Deciding Effectively Propositional Logic with Equality

Ruzica Piskac
EPFL Lausanne
ruzica.piskac@epfl.ch

Leonardo de Moura
Microsoft Research
leonardo@microsoft.com

Nikolaj Bjørner
Microsoft Research
nbjorner@microsoft.com

December 7, 2008

Technical Report
MSR-TR-2008-181

Microsoft Research
Microsoft Corporation
One Microsoft Way
Redmond, WA 98052

Deciding Effectively Propositional Logic with Equality

Ruzica Piskac
EPFL Lausanne
ruzica.piskac@epfl.ch

Leonardo de Moura
Microsoft Research
leonardo@microsoft.com

Nikolaj Bjørner
Microsoft Research
nbjorner@microsoft.com

December 7, 2008

Abstract

Effectively Propositional Logic (EPR), also known as the Bernays-Schönfinkel class, allows encoding problems that are propositional in nature, but EPR encodings can be exponentially more succinct than purely propositional logic encodings. We recently developed a DPLL-based decision procedure that builds on top of efficient SAT solving techniques to handle the propositional case efficiently while maintaining the succinctness offered by the EPR representation. To achieve the effect, it uses sets of substitutions encoded as binary decision diagrams [5]. It is possible to reduce EPR formulas with equality to pure EPR, but the reduction requires adding axioms for equality and congruence. This approach potentially increases the search space and could defeat the efficiency we are aiming to achieve. We here provide a calculus and decision procedure that handles equality natively. The procedure builds in equality propagation, and allows reducing dependencies on equalities during conflict resolution.

Contents

1 Introduction

Pure Effectively Propositional Logic is a fragment of first-order logic. The satisfiability problem for pure EPR is to determine satisfiability for formulas of the form

$$\exists \vec{x} \forall \vec{y} \varphi(\vec{x}, \vec{y})$$

where φ is a quantifier free, and atomic sub-formulas range over un-interpreted relations.

The satisfiability problem for EPR formulas can be reduced to SAT by first replacing all existential variables by skolem constants, and then grounding the universally quantified variables by all combinations of constants. This process produces a propositional formula that is exponentially larger than the original. In a matching bound, the satisfiability problem for EPR is NEXPTIME complete [11]. Ramsey [13] [2] established that the satisfiability problem for EPR remained decidable when adding equality. The more celebrated result of the decidability proof is now known as Ramsey's theorem. The finite counter-part has, *Ramsey theory*, remains an active field in combinatorics [9].

It is rather simple to encode EPR with equality into EPR without equality. Section 3.2 illustrates the embedding. The encoded equality axioms are Horn, and they can be applied by unit-propagation only. Thus, literals occurring in equality axioms need not take part in splits. The added complexity of handling equality arises from the potential of having to propagate equality explicitly. In other words, every time an equality is added to the current set of assumptions, every assignment to literals should be closed under the added equality. Our calculus for EPR with equality addresses the closure of predicates under equality in a uniform way. It avoids explicit equality axioms, and propagates equalities implicitly to all other predicates. The implicit equality propagation requires in return an explicit apparatus for handling equalities during conflict resolution. That is, when producing auxiliary conflict clauses, and performing backjumping during the DPLL search.

Before we continue with the main subject of this paper, which is to formulate a calculus for EPR with equality, we will first motivate the use of EPR using a set of examples ranging form direct to indirect applications of EPR.

2 Examples

Example 1 (Orders)**.** *Problems arising from program verification often involve establishing facts of quantifier-free formulas, but the facts themselves use relations and functions that are conveniently axiomatized using a background theory that uses quantified formulas. One set of examples of this situation comprise of formulas involving partial orders. The theory of partial orders, orders with tree-like properties, and other variants can be axiomatized in EPR with equality. We list a few axioms that can be combined in different ways to produce different theories of binary relations.*

Reflexivity	$\forall x \,.\, x \preceq x$
Anti−Symmetry	$\forall x, y \,.\, x \preceq y \wedge y \preceq x \rightarrow x \simeq y$
Transitivity	$\forall x, y, z \,.\, x \preceq y \wedge y \preceq z \rightarrow x \preceq z$
Tree	$\forall x, y, z \,.\, x \preceq z \wedge y \preceq z \rightarrow x \preceq y \vee y \preceq x$
Linearity	$\forall x, y \,.\, x \preceq y \vee y \preceq x$

Note that the above axioms do not include axioms for specifying dense linear orders, or discrete orders with an infinite number of elements.

Example 2 (Sets and Boolean Algebras)**.** *Constraints over sets (Boolean Algebras) can be encoded into EPR by treating sets as unary predicates and lifting equalities between sets as formula equivalence. For example,* $A \cup B \subseteq C$ *is represented as* $\forall x \,.\, A(x) \vee B(x) \rightarrow C(x)$*, and we can prove theorems, such as*

$$A \cup B \subseteq C \wedge B \setminus A \neq \emptyset \rightarrow A \subset C$$

by dually checking satisfiability of:

$$\begin{array}{rl} & \forall x \,.\, A(x) \vee B(x) \rightarrow C(x) \wedge \exists y \,.\, B(y) \wedge \neg A(y) \\ \wedge & \neg(\forall z \,.\, (A(z) \rightarrow C(z)) \wedge \exists u \,.\, C(u) \wedge \neg A(u)) \end{array}$$

Equality is useful when expressing properties, such as A *is a singleton:* $(\exists x \,.\, A(x)) \wedge (\forall x, y \,.\, A(x) \wedge A(y) \rightarrow x \simeq y)$*. Additional examples of set encodings in EPR have been explored in [8].*

Example 3 (The map property fragment)**.** *The Map Property Fragment [3] (for un-interpreted functions) is a Boolean combination of map properties and quantifier free formulas. The fragment captures several scenarios, including Boolean Algebras. A* map property *is a formula of the form:*

$$\forall i.G[i] \rightarrow F[a[i]] \tag{1}$$

where G *is an index guard,* i *is a set of bound universal variables (*uvar*),* F *is value constraint such that the bound variables* i *only appear in an array access* $a[i]$*. Nested array accesses with bound variables are not allowed. Instead array accesses have to occur only in equalities and disequalities. An index guard is a positive Boolean combination of arbitrary equalities; or disequalities between one constant* c *taken from some set* Σ*, and either a universal variable or a constant.*

$$\begin{array}{rcl} G & ::= & G \wedge G \mid G \vee G \mid atom & (2) \\ atom & ::= & var \simeq var \mid var \not\simeq c \mid c \not\simeq var & (3) \\ var & ::= & c \mid uvar & (4) \end{array}$$

As established in [3], the array property fragment is decidable. In fact every positive universal quantifier can be replaced by a finite conjunction of instantiations, where the instances range over the finite set of index variables *that can be extracted from the formula. The set of possible instantiations is exponential in the number of bound variables.*

One way of reducing the array property fragment to EPR is by anticipating that a finite amplification of an array property formula using the index set introduces a bounded set of array values of the form $a[i]$*, where* a *is an array and* i *is in the computed index set. We can then replace array access terms by a function that has a finite range. Finite range functions can be axiomatized using relations. So we use the steps:*

1. *Collect the set of arrays* $a_1, a_2, a_3, \ldots, a_n$*, and the set of index variables* $e_1, e_2, \ldots, e_m$*.*
2. *Introduce the set of values:* $v_{11} = a_1[e_1], a_1[e_2], \ldots, a_n[e_1], \ldots, v_{nm} = a_n[e_m]$*.*
3. *Replace every occurrence of* $a_i[e_j]$ *in the formula by* v_{ij}*.*
4. *For every array property* $\forall i.G[i] \rightarrow F[a[i]]$ *replace* $a[i]$ *by a fresh variable* w_{ai} *to form the formula* $\forall i, w_{ai} \,.\, G[i] \wedge select(a, i, w_{ai}) \rightarrow F[w_{ai}]$*.*

5. *Add either the axioms (5) and (6), or (6) and (7).*

$$\bigwedge_{1\leq j\leq n, 1\leq k\leq m} \mathit{select}(a_j, e_k, v_{jk}) \tag{5}$$

$$\forall a, i, v, w \;.\; \mathit{select}(a, i, v) \wedge \mathit{select}(a, i, w) \;\rightarrow\; v \simeq w \tag{6}$$

$$\forall a, i, \bigvee_{1\leq j\leq n, 1\leq k\leq m} \mathit{select}(a, i, v_{jk}) \tag{7}$$

The transformation allows to replace the function $a[i]$ by a relation. The resulting formula is in EPR and is polynomial in size of the original formula. The three axioms (5) and (6), and (7) imply that select *is set to $v_j k$ on arguments a_j and e_k, is functional, and is total. However, as suggested above we do not need all three properties of* select *in the context of the array fragment. If we just assume (5) and (6), then all ground occurrences of $a[i]$ are specified, and all occurrences under a quantifier is defined at least on the index set. This suffices for the array fragment. On the other hand, if we assume (6) and (7), then we are not specifying that $a_j[e_k]$ maps to v_{jk}, but in light of axiom (7) it must map to some constant among the v_{jk}, which suffices for satisfiability.*

Both (5) and (6) are relatively expensive because they require a product of arrays and indices. But (5) allows an incremental use of an EPR solver. Initially, only assert $\mathit{select}(a_j, e_k, v_{jk})$ if $a_j[e_k]$ occurs as a ground instance. Check for satisfiability. We can either fix models generated by the EPR solver, by adding functionality axioms when the models don't satisfy them, or we can use a different trick known from finite model finders: assert

$$\forall a, i \;.\; \bigwedge_{a_j[e_k]} (a \not\simeq a_j \vee e_k \not\simeq i) \;\rightarrow\; \mathit{select}(a, i, w_{new}) \vee \mathit{Ans}(a, i) \tag{8}$$

If $\mathit{Ans}(a, i)$ is non-empty, then add constants for (at least one) member(s) of it.

Example 4 (Arrays with Theories)**.** *The Map Property Fragment from the previous example 3 can be generalized for the case where the array formulas are integrated with* theories. *One such theory is the theory of discrete linear orders, in which case we obtain the array property fragment. An array property is a formula of the form (1), but this time atoms in index guards (the atoms in the formula G) are from a larger vocabulary that includes relations.*

$$\mathit{atom} \;::=\; \mathit{var} \simeq \mathit{var} \mid \mathit{var} \not\simeq c \mid c \not\simeq \mathit{var} \mid R(\mathit{var}, \ldots, \mathit{var}) \tag{9}$$

The value constraints can in this case be generalized to also use uninterpreted relations.

$$F \;::=\; F \wedge F \mid F \vee F \mid \mathit{val\_atom} \tag{10}$$

$$\mathit{val\_atom} \;::=\; \mathit{val} \simeq \mathit{val} \mid \mathit{val} \not\simeq \mathit{val} \mid R(\mathit{val}, \ldots, \mathit{val}) \mid \neg R(\mathit{val}, \ldots, \mathit{val}) \tag{11}$$

$$\mathit{val} \;::=\; a[\mathit{var}] \mid c \tag{12}$$

Let $\mathcal{I} = \{c_1, \ldots, c_n\}$ be a set of constants occurring in the formula φ, which is flattened into conjunctive normal form. If φ does not contain any constants, then introduce a dummy constant κ. Let T_R be a theory for R. For the sake of using this example for bootstrapping EPR theories, we could even assume that φ contains a set of axioms for T_R formulated within EPR. To simplify our treatment, let us also assume that R is a binary relation. Let $G_1, \ldots, G_m$ be the index guards in φ (all other clauses are in already within EPR) and to further make our lives simpler, also assume that the array property formulas contain at most one bound variable i. We then say that $\mathcal{I}$ covers *φ if (the EPR portion of) φ implies*

$$\forall i \;.\; \bigvee_{c\in\mathcal{I}} \bigwedge_{j=1}^{m} [G_j[i] \rightarrow G_j[c]] \tag{13}$$

Thus, whenever $G_j[i]$ holds for some i, then there is a constant in c, such that $G_j[c]$ holds, and such that the same constant satisfies all the other guards in φ that are satisfied by i. This property on index guards implies that $\mathcal{I}$ induces a congruence relation: variables are congruent if they satisfy the same set of index guards. Furthermore, every variable is congruent to some constant in $\mathcal{I}$. With the congruence relation we can form a choice function that for every i associates a constant $c \in \mathcal{I}$ to satisfy formula (13). For future reference, we will call the choice function $\texttt{proj}(i)$.

