $\Phi_1 \models \Phi_2$, then $NC(\Phi_1) \models NC(\Phi_2)$.*

Proof. Let β be an arbitrary assignment that satisfies $NC(\Phi_1)$. We need to show that β also satisfies $NC(\Phi_2)$. Let $M = (A, \alpha)$ be a $\mathrm{GL}^{=}$ interpretation constructed from β using the process described in the second half of Theorem 9. By Lemma 12, since $\beta \models NC(\Phi_1)$, $M \models \Phi_1$. By hypothesis, $\Phi_1 \models \Phi_2$, so $M \models \Phi_2$ as well. Let β' be a PL assignment constructed from M using the process described in the first half of Theorem 9. By Lemma 11, since $M \models \Phi_2$, $\beta' \models NC(\Phi_2)$.

We will now show that β and β' agree on $L(NC(\Phi_2))$; that is, that for every proposition in the language of $NC(\Phi_2)$, either both β and β' assign *true*, or both assign *false*. Intuitively, this is not at all surprising, for even though β was an arbitrary assignment, it is related to β' in that the latter was created from M which in turn was created from the former. If both constructions are reasonable, it is not unnatural to expect that they are "reversible" in some sense, and that β and β' will indeed agree on all of the propositions we are interested in.

Because $NC(\Phi_2)$ was created by applying NC, as described in Definition 8, the only propositions its language may contain are those created by NT, as described in Definition 6. Specifically, it may only contain propositions of two forms, which we treat in order. For both forms, we consider the only two possibilities for the truth value β' assigns to that proposition, and for each such possibility we "unroll" the two construction, first deducing what must have been true about M for β' to obtain the value that it has; and then deducing what must have been true about β for M to obtain the value that we have deduced it must have. In all cases, we will show that the truth value that β assigns to the proposition in question must be the same as that assigned by β'.

a. Propositions of form $\overline{EAB}$:
   (a) If β' assigns *true* to $\overline{EAB}$, then (by construction of β' from M as described by the first point of the first half of Theorem 9), it must have been the case that $\alpha(a) = \alpha(b)$. But then, because α was constructed from β by the procedure described by the second half of Theorem 9, it must have been the case that a and b were in the same equivalence class with respect to the binary relation R defined in Lemma 10, so $R(a, b)$ must have held. However, because of the way in which R was defined, this requires that β must have assigned *true* to $\overline{EAB}$. Thus, in this case, β and β' indeed agree on the proposition in question.

(b) If β' assigns *false* to $\overline{EAB}$, then the argument is similar to the one above, and is omitted for space.

b. Propositions of form $\overline{PX_1...X_k}$:

(a) If β' assigns *true* to $\overline{PX_1...X_k}$, then (by construction of β' from M as described by the second point of the first half of Theorem 9), it must have been the case that the k-tuple $(\alpha(x_1), ..., \alpha(x_k))$ belongs to $\alpha(P)$, the intepretation of predicate P under α. But $\alpha(P)$ is defined to include precisely those k-tuples of elements in A for which the corresponding predicate is assigned *true* by β (by construction of second half of Theorem 9). That is, $\alpha(P)$ has been defined to be $\{(\alpha(a_1), ..., \alpha(a_k))|\beta(\overline{PA_1...A_k}) = true\}$. Because we have deduced that $(\alpha(x_1), ..., \alpha(x_k))$ belongs to $\alpha(P)$, it must then have been the case that β assigns *true* to the proposition $\overline{PX_1...X_k}$. Thus, in this case, β and β' indeed agree on the proposition in question.

(b) If β' assigns *false* to $\overline{PX_1...X_k}$, then the argument is similar to the one above, and is omitted for space.

We have shown that β and β' assign the same truth value to all propositions occuring in $NC(\Phi_2)$. Thus, β and β' must either both satisfy or both fail to satisfy $NC(\Phi_2)$. Since we have concluded above that $\beta' \models NC(\Phi_2)$, we can now conclude that $\beta \models NC(\Phi_2)$. But β was an arbitrary assignment that satisfies $NC(\Phi_1)$; we have shown that it also satisfies $NC(\Phi_2)$. Thus, any model of $NC(\Phi_1)$ is also a model of $NC(\Phi_2)$, and so $NC(\Phi_1) \models NC(\Phi_2)$, completing the proof. □

We will also make use of the following result, which intuitively states that NC respects basic contradiction notions. While we will only use the result once in the following theorem, it is general enough to deserve being stated externally.

