Seventh International Conference on Formal Methods in Computer-Aided Design

# Lifting Propositional Interpolants to the Word-Level

Daniel Kroening
Computer Systems Institute
ETH Zurich

Georg Weissenbacher
Computer Systems Institute
ETH Zurich

***Abstract*— Craig interpolants are often used to approximate inductive invariants of transition systems. Arithmetic relationships between numeric variables require word-level interpolants, which are derived from word-level proofs of unsatisfiability. While word-level theorem provers have made significant progress in the past few years, competitive solvers for many logics are based on flattening the word-level structure to the bit-level. We propose an algorithm that lifts a resolution proof obtained from a bit-flattened formula up to the word-level, which enables the computation of word-level interpolants. Experimental results for equality logic suggest that the overhead of lifting the propositional proof is very low compared to the solving time of a state-of-the-art solver.**

## I. Introduction

Fifty years ago, William Craig showed that for each inconsistent pair of logical formulas $\langle A, B\rangle$, there exists a formula $\phi$ – the *Craig interpolant* – that is implied by $A$, inconsistent with $B$, and refers only to non-logical symbols common to $A$ and $B$ [1]. Intuitively, the interpolant $\phi$ can be understood as an abstraction of $A$. This result has been recently rediscovered and is the basis of various abstraction techniques in several automated verification tools. All of these verification techniques require an efficient decision procedure that is able to generate interpolants for unsatisfiable formulas. In program verification, the queries that typically arise are stated in quantifier-free Presburger arithmetic, equality- or difference-logic. These theories enjoy the *small-model* property, i.e., if a formula is satisfiable, then it has a satisfying assignment in a finite domain. Many decision procedures exploit this property and translate the original problem (either eagerly or lazily) into propositional logic. These instances can then be solved efficiently due to the recent advances in Boolean satisfiability solving [2]. The disadvantage of this approach is the loss of structure: Even though interpolants can be easily extracted from the resolution proofs provided by proof logging SAT solvers [3], it is prohibitively complicated to use the resulting bit-level interpolants in combination with word-level implementations or specifications.

*Contribution:* While interpolating decision procedures that generate word-level interpolants do exist [4], they are not yet competitive with state-of-the-art SAT-based theorem provers. We solve this problem by *lifting* existing resolution proofs generated by a SAT-based decision procedure to *word-level proofs*, from which the corresponding word-level interpolants can be easily extracted. Exemplarily, we provide a proof-lifting algorithm for equality logic.

*Related Work*

Our algorithm is not the first that constructs word-level interpolants: McMillan's theorem prover FOCI generates interpolants for quantifier free formulas with linear inequalities and uninterpreted function symbols [4]. His approach requires a tailor-made theorem prover for these theories. The technique presented in this paper is suitable for off-the-shelf, bit-level decision procedures, which are known to perform very well on a variety of logics, e.g., equality logic and bit-vector arithmetic.

Interpolants have various applications in software and hardware model checking. For instance, interpolation can be used to derive an abstract image operator from failed attempts to disprove a property of a finite state transition system. By computing a fix-point for this operator, one can obtain an inductive invariant of the transition system [5].

## II. Background

### A. Propositional Craig Interpolants

A propositional formula consists of *atoms* (indicated by Boolean identifiers $a_i$, $i = 1, 2, 3, \ldots$), the constants $\mathsf{true}$ and $\mathsf{false}$, and the operators $\wedge$, $\vee$, and $\neg$. We use $\overline{a_i}$ as an alternative notation for $\neg a_i$. A propositional formula is either an atom, or of the form $(F \wedge G)$, $(F \vee G)$ or $\overline{F}$, where $F$ and $G$ are also propositional formulas. We write $\mathcal{A}(F)$ to denote the set of atoms that occur in $F$. We use $\Sigma$ to denote a valuation to $\mathcal{A}(F)$. $\Sigma(a_i)$ denotes the truth value of the atom $a_i$ in $\Sigma$, and $\Sigma(a_i) = \bot$ denotes that $a_i$ is not assigned by $\Sigma$. $F$ is tautological if it evaluates to true for *all* Boolean valuations of $\mathcal{A}(F)$, and unsatisfiable if there is no such valuation.

A *literal* is either an atom or the negation of an atom. A disjunction of literals is called a *clause*. In clauses, we sometimes omit the disjunction operator $\vee$, e.g., we write $(\overline{a_1}\,\overline{a_2}\,a_3)$ instead of $(\overline{a_1} \vee \overline{a_2} \vee a_3)$. Furthermore, we use $\Box$ to denote the empty clause. A propositional formula of the form $F = \bigwedge_{i=1}^{n} \left(\bigvee_{j=1}^{m} L_{i,j}\right)$, with $L_{i,j} \in \{a_1, a_2 \ldots\} \cup \{\overline{a_1}, \overline{a_2}, \ldots\}$, is in conjunctive normal form (CNF). For each propositional formula, there exists a logically equivalent formula in CNF.