In this case, it suffices to instantiate the quantifiers using just the constants in $\mathcal{I}$. The justification for this restriction is as follows: Let M be a finite interpretation that satisfies the EPR fragment of $\widehat{\varphi}$ together with all the ground instances of the array fragment of φ. We may lift M to an interpretation $\widehat{M}$ of φ by interpreting $a[i]$ using the interpretation for $a[proj(i)]$, and we can interpret predicates in the value constraint, such as $R(a[i], b[j])$, using the value for $R(a[\texttt{proj}(i)], b[\texttt{proj}(j)])$. To validate that $\widehat{M}$ satisfies φ, notice that it satisfies the EPR fragment of φ, and whenever i satisfies guards G_1, G_2, G_3, then $\texttt{proj}(i)$ *satisfies the same guards. Since $\widehat{M}$ satisfies the entire formula, it must be the case that F_1, F_2, F_3 are satisfied by the values induced by the interpretations of the array access terms $a[i]$, $b[j], c[k], \ldots$.*

Our notion of the array property fragment is an abstraction of the presentation in [3]. There, the index guard may contain arithmetical inequality $\leq$, but not strict inequality. In the theory of integer arithmetic it is possible to replace strict inequality between a bound variable and a closed term by non-strict inequalities by adding or subtracting 1 from the closed term. In other words, the predicate $i < t$ where i is a bound variable and t is a closed term is equivalent to $i \leq t - 1$. The theory of linear orders ensures that for every index guard using a positive combination of $\leq$ there is some $c \in \mathcal{I}$ satisfying condition (13). Notice that if we use the theory of dense linear orders and replace non-strict inequality $\leq$ by strict inequality, $<$, then the property does not hold. For example, assuming a and b are the only variables in $\mathcal{I}$, then given the guard formulas $G_1[x] : a < x \wedge x < b$, $G_2[x] : x \not\simeq a$, $G_3[x] : x \not\simeq b$, neither a, nor b can be used for the case where i satisfies $G_1[i]$ (so i is strictly between a and b).

We here make the observation that the condition on index guards is not specific to the theory of linear orders. For example, if R (from now on written $\sqsubseteq$) satisfies distributive lattice axioms, then we can create the set $\mathcal{I}$ consisting of the closure of all constants in φ with respect to suprema and infima under R. In other words, if the constants $\{c_1, \ldots, c_n\}$ occur in $\mathcal{I}$, then create the finite set of at most 2^{2^n} elements closed under $\sqcap$ and $\sqcup$ (term over $\mathcal{I}$ using $\sqcup$ and $\sqcap$ can be written in disjunctive normal form each using up to 2^n conjunctions). If $G[x]$ is a positive Boolean combination of inequalities of the form $x \sqsubseteq c$, and x satisfies $G[x]$, then let $\mathcal{I}_u$ be the subset of $\mathcal{I}$ such that $x \sqsubseteq c$ for $c \in \mathcal{I}_u$, and let $\mathcal{I}_l$ be the subset of $\mathcal{I}$ such that $c \sqsubseteq x$, for $c \in \mathcal{I}_l$. Then $\sqcup\mathcal{I}_l \sqsubseteq x \sqsubseteq \sqcap\mathcal{I}_u$, and we can choose either bound (at least one of the sets $\mathcal{I}_l$ or $\mathcal{I}_u$ has to be non-empty) as the representative for x.

Example 5 (Finite domains and QBF)**.** *The satisfiability and validity problem for Quantified Boolean Formulas (QBF) is PSPACE complete. Bound variables in QBF formulas range over Boolean truth values. Formulas where bound variables range over a finite domain $\mathcal{D}$ can be more handy, but don't add expressive power. For example, we can reduce $\forall x \in \{0, \ldots, 7\} \,.\, \varphi[x \simeq 5]$ to $\forall x_0, x_1, x_2 \in \mathcal{B} \,.\, \varphi[x_0 \wedge \neg x_1 \wedge x_2]$. We can also directly reduce finite domain constraints to EPR. For example, consider the formula:*

$$\forall x \in \mathcal{D} \,.\, \exists y \in \mathcal{D} \,.\, \forall z \in \mathcal{D} \,.\, \varphi[x, y, z]$$

where we assume x (and y, z) appear in equalities of the form $x \simeq d$, where $d \in \mathcal{D}$, in φ. Skolemization produces:

$$\forall x, z \in \mathcal{D} \,.\, \varphi[x, f_y(x), z]$$

and we can relativize the quantifiers by introducing definitions for the finite domain:

$$\forall x, z \,.\, \mathbf{D}(x) \wedge \mathbf{D}(z) \;\rightarrow\; \varphi[x, f_y(x), z]$$

Now replace every occurrence of $f_y(x) \simeq d$ in φ by a predicate $p_y(x,d)$. Finally, add axioms for the properties of $\mathbf{D}$ *and* p_y*:*

$$\begin{array}{l} \forall x \,.\, \mathbf{D}(x) \leftrightarrow \bigvee_{d\in\mathcal{D}} x \simeq d \\ \forall z,u,v \,.\, p_y(z,u) \wedge p_y(z,v) \;\rightarrow u \simeq v \\ \forall z \bigvee_{d\in\mathcal{D}} p_y(z,d) \end{array}$$

The embedding of QBF into EPR is obtained from this construction by specializing the domain $\mathcal{D}$ to the set $\{\mathtt{tt},\mathtt{ff}\}$. A more general logic of finite domains includes predicates besides equality. An embedding of this logic into EPR is provided in Chapter 4 of [10]. The more general finite domain logic with predicates is easily seen equivalent to EPR, because EPR formulas are equi-satisfiable to a finite domain restriction, but the finite domain variant with just equalities is not. It corresponds to PSPACE, as opposed to NEXPTIME.

So, we embedded a set of problems that belong in PSPACE to EPR that is a potentially exponentially more succinct formalism. Our decision procedure for EPR also takes up to exponential space. One could wonder if we could simulate a PSPACE procedure for QBF within EPR. To this end, let us start examining a canonical way of evaluating QBF formulas. It proceeds by evaluating a QBF formula by recursive descent, while building up a context ρ. Then the closed formula φ is equivalent to `true` *if and only if* $[\![\varphi]\!]_{[]} = \mathtt{true}$.

$$\begin{array}{rcl} [\![\forall x \,.\, \varphi]\!]_\rho &=& [\![\varphi]\!]_{\rho[x\mapsto\mathtt{true}]} \;\mathbf{and}\; [\![\varphi]\!]_{\rho[x\mapsto\mathtt{false}]} \\ [\![\exists x \,.\, \varphi]\!]_\rho &=& [\![\varphi]\!]_{\rho[x\mapsto\mathtt{true}]} \;\mathbf{or}\; [\![\varphi]\!]_{\rho[x\mapsto\mathtt{false}]} \\ [\![\varphi \vee \psi]\!]_\rho &=& [\![\varphi]\!]_\rho \;\mathbf{or}\; [\![\psi]\!]_\rho \\ [\![\varphi \wedge \psi]\!]_\rho &=& [\![\varphi]\!]_\rho \;\mathbf{and}\; [\![\psi]\!]_\rho \\ [\![\neg\varphi]\!]_\rho &=& \mathbf{not}\; [\![\varphi]\!]_\rho \\ [\![x]\!]_\rho &=& \rho(x) \end{array}$$

Consider a formula $\forall x_1, x_2 \exists y Q\overline{z} \,.\, \varphi[y]$ in prenex form. Skolemizing y produces $\forall x_1, x_2 Q\overline{z} \,.\, \varphi[R_y(x_1,x_2)]$. Thus, for every evaluation of x_1 and x_2: $(\mathtt{true},\mathtt{true})$, $(\mathtt{true},\mathtt{false})$, $(\mathtt{false},\mathtt{true})$, $(\mathtt{false},\mathtt{false})$, *we have to determine whether the corresponding tuple belongs to R_y. Note that in order to evaluate the formula it suffices to perform this guess* only once *per combination of x_1, x_2. Let us use this observation to sketch a DPLL-based strategy for EPR that works in polynomial space when the EPR formula is obtained from a QBF or finite domain formula. Thus, assume F is a set of clauses where every occurrence of predicate R is applied to the same set of variables. This holds for QBF and finite domain formulas obtained by skolemization. Then for each predicate R of arity m enumerate the arguments to R using a lexicographic ordering: $\overline{a}_1, \overline{a}_2, \ldots, \overline{a}_{|\mathbf{D}|^m}$, where each a_i has arity m. Following the enumeration, start with the partial model $\Gamma : R(\overline{a}_1)$. Assume without loss of generality that the arguments to all other relations are the same as the ones passed to R. If the other relations are S and T, we would create the model $\Gamma : R(\overline{a}_1), S(\overline{a}_1), T(\overline{a}_1)$. The model suffices to evaluate all clauses of $F[\overline{a}_1]$. If some clause is not satisfied, then the assignment forces at least one of the relations to not contain $\overline{a}_1$, and standard DPLL backtracking can be used to either adjust the assignment to the relations, or detect that F is unsatisfiable. If all clauses are satisfied, then the inferred assignment does not contradict F. Now proceed by guessing an assignment to $\overline{a}_2$. The inference steps are independent of the steps used for $\overline{a}_1$ because every occurrence of every relation uses the same arguments. In this way, we can gradually build up an interpretation for all relations by examining elements $\overline{a}_1$, then $\overline{a}_2$, etc. Now observe that the assignments* *for $\overline{a}_1$ were irrelevant when examining $\overline{a}_2$. We therefore do not need to maintain the partial model for assignments that have been previously examined. We could call this process* guess, assign, and forget. *We saw that the process is sound and complete when relations are applied to the same arguments and when enumerating assignments in a prescribed order.*

Example 6 (Finite model finding)**.** *The DarwinFM [1] model finder reduces model finding problems to EPR. We here review an approach to EPR-based finite model finding and combine it with a refutationally complete extension similar to a GEO-style model-expansion inferences [7]. Other finite model finders are SEM [17], MACE [12] and Paradox [4]. These model finders are based on either pure SAT solvers or on a combination of SAT solver with solvers for uninterpreted function symbols (which is the case with SEM). To encode finite domain functions, the solvers tend to require a super-linear number of propositional clauses compared to the size of the input.*

Like DarwinFM, we can take advantage of the expressiveness of EPR to encode functions as relations, and avoid splitting finite relations into propositional variables.

To simplify the presentation below, assume we have a single binary function f. We introduce the 3-ary relation $\mathcal{R}_f$ to encode the function graph of f. The function f can be eliminated from the input clauses in a standard way, while preserving satisfiability. That is:

1. *Eliminate f from every clause C:*

$$\forall x, y, z. C[f(x, y)] \mapsto \forall x, y, z, u. \neg \mathcal{R}_f(x, y, u) \vee C[u] \tag{14}$$

2. *Add axioms:*

$$\forall \vec{x}, y, y' \, . \, \mathcal{R}_f(\vec{x}, y) \wedge \mathcal{R}_f(\vec{x}, y') \;\rightarrow\; y \simeq y' \tag{15}$$

$$\forall \vec{x} \, . \, \bigvee_{c \in \Sigma} \mathcal{R}_f(\vec{x}, c) \;\vee\; \mathsf{Ans}_f(\vec{x}) \tag{16}$$

The auxiliary predicate $\mathsf{Ans}_f(\vec{x})$ tracks dependencies on the constraints on the range of f.

3. *For each function f assert*

$$\forall \vec{x} \, . \, \bigvee_{j=1}^{0} \vec{a}_j \simeq \vec{x} \vee \; \neg \mathsf{Ans}_f(\vec{x}). \tag{17}$$

The resulting set of clauses are in EPR, and if the clauses are satisfiable, there is a finite model for the original formula. On the other hand, if the clauses are unsatisfiable, then there is a resolution proof of unsatisfiability. If the proof does not use the axiom (17), then the original clauses are unsatisfiable. Assume that proof-search is constrained to produce proofs that do not use (17) whenever possible. Thus, if the clauses are unsatisfiable without using the range constraints on f we will discover this before depending on the current range for f.