Lemma 17. *For any GL$^=$ KB Ψ, $NC(\Psi) \cup NC(\neg\Psi)$ is unsatisfiable.*

Proof. Assume to the contrary. Then let Ψ be such a GL$^=$ KB. Since it is satisfiable, it has a model; call it M. Since $M \models NC(\Psi) \cup NC(\neg\Psi)$, M is a model for both $NC(\Psi)$ and $NC(\neg\Psi)$ (a union of formulas is their conjunction, and thus an assignment which satisfies the union must satisfy all individual formulas in the conjunction). Let M' be a GL$^=$ interpretation obtained by using the construction described in the second part of Theorem 9. By the result shown there, M' will satisfy a GL$^=$ KB Φ if M satisfies $NC(\Phi)$. Because $M \models NC(\Psi)$, $M' \models \Psi$; and because $M \models NC(\neg\Psi)$, $M' \models \neg\Psi$. We now have $M' \models \Psi \cup \neg\Psi$, a contradiction[6], showing that our assumption was incorrect, which completes the proof. □

We are now ready to prove Theorem 18, which we restate here for reference.

[6] Note that we do not need to show that this is a contradiction in GL$^=$, because we know it to be a contradiction in FOL, and $\Psi \cup \neg\Psi$ is an FOL formula, whereas M' is an FOL interpretation.

Theorem 18. *Let Φ be a GL$^=$ KB partitioned into Φ_1 and Φ_2. Let Φ' be a PL KB obtained as follows:*

$$\Phi' := NC(\Phi_1) \cup NC(\Phi_2) \tag{5}$$

Then, Φ and Φ' are satisfiability-equivalent (Definition 5). Equivalently, a conversion algorithm that applies (5) is satisfiability-equivalent.

Proof. It is easy to show that SAT Φ implies SAT Φ'. We show the other implication; that is, that SAT Φ' implies SAT Φ. We will show the contrapositive. Assume UNSAT Φ; that is, UNSAT $\Phi_1 \cup \Phi_2$. By Craig's Theorem (Proposition 15), there exist some GL$^=$ KB γ such that:

a. $\Phi_1 \models \gamma$;
b. Φ_2 is inconsistent with γ, from which we can conclude
c. $\Phi_2 \models \neg\gamma$; and
d. The language of γ is the intersection of the languages of Φ_1 and Φ_2.

Applying Lemma 16 to (a), we conclude (a') $NC(\Phi_1) \models NC(\gamma)$, and applying it to (c), we conclude (b') $NC(\Phi_2) \models NC(\neg\gamma)$. Assume, for contradiction, that SAT $NC(\Phi_1) \cup NC(\Phi_2)$). Then $NC(\Phi_1) \cup NC(\Phi_2)$ has a model, say M. Because M is a model of the union (conjunction) of formulas, it must also be a model for any subset of those formulas; thus, (c') $M \models NC(\Phi_1)$ and (d') $M \models NC(\Phi_2)$. From (a') and (c'), we can immediately conclude $M \models NC(\gamma)$, and from (b') and (d'), we can immediately conclude $M \models NC(\neg\gamma)$. Combining these results, we obtain $M \models NC(\gamma) \cup NC(\neg\gamma)$, so in particular SAT $NC(\gamma) \cup NC(\neg\gamma)$, which contradicts Lemma 17. Thus, our assumption must have been incorrect, which completes the proof. □

The theoretical results achieved in this subsection allow us to create an intuitively significantly more efficient conversion method than the Naive Conversion, because we are applying the costly algorithms to smaller inputs. In addition to being a powerful result in the context of encoding GL$^=$ formulas, it is an intriguing theoretical result in its own right - one would not immediately expect an particular conversion algorithm to behave properly when applied to the fragments of its input.

5 Conclusion

In this section, we will describe related work, future work, and formulate a summary for this paper.

5.1 Related Work

A similar problem is solved for a different fragment of FOL in [4]. That work considers monadic FOL, which restricts predicates to being unary, but allows quantifiers. Our work applies to a fragment that is simultaneously more restrictive (since we disallow quantifiers), and less restrictive (we allow arbitrary arity of quantifiers; indeed, Section 3.3 provides an upper bound of the size of the output as a (exponential) function of the predicate arity).