Given two clauses $a_i \vee \Theta$ and $\overline{a_i} \vee \Theta'$ (where $\Theta$ and $\Theta'$ represent disjunctions of literals), the *resolvent* of these clauses is the clause $\Theta \vee \Theta'$. The corresponding proof rule is known as *resolution*:

$$\textsc{Res}\,\frac{a_i \vee \Theta \qquad \overline{a_i} \vee \Theta'}{\Theta \vee \Theta'} \tag{1}$$

This research was supported by Microsoft Research through its European PhD Scholarship Programme, by SRC contract no. 2006-TJ-1539, and by an award from IBM Research.

0-7695-3023-0/07 $25.00 © 2007 IEEE
DOI 10.1109/.13

A propositional formula $F$ in CNF is unsatisfiable iff the empty clause can be derived by a sequence of such resolution steps starting from the clauses of $F$.

A sequence of resolution steps can be represented by a directed acyclic graph, where each vertex $c$ with no predecessor is a clause of $F$, and all other vertices are resolvents of their (exactly two) predecessors. Such a graph is a proof of unsatisfiability if its root node is the empty clause.

Given an unsatisfiable formula $F$ partitioned into two sets of clauses $\langle A, B\rangle$, an interpolant for $A$ and $B$ is is a formula $P$ such that

- $A$ implies $P$,
- $P \wedge B$ is unsatisfiable, and
- $P$ refers only to the common atoms of $A$ and $B$.

An interpolant for an unsatisfiable set of clauses $\langle A, B\rangle$ can be derived from a resolution proof in linear time [3], [4].

### B. Propositional Encodings of Decision Problems

In logics that are more expressive than propositional logic (e.g., equality logic), formulas may contain atoms that are specific to the theory that is used. We call these atoms the *Theory Atoms*. Let $\mathcal{A}^T(\varphi) \subseteq \mathcal{A}(\varphi)$ denote the set of Theory Atoms in $\varphi$ that are not Boolean identifiers or constants.

*Definition 1 (Propositional Skeleton):* The *Propositional Skeleton* of $\varphi$ is denoted by $\varphi_{sk}$ and is obtained by replacing every theorem atom $\alpha \in \mathcal{A}^T(\varphi)$ by a new Boolean identifier. The Boolean identifier that replaces $\alpha$ is denoted by $e_\alpha$.

We simply write $\mathcal{A}^T$ for $\mathcal{A}^T(\varphi)$ if the formula $\varphi$ is clear from the context.

*Definition 2 (Corresponding Constraint):* Given a truth assignment $\Sigma : \mathcal{A}^T \longrightarrow \{\mathsf{true}, \mathsf{false}, \perp\}$ to the theory atoms of formula $\varphi$, we define the *corresponding constraint* $\Psi(\Sigma)$ as the conjunction of the theory atoms $\alpha \in \mathcal{A}$ with $\Sigma(\alpha) \neq \perp$, where the theory atoms $\alpha$ with $\Sigma(\alpha) = \mathsf{false}$ are negated:

$$\Psi(\Sigma) := \bigwedge_{\alpha \in \mathcal{A}^T \mid \Sigma(\alpha) \neq \perp} \begin{cases} \alpha & : \ \Sigma(\alpha) = \mathsf{true} \\ \neg\alpha & : \ \Sigma(\alpha) = \mathsf{false} \end{cases}$$

## III. Interpolation for Equality Logic

### A. Flattening Equality Logic

We describe how to lift propositional proofs for equality logic to the word level. Equality logic permits formulas with an arbitrary Boolean structure, but limits atoms to Boolean variables and equalities of the form $x = y$, where $x$, $y$ are variables or numeric constants over some infinite domain $D$.[1] We call these non-Boolean variables the *theory variables*, and denote the formula by $\varphi$.

Theorem provers for equality logic and uninterpreted functions are reasonably efficient, and usually rely on a combination of a propositional SAT solver and an implementation of *Congruence Closure*, which is based on the union-find algorithm. They compute a propositional encoding in a lazy manner. Nevertheless, algorithms that perform *range allocation* for equality logic are still superior to procedures that compute encodings lazily.

The idea of range allocation is to compute a bounded range of integer values for the theory variables in $\varphi$. This range is constructed such that it is sufficient for a satisfying assignment of $\varphi$, if one exists. We restrict the presentation to ranges that can be encoded with bit-vectors, i.e., values from 0 to $2^n - 1$ for some integer $n$.

*Definition 3 (Range):* Let $V$ denote the set of theory variables in $\varphi$. A *range* $R : V \longrightarrow \mathbb{N}$ is an assignment of a number of bits to each variable in $V$. We denote the Boolean variable that encodes bit $i \in \{0, \ldots, R(x) - 1\}$ for $x \in V$ by $x_i$. These Boolean variables are called *vector variables*.

We assume that the range is *consistent*, i.e., for any equality $x = y$ in $\varphi$, $R(x) = R(y)$ holds.