4. *Let us assume that the EPR clauses are unsatisfiable, and all resolution proofs of unsatisfiability require (17). In this case there is also a finite ground instance that is unsatisfiable. Suppose the ground instance uses the units $\neg \mathsf{Ans}_f(\vec{a}_1), \ldots, \neg \mathsf{Ans}_f(\vec{a}_n)$. We now repeat adding fresh constants corresponding to terms and modify the clause (17). Assume (17) is of the form:*

$$\forall \vec{x} \, . \, \bigvee_{j=1}^{n} \vec{a}_j \simeq \vec{x} \;\vee\; \neg \mathsf{Ans}_f(\vec{x}).$$

for $n \geq 0$, and that the clauses are unsatisfiable using the ground instances: $\neg\mathsf{Ans}_f(\vec{a}_{n+1}), \ldots, \neg\mathsf{Ans}_f(\vec{a}_{n+m})$. Replace it by the clause:

$$\forall \vec{x} \, . \, \bigvee_{j=1}^{m+n} \vec{a}_j \simeq \vec{x} \; \vee \; \neg\mathsf{Ans}_f(\vec{x}). \tag{18}$$

Furthermore introduce fresh constants of the form $c_{f(\vec{a}_j)}$ to provide range elements for the function f and add the facts:

$$\bigwedge_{j=n}^{m} \mathcal{R}_f(\vec{a}_j, c_{f(\vec{a}_j)}) \tag{19}$$

Theorem 1. *The model search algorithm is both refutationally complete as well as complete for finding finite models.*

Proof. If there is a finite model of size N of the original set of clauses, then every term of depth more than N contains itself as a sub-term. We therefore only have to create constants that correspond to terms of depth at most N in order to admit a finite model of size N.

Refutational completeness follows by compactness of first-order logic: If a set of clauses is unsatisfiable it is unsatisfiable for a finite amplification. We will need a fair way to enumerate the Herbrand Universe. One way of achieving this is to give precedence to proofs that introduce terms of the smallest possible depth. □

This presentation of the finite model-building routine suffers from a few practical problems. We presented the process for modifying the clause (17) by replacement. A better approach, in the context of DPLL, is to assert the original version of (17) as a unit literal, and ensure that the literal does not get simplified away from conflict clauses (it is common to simplify away unit literals from conflict clauses because their assignment never changes). In subsequent rounds, add a fresh answer literal corresponding to the number of rounds, and add a clause that interprets the previous round's answer literal by the newly introduced constants and the new answer literal. Another potential drawback is that the scheme searches the space of terms, and not number of elements. Thus, there may be exponentially many terms of depth N, but a finite model finder based on domain cardinalities does not require searching the set of depth N terms, whereas this could.

3 Preliminaries

3.1 Basic conventions

We use lower case letters from the beginning of the alphabet, $a, b, c, \ldots$, to range over a finite alphabet Σ of constants, while $\vec{a}, \vec{b}, \vec{c}$ are tuples of constants. We use letters $x, y, z, x_0, x_1, x_2, \ldots$ for variables from a set $\mathcal{V}$; and tuples are variables are written $\vec{x}, \vec{y}, \vec{z}$. The range of letters $p_1, p_2, p, q, r, P, Q, R, S, T, \ldots$ are used for atomic predicates of varying arities. Signed predicate symbols are identified by the set $\mathcal{L}$. As usual, literals (identified by the letter ℓ) are either atomic predicates or their negations applied to arguments. For example, $\neg p(x_1, x_2)$ (or $\overline{p}(x_1, x_2)$) is the literal where the binary atomic predicate p is negated.

Clauses consist of a finite set of literals, where each atomic predicate is applied to a tuple of variables and constants. For example $p(x_1, a) \vee \overline{q}(x_3) \vee \overline{q}(b)$ is a (well formed) clause. We use C, C', C_1, C_2 to range over clauses. When we later combine clauses with *substitution sets* (Section 3.4), we will use *normalized* clauses. These are clauses where predicates are applied to distinct variables. The empty clause is identified by a □.

3.2 Equality

EPR remains decidable if we add the theory of equality. The theory of equality can be directly encoded in EPR by supplying the usual equality and congruence axioms. Thus, to axiomatize that equality is an equivalence relation, add the axioms:

$$\begin{array}{rl} \mathsf{Reflexivity} & \forall x_1 \,.\, x_1 \simeq x_1 \\ \mathsf{Symmetry} & \forall x_1, x_2 \,.\, (x_1 \not\simeq x_2 \vee x_2 \simeq x_1) \\ \mathsf{Transitivity} & \forall x_1, x_2, x_3 \,.\, (x_1 \not\simeq x_2 \vee x_2 \not\simeq x_3 \vee x_1 \simeq x_3) \end{array}$$

Furthermore, for each additional predicate, such as P (of arity n) and Q (of arity m), add the axioms:

$$\begin{array}{ll} \mathsf{Cong}(p) & \forall x_1, .., x_n, y_1, .., y_n \,.\, (\neg p(x_1, .., x_n) \vee \bigvee_{i=1}^{n} x_i \not\simeq y_i \vee p(y_1, .., y_n)) \\ \mathsf{Cong}(q) & \forall x_1, .., x_m, y_1, .., y_m \,.\, (\neg q(x_1, .., x_m) \vee \bigvee_{i=1}^{m} x_i \not\simeq y_i \vee q(y_1, .., y_m)) \end{array}$$

Clearly, if a formula $\exists\vec{x}\forall\vec{y}\varphi(\vec{x},\vec{y})$ has a model where equality is interpreted literally as equality, then the same model satisfies the additional equality axioms. Conversely, if there is a model of $\exists\vec{x}\forall\vec{y}\varphi(\vec{x},\vec{y})$ and the additional equality axioms, then we can take a quotient of such a model under $\simeq$. The quotient still satisfies the formula, and furthermore the quotient ensures that no two different elements are congruent modulo equality.

3.3 DPLL as an abstract transition system

Key ingredients to recent efficient decision procedures for Boolean Satisfiability have been a combination of non-chronological back-jumping, lemma learning, and efficient Boolean propagation using literal watch heuristics. We will here recall back-jumping and lemma learning using a presentation of DPLL as an abstract transition system. As we later develop DPLL($\mathcal{SX}$) and DPLL(SE), the inference rules used for the purely propositional case will be generalized for EPR and then for EPR with equality.

During search, the states of the abstract transition system are of the form

$$\Gamma \,\|\, F$$

where Γ is a partial model and F is a set of clauses. The partial model consists of a sequence of literals that are either proceeded by a *decision* marker $\diamond$ to indicate that their value was assigned as a consequence of a guess; or the literals are annotated by a clause (so they are of the form $\ell^{C\vee\ell}$) to indicate that their value was assigned as a consequence of a unit propagation. The clause annotation provides an *explanation* for the propagated literal assignment. It will be convenient to use Γ directly as a partial assignment, that is a map from literals to Boolean values, as follows:

$$\Gamma(\ell) \quad = \quad \begin{cases} \texttt{true} & \text{if } \ell \in \Gamma \\ \texttt{false} & \text{if } \overline{\ell} \in \Gamma \\ \texttt{undef} & \text{otherwise; and we say “}\ell\text{ is unassigned in }\Gamma\text{”} \end{cases} \tag{20}$$

We will use the definition to capture the case where all literals in a clause are forced false by a context

$$\Gamma \Vdash \neg C \qquad \equiv \qquad \Gamma(\ell) = \texttt{false} \text{ for every } \ell \text{ in } C \tag{21}$$

$$\text{Decide}\ \frac{\ell \text{ or } \overline{\ell} \text{ occurs in } F \quad \Gamma(\ell) = \texttt{undef}}{\Gamma \parallel F \implies \Gamma \diamond \ell \parallel F}$$

$$\text{UnitPropagate}\ \frac{\Gamma(\ell) = \texttt{undef} \quad \Gamma \Vdash \neg C}{\Gamma \parallel F, C \vee \ell \implies \Gamma \ell^{C \vee \ell} \parallel F, C \vee \ell}$$

$$\text{Conflict}\ \frac{\Gamma \Vdash \neg C}{\Gamma \parallel F, C \implies \Gamma \parallel F, C \parallel C}$$

$$\text{Factoring}\quad \Gamma \parallel F \parallel C \vee \ell \vee \ell \implies \Gamma \parallel F \parallel C \vee \ell$$

$$\text{Resolve}\ \frac{\ell^{C \vee \ell} \in \Gamma}{\Gamma \parallel F \parallel C' \vee \overline{\ell} \implies \Gamma \parallel F \parallel C \vee C'}$$

$$\text{Backjump}\ \frac{\Gamma \Vdash \neg C}{\Gamma \diamond \ell' \Gamma' \parallel F \parallel C \vee \ell \implies \Gamma \ell^{C \vee \ell} \parallel F}$$

$$\text{Unsat}\quad M \parallel F \parallel \Box \implies \texttt{unsat}$$

$$\text{Learn}\ \frac{C \notin F}{\Gamma \parallel F \parallel C \implies \Gamma \parallel F, C \parallel C}$$

Figure 1: Core DPLL calculus for Boolean satisfiability

During conflict resolution, the abstract transition system maintains states of the form

$$\Gamma \parallel F \parallel C$$

where C is a *conflict* clause and, as before, Γ is a partial model and F is the current set of clauses.

A search process in the abstract presentation of DPLL starts with a state $\epsilon \parallel F$ comprising of an empty partial model ϵ and a set of clauses F. The possible steps from a state $\Gamma \parallel F$ are to (1) guess a literal assignment (using the Decide rule) to a literal that has not already been assigned, (2) (using UnitPropagate) propagate a literal assignment by assigning a truth value to a literal ℓ when it appears in a clause $C \vee \ell$ where all literals in C have been assigned to false, (3) to detect that the current literal assignment contradicts a clause (Conflict), or (4) to conclude with a complete assignment Γ that satisfies all clauses. In the last case, the search is done and a satisfying assignment has been extracted. In case (3), the current assignment needs to be undone by jumping back to a previous partial assignment and flip a previous guess. This process is guided by conflict resolution steps that use and modify a conflict clause in order to track the dependencies of decisions that caused the conflict. The conflict resolution rules use a rule Resolve in order to refine the conflict clause by dependencies, Factoring to remove duplicate literals, and Backjump to flip the last literal assignment that caused the conflict.

The premises for the Decide and UnitPropagate rules ensure that the context is always satisfiable. That is, we have the invariant:

Invariant 1. *For every generated context of the form $\Gamma \parallel F$ and $\Gamma \parallel F \parallel C$ and every atomic predicate p it is the case that either $p \notin \Gamma$ or $\overline{p} \notin \Gamma$. Thus, equation (20) is well defined.*

One thing to notice about conflict resolution is that Backjump is always enabled if the context contains at least one decision literal. This claim follows as the literals in the conflict clause are inserted

$$\textsf{Resolve}\ \Gamma\ell^{C\vee\ell} \parallel F \parallel C' \vee \overline{\ell} \implies \Gamma \parallel F \parallel C \vee C'$$

$$\textsf{Skip}\ \frac{\overline{\ell} \notin C}{\Gamma\ell^{C'} \parallel F \parallel C \implies \Gamma \parallel F \parallel C}$$

Figure 2: First unique implication point resolution

according to the ordering in Γ. More precisely, the UnitPropagate and Backjump rules directly establish the following invariant:

Invariant 2. *For every generated context of the form* $\Gamma\ell^{C\vee\ell}\Gamma' \parallel F$ *it is the case that* $\Gamma \Vdash \neg C$.

3.3.1 Refining resolution

In Figure 1 we formulated the resolution step to apply to an arbitrary non-decision literal in the context Γ. A refinement of this general conflict resolution rule is the *first unique implication point* (FUIP) conflict resolution strategy [18]. There, the first-unique implication point heuristic for conflict resolution was shown experimentally to offer advantages over an array of other proposals. We can capture the strategy by restricting conflict resolution to always pop the top-most literal from the partial model Γ until the context below the last decision literal implies a conflict. Figure 2 contains this rule. We should of course always apply Factoring as much as possible prior to Resolve to avoid getting stuck.

3.4 Relational Algebra

The extension of DPLL to EPR uses notations known from relational algebra heavily, and we will recall those here. These will be useful in manipulating substitution sets that are used in DPLL($\mathcal{SX}$). See also [15], [14] and [16].