5.2 Future Work

In our future work, we plan to build on the achieved results in several ways. The most obvious extension is to advance the partitioning to work recursively, so that the input KB can be split into arbitrary large number of fragments, which are then encoded using NC. Because, as Section 3.3 has shown, the size of the output KB is directly dependent on size of the input $GL^{=}$ KB (and hence the running time of the final PL SAT solver is highly dependent on that size), employing NC on only small fragments can decrease the size of the resulting KB tremendously, thus achieving considerable saving in the SAT solver running time. The recursive fragmentation will be required to maintain the so-called running intersection property[6]; we plan to use a partitioning program already developed by one of us [9] for this purpose.

Noting that the "base case" conversion (NC in our case) is orthogonal to the partitioning mechanism, we also plan to improve the conversion algorithm to generate smaller-sized KBs. NC generates a significant amount of information, which represents a plethora of internal connections between the formulas of the original KB; indeed, it is such richness that made our final result (Theorem 18) possible. However, some of that information is extraneous for each particular case; specifically, some of the equivalence semantics formulas generates may not be required or even used in concrete cases. Any improvement in the underlying algorithm will result in propagated efficiency improvements in the complete application, because that underlying algorithm is invoked several times (two times in the described method, and much more in the recursive fragmentation extension proposed above, since NC will be used on every partition).

5.3 Summary

We have introduced $GL^{=}$, a decidable fragment of FOL. We have argued that $GL^{=}$ enjoys a unique position as being sufficiently expressive yet sufficiently simple, a position not shared by either general FOL (which has more expressive power, but suffers from inefficient deciding algorithms) or PL (which enjoys a plethora of very efficient SAT solving methods, but lack many constructs of FOL and thus can be hard to use for a particular application). In this context, we have formulated and throughly investigated a solution to general satisfiability problem by converting an arbitrary input knowledge base into a PL knowledge base, mandating that such conversion does not change satisfiability. We have presented several increasingly complex and increasingly efficient methods for such conversions, starting from Naive Conversion (which translated the input KB directly, ensuring both predicate and equality instance translation, and the equality semantics translation), to using divide-and-conquer partitioning paradigm in conjunction with Craig's Lemma in two different ways: both by putting the computed interpolants of one fragment into the resulting encoding, and by using Craig's Lemma to prove the ultimate contribution of this paper - Theorem 18. For each of these conversion methods, we have formulated the algorithm itself and proved that it preserves satisfiability. Specifically, Theorem

18 illustrates a very interesting theoretical result, which may prove to be useful beyond this paper - that an input KB can be divided into two fragments, which can then be encoded individually, and the union of the resulting encodings is satisfiability equivalent to the encoding of the entire KB. We believe that this theoretical result, besides its obvious applicability for the purposes of encoding a $GL^{=}$ KB into PL, serves as an important example of the utility of the general divide-and-conquer paradigm.

References

1. Megan Nance, Adam Vogel, E.A.: Reasoning about partially observed actions. AAAI (2006)
2. Baumgartner, P.: FDPLL – A First-Order Davis-Putnam-Logeman-Loveland Procedure. In McAllester, D., ed.: CADE-17 – The 17th International Conference on Automated Deduction. Volume 1831 of Lecture Notes in Artificial Intelligence., Springer (2000) 200–219
3. Russel, S., Norvig, P.: Artificial Intelligence: A Modern Approach. 2nd edn. Prentice Hall (2003)
4. Ramachandran, D., Amir, E.: Compact propositional encodings of first-order theories. AAAI (2005) 340–345
5. Amir, E.: Dividing and Conquering Logic. PhD thesis, Stanford University (2002)
6. Amir, E., McIlraith, S.: Partition-based logical reasoning for first-order and propositional theories. Artificial Intelligence **162**(1-2) (2005) 49–88
7. Craig, W.: Three uses of the herbrand-gentzen theorem in relating model theory and proof theory. Journal of Symbolic Logic **22**(3) (1957) 269–285
8. McMillan, K.L.: An interpolating theorem prover. Theor. Comput. Sci. **345**(1) (2005) 101–121
9. Amir, E.: Partitioning version 1.2. Technical report, Stanford University (2002) Software description available at http://reason.cs.uiuc.edu/eyal/decomp/README.