For a fixed set of variables $\mathcal{V}$ and a fixed set of constants Σ, a substitution θ is an idempotent partial function from $\mathcal{V}$ to $\mathcal{V} \cup \Sigma$. A domain of a substitution can also be the empty set. If a substitution maps variables only to constants then it is called *instantiation*. With each substitution we associate a set of instances, denoted with $instancesOf(\theta)$. As an illustration, for the substitution $\theta = [x \mapsto y, y \mapsto y, z \mapsto a]$ and set of constants $\Sigma = \{a, b, c\}$, set of instances is $instancesOf(\theta) = \{[x \mapsto a, y \mapsto a, z \mapsto a], [x \mapsto b, y \mapsto b, z \mapsto a], [x \mapsto c, y \mapsto c, z \mapsto a]\}$. Formally, it is defined as $instancesOf(\theta) = \{\theta' \in (\mathbf{Dom}(\theta) \to \Sigma) \mid \forall x \in \mathbf{Dom}(\theta).\ \theta'(x) = \theta'(\theta(x))\}$.

We define a set of substitutions as a set of instances, but we will use the terminology substitution set to express that we will represent those set as a composition of instantiations and substitutions.

We denote sets of substitutions with Θ. Those sets are used in the notion of substitution-set constrained clauses. A substitution-set constrained clause is a pair $C \cdot \Theta$, where C is a clause and Θ is a substitution set. $C \cdot \Theta = \{\theta(C) \mid \theta \in \Theta\}$ and $\theta(C)$ is defined in a standard way: $\theta(\ell_1 \vee \ldots \vee \ell_m) = \theta(\ell_1) \vee \ldots \vee \theta(\ell_m)$ and $\theta(f(t_1, \ldots, t_m)) = f(\theta(t_1), \ldots, \theta(t_m))$.

Clauses can be represented more succinctly when using substitutions sets, for example set of clauses $p(a, b), p(b, c), p(c, d)$ can be represented as $p(x, y) \cdot \{[x \mapsto a, y \mapsto b], [x \mapsto b, y \mapsto c], [x \mapsto c, y \mapsto d]\}$, or simply: $p(x, y) \cdot \{(a, b), (b, c), (c, d)\}$.

The operations we consider on substitution sets are:

Selection $\sigma_{\varphi(\vec{x})}\Theta$ is shorthand for $\{\theta \in \Theta \mid \varphi(\theta(\vec{x}))\}$.

Projection $\pi_{\vec{x}}\Theta$ is shorthand for the set of substitutions obtained from Θ by removing domain elements other than $\vec{x}$. For example, $\pi_x\{[x \mapsto a, y \mapsto b], [x \mapsto a, y \mapsto c]\} = \{[x \mapsto a]\}$.

Co Projection $\hat{\pi}_{\vec{x}}\Theta$ is shorthand for the set of substitutions obtained from Θ by removing $\vec{x}$. So $\hat{\pi}_x\{[x \mapsto a, y \mapsto b], [x \mapsto a, y \mapsto c]\} = \{[y \mapsto b], [y \mapsto c]\}$.

Join $\Theta \bowtie \Theta'$ is the natural join of two relations. If Θ uses the variables $\vec{x}$ and $\vec{y}$, and Θ' uses variables $\vec{x}$ and $\vec{z}$, where $\vec{y}$ and $\vec{z}$ are disjoint, then $\Theta \bowtie \Theta'$ uses $\vec{x}, \vec{y}$ and $\vec{z}$ and is equal to $\{\theta \mid \hat{\pi}_{\vec{z}}(\theta) \in \Theta, \hat{\pi}_{\vec{y}}(\theta) \in \Theta'\}$. For example, $\{[x \mapsto a, y \mapsto b], [x \mapsto a, y \mapsto c]\} \bowtie \{[y \mapsto b, z \mapsto b], [y \mapsto b, z \mapsto a]\} = \{[x \mapsto a, y \mapsto b, z \mapsto b], [x \mapsto a, y \mapsto b, z \mapsto a]\}$.

Renaming $\delta_{\vec{x}\to\vec{y}}\Theta$ is the relation obtained from Θ by renaming the variables $\vec{x}$ to $\vec{y}$. We here assume that $\vec{y}$ is not used in Θ already.

Restriction $\Theta \wr \theta$ restricts the set Θ to the substitution θ. It is shorthand for a sequence of selection and co-projections. For example, $\Theta \wr [x \mapsto a] = \hat{\pi}_x \sigma_{x=a}\Theta$, and $\Theta \wr [x \mapsto y, y \mapsto y] = \hat{\pi}_x\sigma_{x=y}\Theta$. More generally, $\Theta \wr \theta$ is $\hat{\pi}_{\vec{x}}\sigma_{\vec{x}=\theta(\vec{x})}\Theta$ where $\vec{x}$ is the subset of the domain of θ where θ is not idempotent. Thus, if $\vec{x} = \{x_1, \ldots, x_n\}$, then $\theta(x_1) \neq x_1, \ldots, \theta(x_n) \neq x_n$.

Set operations $\Theta \cup \Theta'$ creates the union of Θ and Θ', both sets of n-ary tuples (sets of instances with n variables in the domain). Subtraction $\Theta \setminus \Theta'$ is the set $\{\theta \in \Theta \mid \theta \notin \Theta'\}$. The complement $\overline{\Theta}$ is $\Sigma^n \setminus \Theta$.

The empty set $\emptyset$ should not be confused with the singleton set containing the substitution $\{[]\}$ with an empty domain. For example, $\emptyset \bowtie \Theta = \emptyset$, but $\{[]\} \bowtie \Theta = \Theta$.

3.5 Closing relations under equality

We will also recall how binary relations can be closed under reflexivity, symmetry and transitivity. In the following, let R be a binary relation, then we define auxiliary operations on R:

$$\begin{aligned} R^{-1} &= \{(y,x) \mid (x,y) \in R\} && (22)\\ R^0 &= \{(x,y) \mid x = y\} && (23)\\ R^{n+1} &= R^0 \cup \hat{\pi}_z(\delta_{y\to z}(R^n) \bowtie \delta_{x\to z}(R^n)) && (24)\\ R^* &= R^0 \cup R \cup R^2 \cup \ldots && (25)\\ [R] &= (R \cup R^{-1})^* && (26) \end{aligned}$$

We call $[R]$ the *equivalence* closure of R. The equivalence closure of any relation contains the identity relation. In the following, we will use E for a binary relation that is closed under equivalence, that is, we will assume that:

$$[E] = E.$$

The construction for the equivalence closure outlined above and in particular in (24) uses iterative squaring. The number of iterations required for computing the equivalence closure is therefore logarithmic in the diameter of the graph induced by R.

For an n-ary relation Θ and equivalence relation E define the closure of Θ under E as:

$$[\Theta]_E = \{\vec{x} \mid \exists \vec{y} \in \Theta \bigwedge_{i=1}^{n} . (x_i, y_i) \in E\} \qquad (27)$$

Suppose Θ is an n-ary relation. We can compute the closure using a convolution Θ_n, where:

$$\begin{aligned}\Theta_0 &= \Theta \\ \Theta_{k+1} &= \hat{\pi}_y(E \bowtie \Theta_k),\ 0 \le k < n\end{aligned}$$

We can also compute the n-ary equality closure E_n by using the auxiliary definition:

$$\begin{aligned}E_0 &= E^0 \\ E_{k+1} &= E_k \bowtie \delta_{x,y \to x_k,y_k} E\end{aligned}$$

Lemma 1. *The convolution computes the closure of Θ under E:*

$$[\Theta]_E = \Theta_n = \delta_{\overline{y}_k \to \overline{x}_k}(\pi_{\overline{y}_k}(\Theta \bowtie E_n)).$$

Lemma 2. *Suppose E is closed under equivalence. That is $[E] = E$. Let R be a binary relation, then*

$$[E \cup R] \quad = \quad E \cup [[R]_E] \tag{28}$$

Proof. The inclusion $\supseteq$ is immediate. To establish $\subseteq$ consider a pair (a, b) that is an element of $[E \cup R]$. Thus, there is a path $(c_1, \ldots, c_n)$, where $a = c_1$ and $b = c_n$, such that each pair on the path is in either E or in R or R^{-1}. If all elements on the path are in E we are done because then $(a, b) \in E$. So assume the path contains pairs of elements in R. Each subsequence that contains at most one pair in R (the rest in E) will be in $[R]_E$. These subsequences are combined by taking the reflexive, symmetric, transitive closure, which is $[[R]_E]$. ☐

3.6 A DPLL($\mathcal{SX}$) calculus for pure EPR

We here extend the basic DPLL calculus presentation to EPR. A detailed treatment of this extension is provided in [6], but we repeat the main calculus with *simultaneous* unit propagation rules as it prepares the ground for adding built-in inference support for equality to EPR.

In DPLL($\mathcal{SX}$), a clause is represented as a pair $C \cdot \Theta$, where C as before is a list of literals, comprising of n-ary predicate symbols, and Θ is a set of instances for the predicates in C. The set Θ may be represented succinctly using a combination of Binary Decision Diagrams and substitutions (as described in [6]); so we call Θ a *substitution set*. The definition of a context Γ is also lifted to substitution sets. Literals in Γ are associated with set of instances, so now a context is of the form $\ell_1\Theta_1 \ldots \ell_k\Theta_k$.

The rules use a generalization of definition (20) to substitution sets; namely:

$$\Gamma(\ell) \quad = \quad \bigcup \{\Theta \mid \ell \cdot \Theta \in \Gamma\} \tag{29}$$

Thus, the truth assignments for a literal ℓ consists of the union of instances for ℓ in Γ.

Figures 3 and 4 summarize DPLL($\mathcal{SX}$). For example, the Decide rule has been generalized to use substitution sets for identifying new case splits. Instead of requiring $\Gamma(\ell) = \texttt{undef}$ it requires a more general side condition $\Gamma(\ell) \bowtie \Theta = \Gamma(\overline{\ell}) \bowtie \Theta = \emptyset$. Likewise, unit propagation is generalized as it assigns new instances to a literal based on joining the instances assigned to the complement of other literals in a clause. A clause is conflicting if the instances assigned to the complement of all literals in it forms a non-empty intersection.

One property maintained by the resulting system is that the context Γ is always consistent. That is, similar to invariant 1 we have

Invariant 3. *For every generated context of the form $\Gamma \parallel F$ it is the case that $\Gamma(\ell) \cap \Gamma(\overline{\ell}) = \emptyset$.*

So whenever UnitPropagate, Decide and Conflict are disabled, then the invariant allows us to conclude that the set of clauses are satisfiable with model Γ.

Invariant 2 admits a similar lifting:

Invariant 4. *For every derived context of the form* $\Gamma\ell^{C\cdot\Theta}\Theta'\Gamma'$ *it is the case that* $C = (\ell_1 \vee \ldots \vee \ell_k \vee \ell(\vec{x}))$ *and* $\emptyset \neq \Theta' \subseteq \pi_{\vec{x}}\,(\Theta \bowtie \Gamma(\overline{\ell}_1) \bowtie \ldots \bowtie \Gamma(\overline{\ell}_k))$.

Invariant 4 allows us to introduce the notion of the set of premises for a literal. We write *premises* $(\ell^{C\cdot\Theta}\Theta')$ to extract particular literal positions $\overline{\ell}_1\Theta_1, \ldots, \overline{\ell}_k\Theta_k$ in Γ such that $\Theta' \bowtie \Theta_1 \bowtie \ldots \bowtie \Theta_k \neq \emptyset$ (where the variables in $\Theta_1, \ldots, \Theta_k$ have been renamed appropriately to align with the names used in Θ').

$$\textsf{Decide}\ \frac{\ell \in F \quad \Gamma(\ell) \bowtie \Theta = \Gamma(\overline{\ell}) \bowtie \Theta = \emptyset}{\Gamma \parallel F \implies \Gamma \diamond \ell\Theta \parallel F}$$

$$\textsf{UnitPropagate}\ \frac{\begin{array}{l} C = (\ell_1 \vee \ldots \vee \ell_k \vee \ell(\vec{x})), \\ \Theta' = \pi_{\vec{x}}(\Theta \bowtie \Gamma(\overline{\ell}_1) \bowtie \ldots \bowtie \Gamma(\overline{\ell}_k)) \setminus \Gamma(\ell) \neq \emptyset \\ \Theta' \bowtie \Gamma(\overline{\ell}) = \emptyset \end{array}}{\Gamma \parallel F, C \cdot \Theta \implies \Gamma\ell^{C\cdot\Theta} \cdot \Theta' \parallel F, C \cdot \Theta}$$

$$\textsf{Conflict}\ \frac{C = (\ell_1 \vee \ldots \vee \ell_k),\ \Theta_r = \Theta \bowtie \Gamma(\overline{\ell}_1) \bowtie \ldots \bowtie \Gamma(\overline{\ell}_k) \neq \emptyset}{\Gamma \parallel F, C \cdot \Theta \implies \Gamma \parallel F, C \cdot \Theta \parallel C \cdot \Theta, \Theta_r}$$

Figure 3: Search inference rules

$$\textsf{Resolve}\ \frac{\begin{array}{l} \delta_{\vec{y}\to\vec{x}}\pi_{\vec{y}}\Theta_r \bowtie \Theta_\ell = \emptyset \text{ for every } \ell(\vec{y}) \in C',\ C_\ell = (C(\vec{y}) \vee \overline{\ell}(\vec{x})) \\ \Theta'_r = \hat{\pi}_{\vec{x}}(\Theta_r \bowtie \Theta_\ell \bowtie \textit{premises}(\overline{\ell}\Theta_\ell)) \neq \emptyset,\ \ \Theta'' = \hat{\pi}_{\vec{x}}(\Theta \bowtie \Theta') \end{array}}{\Gamma\overline{\ell}^{C_\ell\cdot\Theta'}\Theta_\ell \parallel F \parallel (C'(\vec{z}) \vee \ell(\vec{x})) \cdot \Theta, \Theta_r \implies \Gamma \parallel F \parallel (C(\vec{y}) \vee C'(\vec{z})) \cdot \Theta'', \Theta'_r}$$

$$\textsf{Skip}\ \frac{\delta_{\vec{y}\to\vec{x}}\pi_{\vec{y}}\Theta_r \bowtie \Theta_\ell = \emptyset \text{ for every } \ell(\vec{y}) \in C}{\Gamma\overline{\ell}^{C_\ell\cdot\Theta'}\Theta_\ell \parallel F \parallel C \cdot \Theta, \Theta_r \implies \Gamma \parallel F \parallel C \cdot \Theta, \Theta_r}$$

$$\textsf{Factoring}\ \frac{\Theta'_r = \hat{\pi}_{\vec{z}}\sigma_{\vec{y}=\vec{z}}\Theta_r \neq \emptyset,\ \Theta' = \hat{\pi}_{\vec{z}}\sigma_{\vec{y}=\vec{z}}\Theta}{\Gamma \parallel F \parallel (C(\vec{x}) \vee \ell(\vec{y}) \vee \ell(\vec{z})) \cdot \Theta, \Theta_r \implies \Gamma \parallel F \parallel (C(\vec{x}) \vee \ell(\vec{y})) \cdot \Theta', \Theta'_r}$$

$$\textsf{Learn}\ \frac{C \cdot \Theta \notin F}{\Gamma \parallel F \parallel C \cdot \Theta, \Theta_r \implies \Gamma \parallel F, C \cdot \Theta \parallel C \cdot \Theta, \Theta_r}$$

$$\textsf{Unsat}\ \Gamma \parallel F \parallel \Box \cdot \Theta, \Theta_r \implies \textsf{unsat} \qquad \textbf{if } \Theta \neq \emptyset$$

$$\textsf{Backjump}\ \frac{\begin{array}{l} C = (\ell_1 \vee \ldots \vee \ell_k \vee \ell(\vec{x})), \\ \Theta' = \pi_{\vec{x}}(\Theta \bowtie \Gamma_1(\overline{\ell}_1) \bowtie \ldots \bowtie \Gamma_1(\overline{\ell}_k)) \setminus \Gamma_1(\ell) \neq \emptyset \end{array}}{\Gamma_1 \diamond \Gamma_2 \parallel F \parallel C \cdot \Theta, \Theta_r \implies \Gamma_1\ell^{C\cdot\Theta}\Theta' \parallel F}$$

$$\textsf{Refine}\ \ \Gamma \diamond \ell\Theta_1\Gamma' \parallel F \parallel C \cdot \Theta, \Theta_r \implies \Gamma \diamond \ell\Theta'_1 \parallel F \qquad \textbf{if } \emptyset \neq \Theta'_1 \subset \Theta_1$$

Figure 4: Conflict resolution rules

4 DPLL(SE)

The main objective in this paper is to give direct support for equality as an extension to DPLL($\mathcal{SX}$). This section presents the extension. While the rules in DPLL($\mathcal{SX}$) are based on extracting truth assignments using the auxiliary function $\Gamma(\ell)$, a key change here is to extend this facility to take the set of asserted equalities into account. The reason is that we wish to use all consequences of asserted equalities during search without creating an explicit trail of equality propagation. Define:

$$
\begin{aligned}
E(\Gamma) &= [\Gamma(\simeq)] && (30)\\
\Gamma^{\simeq}(\ell) &= [\Gamma(\ell)]_{E(\Gamma)} \;=\; \Big[\bigcup\{\Theta' \mid \ell \cdot \Theta' \in \Gamma\}\Big]_{E(\Gamma)} && (31)
\end{aligned}
$$

- $E(\Gamma)$ denotes the equivalence closure constructed using all equalities that occur in the context Γ
- $\Gamma^{\simeq}(\ell)$ describes the set of all instances of the literal ℓ that occur in the context Γ. Moreover, $\Gamma^{\simeq}(\ell)$ also contains all the instances of $\ell(\vec{x})$ that are inferred using the congruence closure $E(\Gamma)$. Note that $\Gamma^{\simeq}(\simeq) = E(\Gamma)$.

The next step is to lift the rules for DPLL($\mathcal{SX}$) to the equality case. We will lift invariants 3 and 4.

Invariant 5. *For every generated context of the form $\Gamma \parallel F$ it is the case that $\Gamma^{\simeq}(\ell) \cap \Gamma^{\simeq}(\overline{\ell}) = \emptyset$.*

Invariant 6. *For every derived context of the form $\Gamma\ell^{C\cdot\Theta}\Theta'\Gamma'$ it is the case that $C = (\ell_1 \vee \ldots \vee \ell_k \vee \ell(\vec{x}))$ and $\emptyset \neq \Theta' \subseteq \pi_{\vec{x}}(\Theta \bowtie \Gamma^{\simeq}(\overline{\ell}_1) \bowtie \ldots \bowtie \Gamma^{\simeq}(\overline{\ell}_k))$.*

But a direct lifting of the calculus that could be obtained by just replacing $\Gamma(\ell)$ by $\Gamma^{\simeq}(\ell)$ is not possible. We will expose one of the difficulties and hint at our approach before formulating DPLL(SE) in detail.

Example 7 (A difficulty with equality)**.** *Consider the search state:*

$$\diamond p(a),\ \diamond\overline{p}(b) \parallel \underbrace{p(b) \vee a \simeq b}_{C}$$

One application of unit propagation produces the context:

$$
\begin{aligned}
\Longrightarrow\ & \mathsf{UnitPropagate}\\
& \diamond p(a),\ \diamond\overline{p}(b),\ a \simeq b^{C} \parallel p(b) \vee a \simeq b
\end{aligned}
$$

This context is no longer consistent in the theory of equality. If we aim to develop a calculus and decision procedure where contexts are always satisfiable, such that invariant 3 holds (and inconsistencies are captured by conflict clauses), then cases like this one have to be handled by limiting unit propagation and instead identify uses for applying equality axioms. A potential problem in this example is that the clause C is not directly a conflict clause with the current context Γ.

Our presentation of the extensions to DPLL($\mathcal{SX}$) explain how to change the basic calculus in a way that avoids producing inconsistent contexts during search. This will be mostly done using the axioms for equalities (transitivity and congruence) and applying their instances. In the above state we can resolve C with the axiom $\overline{p}(a) \vee a \not\simeq b \vee p(b)$ which is an instance of the congruence axiom for p. The resulting clause becomes $\overline{p}(a) \vee p(b) \vee p(b)$ which after factoring simplifies to $\overline{p}(a) \vee p(b)$. This clause is conflicting in Γ and we proceed with conflict resolution.

The derivation in the new calculus that we will present is therefore:

$$\begin{array}{ll}
\Longrightarrow & \mathsf{E{-}CongConflict} \\
& \diamond p(a),\ \diamond\overline{p}(b) \parallel C \parallel \overline{p}(a) \vee p(b) \vee p(b) \\
\Longrightarrow & \mathsf{Factoring} \\
& \diamond p(a),\ \diamond\overline{p}(b) \parallel C \parallel \overline{p}(a) \vee p(b) \\
\Longrightarrow & \mathsf{Backjump} \\
& \diamond p(a),\ p(b)^{(\overline{p}(a) \vee p(b))} \parallel C
\end{array}$$

4.1 Decisions and Propagation

For literals different than equality $\simeq$ we can lift the Decide rule directly. We call it E-Decide to distinguish it, but the only difference is the use of $\Gamma^{\simeq}(\ell)$ instead of $\Gamma(\ell)$.

$$\textsf{E-Decide}\ \frac{\ell \in F,\ \ \ell \neq \simeq,\ \ \Theta \bowtie \Gamma^{\simeq}(\ell) = \emptyset,\ \ \Theta \bowtie \Gamma^{\simeq}(\overline{\ell}) = \emptyset}{\Gamma \parallel F \implies \Gamma \diamond \ell\Theta \parallel F}$$

For equality $\simeq$ we have the potential problem that asserting a new equality violates invariant 3. We filter potential violations in the pre-condition for decisions on equalities:

$$\textsf{E-Decide}_{\simeq}\ \frac{\Gamma_1 = \Gamma \diamond x \simeq y \cdot \Theta,\ \ \text{for every } \ell \in \Gamma_1\ \Gamma_1^{\simeq}(\ell) \bowtie \Gamma_1^{\simeq}(\overline{\ell}) = \emptyset}{\Gamma \parallel F \implies \Gamma_1 \parallel F}$$

Similar to applications of E-Decide we ensure that unit-propagation does not ignore conflicts that are implied by adding equalities to the context. We therefore check that the new context is consistent when adding the implied literal. This additional check is only necessary when the new propagated literal is $\simeq$, but we formulate the rule with this generality for an arbitrary propagated literal ℓ.

$$\textsf{E-UnitPropagate}\ \frac{\begin{array}{l} C = (\ell_1 \vee \ldots \vee \ell_k \vee \ell(\vec{x})), \\ \Theta' = \pi_{\vec{x}}(\Theta \bowtie \Gamma^{\simeq}(\overline{\ell}_1) \bowtie \ldots \bowtie \Gamma^{\simeq}(\overline{\ell}_k)) \setminus \Gamma^{\simeq}(\ell) \neq \emptyset \\ \Theta' \bowtie \Gamma^{\simeq}(\overline{\ell}) = \emptyset,\ \ \Gamma_1 = \Gamma\ell\Theta' \\ \Gamma_1^{\simeq}(\ell') \bowtie \Gamma_1^{\simeq}(\overline{\ell'}) = \emptyset,\ \text{for each } \ell' \text{ occurring in } \Gamma \end{array}}{\Gamma \parallel F, C \cdot \Theta \implies \Gamma\ell^{C \cdot \Theta} \cdot \Theta' \parallel F, C \cdot \Theta}$$

4.2 Conflicts

We may lift basic conflict detection to equalities directly; a conflict detected by taking the closure of instantiations under equality is still a conflict. So DPLL(SE) retains a variant of the rule Conflict:

$$\textsf{E-Conflict}\ \frac{C = (\ell_1 \vee \ldots \vee \ell_k),\ \ \emptyset \neq \Theta_r = \Theta \bowtie \Gamma^{\simeq}(\overline{\ell}_1) \bowtie \ldots \bowtie \Gamma^{\simeq}(\overline{\ell}_k)}{\Gamma \parallel F, C \cdot \Theta \implies \Gamma \parallel F, C \cdot \Theta \parallel C \cdot \Theta, \Theta_r}$$

However, the case covered by E-Conflict is not the only way that an assignment Γ can induce a contradiction. Example 7 gave an appetizer of the situation. It comprised of a context Γ containing $p(a)$ and $\overline{p}(b)$, while the set of clauses contained $p(b) \vee a \simeq b$. We solved it by replacing $a \simeq b$ in C by the literals $\overline{p}(a) \vee p(b)$.

Basically, we need to detect and resolve conflicts if an assignment Γ and clause $C \vee x \simeq y \cdot \Theta$ implies a set of equalities by unit propagation from C, but the newly propagated equalities contradict the assignment in Γ. The problem boils down to inferring congruence relations implied from (would-be) propagated equalities, and extracting a conflict state based on the (would-be) propagation. We summarize this situation in the rule E-CongConflict.

$$\textsf{E-CongConflict}\ \frac{\begin{array}{l} C = (\ell_1 \vee \ldots \vee \ell_k \vee x \simeq y), \\ \Theta_r = \Theta \bowtie \Gamma^{\simeq}(\overline{\ell}_1) \bowtie \ldots \bowtie \Gamma^{\simeq}(\overline{\ell}_k) \bowtie \overline{E(\Gamma)}, \\ \Theta' = \pi_{xy}\Theta_r \neq \emptyset, \\ \Gamma_1 = \Gamma x \simeq y\Theta' \\ \Gamma_1^{\simeq}(p) \bowtie \Gamma_1^{\simeq}(\overline{p}) \neq \emptyset,\ \text{for some } p \text{ occurring in } \Gamma \end{array}}{\Gamma \,\|\, F, C \cdot \Theta \implies \Gamma \,\|\, F, C \cdot \Theta \,\|\, C \cdot \Theta, \Theta_r}$$

The rule requires that Θ_r is the set of instances that imply $x \simeq y$. We obtain Θ_r by joining the instances for the complement of the literals in C. The derived substitution set Θ' is obtained from Θ_r by projecting x and y, and subtracting instances that are already equal. Observe that since Γ is consistent, and we subtracted the complement of existing derived equalities, we have that Θ' does not contain any instances for disequalities; so $\Theta_r \bowtie \Gamma^{\simeq}(\not\simeq) = \emptyset$. The rule then checks for the complement of the enabling condition for E-UnitPropagate. In this case there is a predicate p in Γ that receives contradictory assignments by the implied equality.

Prior to producing a conflict clause, our approach is to resolve this particular literal with the congruence axiom for the literal that would receive a contradictory assignment.

Example 8. *Consider the state:*

$$\overline{p}(a)\ p(f)\ x \simeq y\{(a,b),(c,d),(e,f)\}\ q(b,c)\ q(d,e) \,\|\, \overline{q}(x,y) \vee x \simeq y$$

The clause satisfies the premises of E-CongConflict*. A corresponding conflict clause is:*

$$\overline{q}(b,c)\ \vee\ \overline{q}(d,e)\ \vee\ p(b)\ \vee\ \overline{p}(e)$$

The function CongConflictInstance is used to produce the desired resolvent. We will motivate the function by first establishing a lemma that implies the existence of the function:

Lemma 3. *Following the pre-conditions of rule* E-CongConflict*: Given a clause $C = (\ell_1 \vee \ldots \vee \ell_k \vee x \simeq y)$ and a context Γ, let $\Theta' = \pi_{xy}(\Theta \bowtie \Gamma^{\simeq}(\overline{\ell}_1) \bowtie \ldots \bowtie \Gamma^{\simeq}(\overline{\ell}_k)) \setminus E(\Gamma)$. If $\Theta' \neq \emptyset$ then define the new context $\Gamma_1 = \Gamma x \simeq y\Theta'$. If there exists p occurring in Γ such that $\Gamma_1^{\simeq}(p) \bowtie \Gamma_1^{\simeq}(\overline{p}) \neq \emptyset$, then there exists a clause $C_1\Theta_1$ such that*

1. *$C_1\Theta_1$ is conflicting in Γ. In other words, the premises of rule* E-Conflict *apply.*
2. *$C_1\Theta_1$ can be derived from $C\Theta$ using congruence and resolution*

Proof. Let p be a predicate occurring in Γ such that $\Gamma_1^{\simeq}(\overline{p}) \bowtie \Gamma_1^{\simeq}(p) \neq \emptyset$. Assume that the premises of the lemma are satisfied. That implies that $\Gamma^{\simeq}(\overline{p}) \bowtie \Gamma^{\simeq}(p) = \emptyset$.

For simplicity we assume that the arity of p is 1. If p has a greater arity, then it is enough to apply the procedure that we will describe in this proof on an argument i of p such that $\Gamma_1^{\simeq}(\overline{p\mid_i}) \bowtie \Gamma_1^{\simeq}(p\mid_i) \neq \emptyset$. We know that at least one such an argument needs to exist.

The fact that adding equalities $x \simeq y\Theta'$ will raise the contradiction in the context Γ is expressed as follows:

$$\Gamma^{\simeq}(\overline{p}) \bowtie \delta_{x\to y}\Gamma^{\simeq}(p) \bowtie E(\Gamma) = \emptyset \tag{32}$$

but,

$$\Gamma^{\simeq}(\overline{p}) \bowtie \delta_{x\to y}\Gamma^{\simeq}(p) \bowtie E(\Gamma_1) \neq \emptyset \tag{33}$$

That implies that there exist a set of equalities $x \simeq y\Theta_N \in E(\Gamma_1)$ that binds $\Gamma^{\simeq}(\overline{p})$ and $\Gamma^{\simeq}(p)$. Before adding equalities $x \simeq y\Theta'$ to the context Γ those two sets were disjoint. Since $E(\Gamma_1) = [\Gamma_1(\simeq)] = [\Gamma(\simeq) \cup \Theta']$ a chain of equations connecting $\Gamma^{\simeq}(\overline{p})$ and $\Gamma^{\simeq}(p)$ will consist of a finite number of equalities belonging to $E(\Gamma)$ or Θ'. At least one equality from Θ' has to be present. Figure 5 visualizes the idea: the chain of equalities that connects $\Gamma^{\simeq}(\overline{p})$ and $\Gamma^{\simeq}(p)$ consists of alternating equalities from $[\Theta']$ and $E(\Gamma)$. The chain is defined as:

$$H_m \overset{def}{=} \delta_{y\to n_1}[\Theta'] \bowtie \delta_{xy\to n_1 o_1}E(\Gamma) \bowtie \ldots \bowtie \delta_{xy\to n_m o_m}E(\Gamma) \bowtie \delta_{x\to o_m}[\Theta'] \tag{34}$$

and an integer m is chosen in such a way that

$$B \overset{def}{=} \delta_{x\to y}\Gamma^{\simeq}(\overline{p}) \bowtie H_m \bowtie \Gamma^{\simeq}(p) \neq \emptyset \tag{35}$$

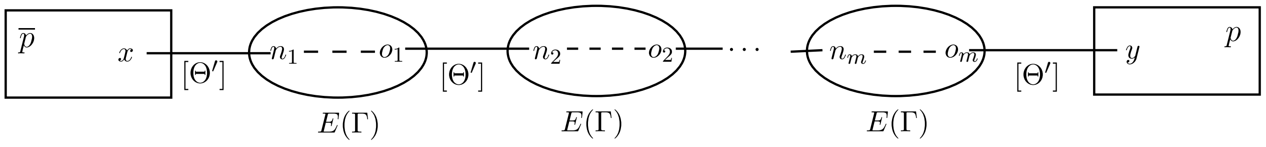

Figure 5: The chain of equalities binding $\Gamma^{\simeq}(\overline{p})$ and $\Gamma^{\simeq}(p)$ is constructed alternating equalities from Θ' and $E(\Gamma)$

Applying this on the Example 8 we obtain the following substitution set: $B = \{[x \mapsto b, n_1 \mapsto c, o_1 \mapsto d, y \mapsto e]\}$. Note that the B was constructed using only Θ' and $E(\Gamma)$. Now we explain how to derive the formula $C_1\Theta_1$ from $C\Theta$.

First we construct the formula $(x \not\simeq n_1 \vee n_1 \not\simeq o_1 \vee \ldots \vee o_m \not\simeq y \vee x \simeq y)H$, where the substitution set H contain a tuple of values that were used to establish the chain of equalities. This formula represents a transitive derivation of $x \simeq y\ \pi_{xy}H$. All equalities $n_i \simeq o_i\ \pi_{n_i o_i}H$ are already contained in $E(\Gamma)$. The remaining $m+1$ inequalities are resolved with the equality in the $C = (\ell_1 \vee \ldots \vee \ell_k \vee x \simeq y)$ clause.

The resulting clause has the form

$$(\ell_1^1 \vee \ldots \vee \ell_k^1 \vee \ldots \vee \ell_1^{m+1} \vee \ldots \vee \ell_k^{m+1} \vee x \simeq y)\Theta_R \tag{36}$$

where $\pi_{xy}\Theta_R = \pi_{xy}H$.

Next, we construct the axiom $(p(x) \vee x \not\simeq y \vee \overline{p}(y))\pi_{xy}H$. This is a congruence axiom for p. We resolve this axiom with (36) and denote the result by $C_1\Theta_1$. $C_1\Theta_1$ is conflicting in Γ and it was derived from $C\Theta$.

□

Definition 1 (CongConflictInstance)**.** *For a context* Γ *and a clause* $C\Theta$ *that fulfills the preconditions of Lemma 3, let* $C_1\Theta_1$ *be a clause derived from* $C\Theta$ *as described in the proof of the Lemma. We introduce the following shorthand to describe this fact:*

$$C_1\Theta_1 = \text{CongConflictInstance}(\Gamma, C, \Theta)$$

Definition 2 (Bridge)**.** *Given a context* Γ*, a new set of equalities* $x \simeq y\Theta'$ *added to* Γ *and an integer* m*, with* $\mathsf{Bridge}(E(\Gamma), \Theta', m)$ *we define a chain of equalities of size* $2m-1$ *defined in the same manner as in the proof of Lemma 3:*

$$\mathsf{Bridge}(E(\Gamma), \Theta', m) \overset{def}{=} \delta_{y\to n_1}[\Theta'] \bowtie \delta_{xy\to n_1 o_1}E(\Gamma) \bowtie \ldots \bowtie \delta_{xy\to n_m o_m}E(\Gamma) \bowtie \delta_{x\to o_m}[\Theta']$$

4.3 Conflict Resolution

Propagation of equalities are left implicit in the search inference rules. The price to pay is that dependencies on equalities in conflicts have to be re-produced during conflict resolution. So, a given conflict clause may depend on some subset of the asserted equalities in Γ. Conflict resolution would have to determine which. In this section we show how the set of equality dependencies can be unfolded lazily during conflict resolution.

We will approach conflict resolution as an instance of the FUIP strategy. This means that we will proceed from a conflict state $\Gamma \ell^{C' \vee \ell} \Theta' \parallel F \parallel C\Theta, \Theta_r$ and either pop $\ell\Theta'$ from the context by using Skip, apply Backjump, apply Factoring, or if none of these rules apply, resolve $C' \vee \ell$ with the conflict clause. The rules for resolution cannot be identical to the rule listed in Figure 4, first because equality is used implicitly during constraint propagation, second in light of E-CongConflict, we need to super-pose equalities into literals. Prior to formulating the conflict resolution rules, let us consider the possible cases where they should apply:

4.3.1 Resolve

The first case is when the premises of Resolve apply directly.

$$\frac{\begin{array}{c}\delta_{\vec{y}\to\vec{x}}\pi_{\vec{y}}\Theta_r \bowtie \Theta_\ell = \emptyset \text{ for every } \ell(\vec{y}) \in C',\ C_\ell = (C(\vec{y}) \vee \overline{\ell}(\vec{x})) \\ \Theta'_r = \hat{\pi}_{\vec{x}}(\Theta_r \bowtie \Theta_\ell \bowtie \textit{premises}(\overline{\ell}\Theta_\ell)) \neq \emptyset,\ \ \Theta'' = \hat{\pi}_{\vec{x}}(\Theta \bowtie \Theta')\end{array}}{\Gamma \overline{\ell}^{C_\ell \cdot \Theta'} \Theta_\ell \parallel F \parallel (C'(\vec{z}) \vee \ell(\vec{x})) \cdot \Theta, \Theta_r \implies \Gamma \parallel F \parallel (C(\vec{y}) \vee C'(\vec{z})) \cdot \Theta'', \Theta'_r}$$

In this case we can resolve ℓ with the matching negated literal in the context to obtain a conflict clause without the particular occurrence of ℓ.

4.3.2 E-Resolve

It can be the case that the last asserted literal in the context Γ_1 is an equality, $x \simeq y\Theta'$, which is used in conjunction with other equalities to establish the current conflict clause. The inter-dependencies of the last asserted equality and other equalities have to be untangled before we can apply E-Resolve.

$$\frac{\begin{array}{rcl}\Gamma_1 & = & \Gamma u \simeq v\Theta',\ \ \Gamma^{\simeq}(\overline{\ell}\ |_i) \bowtie \Theta_r = \emptyset,\ \ (\text{but } \Gamma_1^{\simeq}(\overline{\ell}\ |_i) \bowtie \Theta_r \neq \emptyset) \\ P & = & \delta_{xy\to x_i y_i}\mathsf{Bridge}(E(\Gamma), \Theta', m) \\ \Theta'' & = & \Gamma^{\simeq}(\overline{\ell}) \bowtie P \bowtie \delta_{\vec{x}\to\vec{y}}\Theta,\ \ \Theta''_r = \Theta'' \bowtie \delta_{\vec{x}\to\vec{y}}\Theta_r \neq \emptyset \\ C'' & = & C \vee \ell(\vec{y}) \vee x_i \not\simeq n_1 \vee n_1 \not\simeq o_1 \vee \ldots \vee n_m \not\simeq o_m \vee o_m \not\simeq y_i\end{array}}{\Gamma_1 \parallel F \parallel C \vee \ell(\vec{x}) \cdot \Theta, \Theta_r \implies \Gamma_1 \parallel F \parallel C'' \cdot \Theta'', \Theta''_r}$$

In the rule, the literal $\ell(\vec{z})$ is only contradictory in the context of the equalities $x \simeq y\Theta'$. That means there must be an argument i of ℓ such that projecting i-th component of ℓ results with $\Gamma_1^{\simeq}(\overline{\ell}\ |_i$ $) \bowtie \Theta_r \neq \emptyset$. In the rule we consider contradicting arguments of ℓ one by one. The Bridge relation P that was introduced in the proof of Lemma 3 encodes the use of the last asserted equalities to derive the contradiction. Just as in Lemma 3, we will bound the number of applications of the inserted equalities with m. The contradiction can be derived using finite number of equalities from both, $E(\Gamma)$ and the set of newly inserted equalities. That guarantees the existence of m.

4.3.3 E-CongResolve

The rule E-CongResolve is introduced to match E-CongConflict when the top-most literal is conflicting with an equality in the conflict clause. used for propagating an equality.

$$\frac{\begin{array}{rcl} \Gamma & = & \Gamma_0 \ell^{C' \vee \ell\Theta'}\Theta_\ell, \quad \ell \text{ is not} \simeq \\ C & = & (\ell_1 \vee \ldots \vee \ell_k \vee x \simeq y), \\ \Theta_r & = & \Theta \bowtie \Gamma^{\simeq}(\overline{\ell}_1) \bowtie \ldots \bowtie \Gamma^{\simeq}(\overline{\ell}_k) \bowtie \overline{E(\Gamma)}, \\ \Theta'' & = & \pi_{xy}\Theta_r \neq \emptyset, \\ \Gamma_1 & = & \Gamma x \simeq y \Theta'', \;\; [\Theta_\ell]_{E(\Gamma_1)} \bowtie \Gamma_1^{\simeq}(\overline{\ell}) \neq \emptyset \\ C_1\Theta_{r1} & = & \text{CongConflictInstance}(\Gamma, C, \Theta_r), \;\; \Theta_1 = \Theta \bowtie \Theta_{r1} \end{array}}{\Gamma \,\|\, F \,\|\, C \cdot \Theta, \Theta_r \implies \Gamma \,\|\, F \,\|\, C_1 \cdot \Theta_1, \Theta_{r1}}$$

An implied equality in the conflict clause contradicts the top-most literal assignment in the context, and the top-most literal assignment is not an equality (that case was handled using the E-Conflict rule). The side-conditions match the premises of lemma 3. The lemma ensures that there are $C_1 \cdot \Theta_1, \Theta_{r1}$ that can be derived from $C \cdot \Theta, \Theta_r$, such that resolution with the top-most literal can be applied.

The rules E-Resolve and E-CongResolve essentially handle the two cases where standard resolution does not yet apply. The premises for the two rules are disjoint: one handles the case where the top-most literal in Γ is an equality, the other handles the case where it isn't an equality. The rules both reduce the dependencies on the top-most literal for the conflict clause. The E-Resolve rule takes care of the use of transitive closure, the E-CongResolve rule takes care of the dependencies on conflicts caused indirectly from congruence (E-CongConflict). These observations are summarized in the following lemma.

Lemma 4. *In every reachable conflict state it is the case that:*

1. *If* E-Resolve *applies, then it is only applicable a finite number of times.*

2. *If* E-CongResolve *applies, then it is only applicable a finite number of times.*

In either case, these rules can be followed only by either a finite number of Factoring*, after which either* Resolve*, or* Refine *apply.*

4.3.4 Other conflict resolution rules

The other conflict resolution rules are direct liftings of the same rules that apply for DPLL($\mathcal{SX}$). The rules for DPLL($\mathcal{SX}$) were listed in Figure 4, and the rules for DPLL(SE) are listed in Figure 6.

4.4 Summarizing the DPLL(SE) calculus

We can now summarize the full DPLL(SE) calculus as comprising of the rules listed in Figure 6 as well as E-Decide, E-Decide$_\simeq$, E-UnitPropagate, E-Conflict, E-CongConflict, Resolve, E-Resolve, and E-CongResolve.

4.5 Examples

We here illustrate the rules on a couple of small example. Our first example exercises just the E-CongConflict rule. The second example shows how the conflict resolution rules are used.

$$\textsf{Skip}\ \frac{\Gamma_1 = \Gamma\overline{\ell}^{C_\ell\cdot\Theta'}\Theta_\ell,\ \ \delta_{\vec{y}\to\vec{x}}\pi_{\vec{y}}\Theta_r \bowtie [\Theta_\ell]_{E(\Gamma_1)} = \emptyset\ \text{ for every } \ell(\vec{y}) \in C}{\Gamma\overline{\ell}^{C_\ell\cdot\Theta'}\Theta_\ell \,\|\, F \,\|\, C\cdot\Theta, \Theta_r \implies \Gamma \,\|\, F \,\|\, C\cdot\Theta, \Theta_r}$$

$$\textsf{Factoring}\ \frac{\Theta_r' = \hat{\pi}_{\vec{z}}\sigma_{\vec{y}=\vec{z}}\,[\Theta_r]_{E(\Gamma)} \neq \emptyset,\ \Theta' = \hat{\pi}_{\vec{z}}\sigma_{\vec{y}=\vec{z}}\Theta}{\Gamma \,\|\, F \,\|\, (C(\vec{x}) \vee \ell(\vec{y}) \vee \ell(\vec{z}))\cdot\Theta, \Theta_r \implies \Gamma \,\|\, F \,\|\, (C(\vec{x}) \vee \ell(\vec{y}))\cdot\Theta', \Theta_r'}$$

$$\textsf{Learn}\ \frac{C\cdot\Theta \notin F}{\Gamma \,\|\, F \,\|\, C\cdot\Theta, \Theta_r \implies \Gamma \,\|\, F, C\cdot\Theta \,\|\, C\cdot\Theta, \Theta_r}$$

$$\textsf{Unsat}\ \Gamma \,\|\, F \,\|\, \Box\cdot\Theta, \Theta_r \implies \textsf{unsat} \qquad \textbf{if } \Theta \neq \emptyset$$

$$\textsf{Backjump}\ \frac{\begin{array}{l} C = (\ell_1 \vee \ldots \vee \ell_k \vee \ell(\vec{x})), \\ \Theta' = \pi_{\vec{x}}(\Theta \bowtie \Gamma^{\simeq}(\overline{\ell}_1) \bowtie \ldots \bowtie \Gamma^{\simeq}(\overline{\ell}_k)) \setminus \Gamma^{\simeq}(\ell) \neq \emptyset \end{array}}{\Gamma \diamond \Gamma' \,\|\, F \,\|\, C\cdot\Theta, \Theta_r \implies \Gamma\ell^{C\cdot\Theta}\Theta' \,\|\, F}$$

$$\Gamma \diamond \ell\Theta_1\Gamma' \,\|\, F \,\|\, C\cdot\Theta, \Theta_r \implies \Gamma \diamond \ell\Theta_1' \,\|\, F \qquad \textbf{if } \emptyset \neq \Theta_1' \subset \Theta_1$$

Figure 6: Remaining conflict resolution rules for DPLL(SE)

Example 9. *Let us check satisfiability of the following set of clauses:*

$$\begin{aligned} &x_1 \simeq x_2 \ \vee\ x_3 \not\simeq x_4 \ \cdot\{(a,c,b,c)\}, \\ &x_1 \simeq x_2 \ \vee\ x_3 \simeq x_4 \ \cdot\{(b,c,a,b)\}, \\ &x_1 \not\simeq x_2 \ \vee\ x_3 \not\simeq x_4 \ \cdot\{(a,b,a,c)\} \end{aligned}$$

A possible derivation can take the form:

Example 10. *Let us check satisfiability of the following set of clauses:*

$$\begin{aligned} &\overline{p}(a), \\ &p(f), \\ &a \simeq b \vee a \simeq c, \\ &e \simeq f \vee c \simeq f, \\ &c \simeq d, \\ &\forall x, \forall y.\ x \simeq y \vee \overline{q}(x,y), \\ &d \not\simeq f \vee q(a,c), \\ &a \not\simeq c \vee q(d,f) \\ &a \not\simeq b \vee q(b,c) \\ &e \not\simeq f \vee q(d,e) \end{aligned}$$

In order to enhance readability we did not use the notation of substitution sets. Furthermore, where it is obvious, we do not annotate literals with explanations. To show that the above set of clauses is unsatisfiable, a proof looks as follows:

4.6 Soundness, Completeness, Stuck-freeness, and Complexity

As we have now presented the full DPLL(SE) calculus let us summarize its properties. The more detailed justification for these properties follow the lines of the justification of the corresponding theorems in [6] together with lemma 3 and lemma 4.

$$
\begin{array}{ll}
 & \parallel F \\
\Longrightarrow & \mathsf{E{-}Decide}_{\simeq} \\
 & \diamond, x_1 \simeq x_2\{(a,c)\} \parallel F \\
\Longrightarrow & \mathsf{E{-}UnitPropagate} \\
 & \diamond, x_1 \simeq x_2\{(a,c)\}, x_1 \not\simeq x_2^{C_3\Theta_3}\{(a,b)\} \parallel F \\
\Longrightarrow & \mathsf{E{-}CongConflict}\text{ (the second clause), } b \simeq c \\
 & \diamond, x_1 \simeq x_2\{(a,c)\}, x_1 \not\simeq x_2^{C_3\Theta_3}\{(a,b)\} \parallel \\
 & \parallel F, (x_1 \not\simeq x_2 \vee x_3 \simeq x_4 \vee x_5 \simeq x_6) \cdot \{(a,c,a,b,a,b)\} \\
\Longrightarrow & \mathsf{E{-}Conflict}\text{ (the newly added clause)} \\
 & \diamond, x_1 \simeq x_2\{(a,c)\}, x_1 \not\simeq x_2^{C_3\Theta_3}\{(a,b)\} \parallel F_1 \parallel \ldots \\
 & \parallel (x_1 \not\simeq x_2 \vee x_3 \simeq x_4 \vee x_5 \simeq x_6) \cdot \{(a,c,a,b,a,b)\} \\
\Longrightarrow & \mathsf{Factoring} \\
 & \diamond, x_1 \simeq x_2\{(a,c)\}, x_1 \not\simeq x_2^{C_3\Theta_3}\{(a,b)\} \parallel F_1 \parallel (x_1 \not\simeq x_2 \vee x_3 \simeq x_4) \cdot \{(a,c,a,b)\} \\
\Longrightarrow & \mathsf{Resolve}\text{ (the conflicting and the third clause),} \\
 & \diamond, x_1 \simeq x_2\{(a,c)\} \parallel F_1 \parallel (x_1 \not\simeq x_2 \vee x_3 \not\simeq x_4) \cdot \{(a,c,a,c)\} \\
\Longrightarrow & \mathsf{Factoring} \\
 & \diamond, x_1 \simeq x_2\{(a,c)\} \parallel F_1 \parallel (x_1 \not\simeq x_2) \cdot \{(a,c)\} \\
\Longrightarrow & \mathsf{Backjump} \\
 & x_1 \not\simeq x_2^{(x_1 \not\simeq x_2)\cdot\{(a,c)\}}\{(a,c)\} \parallel F_1 \\
\Longrightarrow & \mathsf{E{-}UnitPropagate} \\
 & x_1 \not\simeq x_2^{(x_1 \not\simeq x_2)\cdot\{(a,c)\}}\{(a,c)\}, x_1 \not\simeq x_2^{C_1\Theta_1}\{(b,c)\} \parallel F_1 \\
\Longrightarrow & \mathsf{E{-}UnitPropagate} \\
 & x_1 \not\simeq x_2^{(x_1 \not\simeq x_2)\cdot\{(a,c)\}}\{(a,c)\}, x_1 \not\simeq x_2^{C_1\Theta_1}\{(b,c)\}, x_1 \simeq x_2^{C_2\Theta_2}\{(a,b)\} \parallel F_1
\end{array}
$$

Figure 7: Derivation for example 9

Theorem 2 (Soundness)**.** *The DPLL(SE) calculus is sound. Thus, whenever it establishes* Unsat*, from* $\epsilon \parallel F$*, then the set of clauses* F *are unsatisfiable.*

Proving soundness is a standard inspection of the derivation rules.

Theorem 3 (Completeness)**.** *The DPLL(SE) calculus is complete. Thus, whenever it saturates in a state* $\Gamma \parallel F$ *where no rule can be applied, then the set of clauses* F *is satisfiable using the assignment* Γ*. Conversely, if the set* F *is unsatisfiable, then the rules derive the empty clause* $\Box$*.*

Theorem 4 (Stuck-freeness)**.** *The transition rules of the DPLL(SE) calculus terminate. In particular the conflict resolution rules always admit a state where either* Backjump *or* Refine *are enabled.*

The proof of Theorem 4 is a direct lifting of the theorem for stuck-freeness of DPLL($\mathcal{SX}$) presented in [6], except we use rules E-CongConflict and E-Resolve to reduce the conflict clause to the cases where the rules from DPLL($\mathcal{SX}$) apply. Stuck-freeness corresponds to a confluence property: every strategy is ensured to make progress towards establishing whether the set of input clauses F are satisfiable.

Theorem 5 (Complexity)**.** *Similar to the DPLL($\mathcal{SX}$) calculus, the complexity of applying the calculus as an algorithm for DPLL(SE) requires up to doubly exponential time and up to exponential space.*

The justification for the theorem follows the same argument used in [6].

$$
\begin{array}{ll}
 & \parallel F \\
\Longrightarrow & \mathsf{UnitPropagate} \quad \text{(3 times)} \\
 & \overline{p}(a), p(f), c \simeq d \parallel F \\
\Longrightarrow & \mathsf{Decide} \quad \text{(2 times)} \\
 & \overline{p}(a), p(f), c \simeq d, \diamond a \simeq b, \diamond e \simeq f \parallel F \\
\Longrightarrow & \mathsf{UnitPropagate} \quad \text{(2 times)} \\
 & \overline{p}(a), p(f), c \simeq d, \diamond a \simeq b, \diamond e \simeq f, q(b,c), q(d,e) \parallel F \\
\Longrightarrow & \mathsf{E-CongConflict} + \mathsf{E-CongResolve} \text{ (as described in Example 8)} \\
 & \overline{p}(a), p(f), c \simeq d, \diamond a \simeq b, \diamond e \simeq f, q(b,c), q(d,e) \parallel F \parallel \overline{q}(b,c) \vee \overline{q}(d,e) \vee p(b) \vee \overline{p}(e) \\
\Longrightarrow & \mathsf{Resolve} \quad \text{(2 times)} \\
 & \overline{p}(a), p(f), c \simeq d, \diamond a \simeq b, \diamond e \simeq f \parallel F \parallel p(b) \vee \overline{p}(e) \vee a \not\simeq b \vee e \not\simeq f \\
\Longrightarrow & \mathsf{E-Resolve} + \mathsf{Factoring} \\
 & \overline{p}(a), p(f), c \simeq d, \diamond a \simeq b, \diamond e \simeq f \parallel F \parallel p(b) \vee \overline{p}(f) \vee a \not\simeq b \vee e \not\simeq f \\
\Longrightarrow & \mathsf{Backjump} \quad \text{where } C_1 = p(b) \vee \overline{p}(f) \vee a \not\simeq b \vee e \not\simeq f \\
 & \overline{p}(a), p(f), c \simeq d, \diamond a \simeq b, e \not\simeq f^{C_1} \parallel F \\
\Longrightarrow & \mathsf{UnitPropagate} \\
 & \overline{p}(a), p(f), c \simeq d, \diamond a \simeq b, e \not\simeq f^{C_1}, c \simeq f \parallel F \\
\Longrightarrow & \mathsf{UnitPropagate} \\
 & \overline{p}(a), p(f), c \simeq d, \diamond a \simeq b, e \not\simeq f^{C_1}, c \simeq f, q(a,c) \parallel F \\
\Longrightarrow & \mathsf{E-CongConflict} + \mathsf{E-CongResolve} \quad \text{(preventing to add } a \simeq c) \\
 & \overline{p}(a), p(f), c \simeq d, \diamond a \simeq b, e \not\simeq f^{C_1}, c \simeq f, q(a,c) \parallel F \parallel p(a) \vee \overline{p}(f) \vee \overline{q}(a,c) \\
\Longrightarrow & \mathsf{Resolve} \\
 & \overline{p}(a), p(f), c \simeq d, \diamond a \simeq b, e \not\simeq f^{C_1}, c \simeq f \parallel F \parallel p(a) \vee \overline{p}(f) \vee d \not\simeq f \\
\Longrightarrow & \mathsf{E-Resolve} \\
 & \overline{p}(a), p(f), c \simeq d, \diamond a \simeq b, e \not\simeq f^{C_1}, c \simeq f \parallel F \parallel p(a) \vee \overline{p}(f) \vee d \not\simeq c \vee c \not\simeq f \\
\Longrightarrow & \mathsf{2xResolve} + \mathsf{Factoring} \\
 & \overline{p}(a), p(f), c \simeq d, \diamond a \simeq b \parallel F \parallel p(a) \vee p(b) \vee \overline{p}(f) \vee a \not\simeq b \vee d \not\simeq c \\
\Longrightarrow & \mathsf{Resolve} + \mathsf{Factoring} \\
 & \overline{p}(a), p(f), c \simeq d, \diamond a \simeq b \parallel F \parallel p(a) \vee \overline{p}(f) \vee a \not\simeq b \vee d \not\simeq c \\
\Longrightarrow & \mathsf{Backjump} \quad \text{where } C_1 = p(a) \vee \overline{p}(f) \vee a \not\simeq b \vee d \not\simeq c \\
 & \overline{p}(a), p(f), c \simeq d, a \not\simeq b^{C_1} \parallel F \\
\Longrightarrow & \mathsf{2xUnitPropagate} \\
 & \overline{p}(a), p(f), c \simeq d, a \not\simeq b^{C_1}, a \simeq c, q(d,f) \parallel F \\
\Longrightarrow & \mathsf{E-CongConflict} + \mathsf{E-CongResolve} \quad \text{(preventing to add } d \simeq f) \\
 & \overline{p}(a), p(f), c \simeq d, a \not\simeq b^{C_1}, a \simeq c, q(d,f) \parallel F \parallel p(d) \vee \overline{p}(f) \vee \overline{q}(d,f) \\
\Longrightarrow & \text{after sequence of applying } \mathsf{Resolve}, \mathsf{E\text{-}Resolve}, \text{ and } \mathsf{Factoring} \\
 & \overline{p}(a), p(f), c \simeq d \parallel F \parallel p(a) \vee \overline{p}(f) \vee d \not\simeq c \\
\Longrightarrow & \mathsf{3xResolve} \\
 & \emptyset \parallel F \parallel \Box \\
\Longrightarrow & \mathsf{unsat}
\end{array}
$$

Figure 8: Derivation for example 10

5 Conclusions

We have presented a calculus for EPR with equality based on the Davis-Putnam-Logemann-Loveland procedure with substitution sets. By building in equality the calculus avoids explicitly adding equality axioms and allows for propagating equality facts implicitly during search. On the other hand, conflict resolution for backjumping and lemma learning necessitates reconstructing the dependencies on equalities.

Adding the theory of equality as a primitive to a calculus for EPR can be seen as one instance of adding theory reasoning to an EPR calculus. The wider programme is thus to add theory solvers in the context of an efficient EPR calculus. Just like there is a simple way of reducing EPR to propositional satisfiability by grounding, there is a simple way of adding theories to EPR calculi based on DPLL: for each ground model identified by a the core calculus, assert all ground facts to the theory solver. Our work with handling equality illustrates an approach that seeks an integration that does not require grounding.

Future work includes experimenting with the equality calculus and evaluating it relative to alternatives, such as the direct encoding of equality in EPR.

References

[1] Peter Baumgartner, Alexander Fuchs, Hans de Nivelle, and Cesare Tinelli. Computing Finite Models by Reduction to Function-Free Clause Logic. *Journal of Applied Logic*, July 2007. In Press, available online, doi:10.1016/j.jal.2007.07.005.

[2] Egon Börger, Erich Grädel, and Yuri Gurevich. *The Classical Decision Problem*. Springer-Verlag, 1997.

[3] Aaron R. Bradley, Zohar Manna, and Henny B. Sipma. What's decidable about arrays? In E. Allen Emerson and Kedar S. Namjoshi, editors, *VMCAI*, volume 3855 of *Lecture Notes in Computer Science*, pages 427–442. Springer, 2006.

[4] K. Claessen and N. Sorensson. New techniques that improve mace-style finite model finding. In *CADE-19 Workshop: Model Computation - Principles, Algorithms, Applications*, 2003.

[5] L. de Moura and N. Bjørner. Deciding Effectively Propositional Logic using DPLL and substitution sets. In Allesandro Armando, Peter Baumgartner, and Gilles Dowek, editors, *IJCAR 2008*, 2008.

[6] L. de Moura, R. Piskac, and N. Bjørner. Deciding Effectively Propositional Logic using DPLL and substitution sets. Technical Report MSR-2008-108, Microsoft Research, 2008.

[7] Hans de Nivelle and Jia Meng. Geometric resolution: A proof procedure based on finite model search. In Ulrich Furbach and Natarajan Shankar, editors, *IJCAR*, volume 4130 of *Lecture Notes in Computer Science*, pages 303–317. Springer, 2006.

[8] Pascal Fontaine. Combinations of theories and the Bernays-Schönfinkel-Ramsey class. In Bernhard Beckert, editor, *4th International Verification Workshop - VERIFY'07, Bremen, 15/07/07-16/07/07*, July 2007.

[9] R. Graham, B. Rothschild, and J. Spencer. *Ramsey Theory*. Wiley, 2 edition, 1990.

[10] J. A. Navarro Pérez. *Encoding and Solving Problems in Effectively Propositional Logic*. PhD thesis, The University of Manchester, 2007.

[11] Harry R. Lewis. Complexity results for classes of quantificational formulas. *J. Comput. Syst. Sci.*, 21(3):317–353, 1980.

[12] William McCune. Mace4 reference manual and guide. *CoRR*, cs.SC/0310055, 2003.

[13] F. Ramsey. On a problem of formal logic. *Proc. of the London Mathematical Society*, 30:264–286, 1930.

[14] 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. Springer, 2003.

[15] Andrei Voronkov. Merging relational database technology with constraint technology. In Dines Bjørner, Manfred Broy, and Igor V. Pottosin, editors, *Ershov Memorial Conference*, volume 1181 of *Lecture Notes in Computer Science*, pages 409–419. Springer, 1996.

[16] John Whaley, Dzintars Avots, Michael Carbin, and Monica S. Lam. Using datalog with binary decision diagrams for program analysis. In Kwangkeun Yi, editor, *APLAS*, volume 3780 of *Lecture Notes in Computer Science*, pages 97–118. Springer, 2005.

[17] Jian Zhang and Hantao Zhang. System description: Generating models by sem. In Michael A. McRobbie and John K. Slaney, editors, *CADE*, volume 1104 of *Lecture Notes in Computer Science*, pages 308–312. Springer, 1996.

[18] Lintao Zhang, Conor F. Madigan, Matthew W. Moskewicz, and Sharad Malik. Efficient conflict driven learning in boolean satisfiability solver. In *ICCAD*, pages 279–285, 2001.