*Definition 4 (Small-Domain Assignment):* An assignment $\sigma$ to the variables $V$ with $\sigma(x) \in \{0, \ldots, 2^{R(x)} - 1\}$ is called a *small-domain assignment*.

*Definition 5 (sufficient):* A range is *sufficient* for a formula $\varphi$ if there is a small-domain assignment $\sigma$ for every satisfiable corresponding constraint $\Psi(\Sigma)$ (see Def. 2).

Given a consistent range, the variables are interpreted as bit-vectors with just enough bits to encode the values in the range. Formally, we denote the propositional constraint for $x = y$, where $x$ and $y$ are non-Boolean variables, by $E(x = y)$. Let $n = R(x) = R(y)$. The propositional constraint is defined as follows:

$$E(x = y) \ :\Longleftrightarrow \ e_{x=y} \longleftrightarrow \bigwedge_{i=0}^{n-1} x_i \longleftrightarrow y_i \qquad (2)$$

Recall that $e_{x=y}$ is the variable used to replace $x = y$ in the skeleton $\varphi_{sk}$. We assume that the constants in the formula are mapped to the range. The propositional encoder for an equality between a variable and a constant is straight-forward.

*Lemma 1:* Given a sufficient range, $\varphi$ and

$$\varphi_{sk} \wedge \bigwedge_{\alpha \in \mathcal{A}^T} E(\alpha) \qquad (3)$$

are equi-satisfiable. We denote formula (3) as $\varphi_{enc}$.

We assume that a sufficient range is given. An obviously sufficient range is $R(v) = \lceil \log_2 |V| \rceil$ [6]. Procedures to compute a smaller but still sufficient range are beyond the scope of this article. The techniques we propose are still applicable even if smaller ranges are used as long as the range is sufficient according to Def. 5.

*Example 1:* Consider the formula

$$x = y \wedge y = z \wedge z \neq x$$

as a running example. A sufficient range is $R(x) = R(y) = R(z) = 2$, which results in the following propositional encoding:

$$\begin{array}{llll} & e_{x=y} \longleftrightarrow & (x_0 \longleftrightarrow y_0 \wedge x_1 \longleftrightarrow y_1) & \text{for } x = y, \\ \wedge & e_{y=z} \longleftrightarrow & (y_0 \longleftrightarrow z_0 \wedge y_1 \longleftrightarrow z_1) & \text{for } y = z, \\ \wedge & e_{z=x} \longleftrightarrow & (z_0 \longleftrightarrow x_0 \wedge z_1 \longleftrightarrow x_1) & \text{for } z = x, \\ \wedge & & e_{x=y} \wedge e_{y=z} \wedge \overline{e_{z=x}} & \text{for } \varphi_{sk} \end{array}$$

[1] A construction of a word-level proof that uses only the tree proof rules for equality given in Sec. III-D from a bit-level refutation is not always possible for a finite domain.

### B. Bit-Level Resolution Proofs for Equality Logic

The propositional formula $\varphi_{enc}$ can be converted into CNF, and is then passed to a propositional SAT solver. In case $\varphi$ is unsatisfiable, so is $\varphi_{enc}$, and we can obtain a propositional resolution proof from the SAT solver. The form of this proof depends on the particular encoding that is used to convert $\varphi_{enc}$ into CNF. We define a particular encoding in order to record claims about the propositional resolution proofs that are obtained. However, the method is not limited to a particular encoding.

We restrict the presentation to CNF obtained by means of Tseitin's encoding [7], which is the basic technique behind most tools that generate CNF.

*Definition 6 (Bit-Flattening of E):* The propositional constraint $E(x = y)$ is transformed into CNF using auxiliary variables $o_0, \ldots, o_{n-1}$, where $o_i$ holds if $x_i \longleftrightarrow y_i$. Note that these auxiliary variables are specific to $E(x = y)$, and are not shared with other constraints.

$$\begin{aligned}&E(x=y) \iff \\ &\bigwedge_{i=0}^{n-1} (\overline{x_i y_i} o_i) \wedge (x_i y_i o_i) \wedge (\overline{x_i} y_i \overline{o_i}) \wedge (x_i \overline{y_i o_i}) \\ &\wedge \bigwedge_{i=0}^{n-1} (o_i \overline{e_{x=y}}) \wedge (\overline{o_0} \ldots \overline{o_{n-1}} e_{x=y})\end{aligned} \tag{4}$$

This encoding may be optimized in numerous ways, e.g., many clauses can be omitted if the polarity of the atoms is exploited. These techniques are beyond the scope of this article, but our method is still applicable after applying the commonly used optimizations. The propositional skeleton $\varphi_{sk}$ is transformed by similar means, i.e., introducing a new auxiliary variable for each node of the parse-tree of $\varphi_{sk}$.

*Example 2:* Continuing our running example from the previous subsection, the following set of clauses[2] is a CNF encoding of the formula $\varphi_{enc}$:

for $x = y$:

$x_0\overline{y_0 o_{10}}$ $\overline{x_0} y_0 \overline{o_{10}}$ $x_1 \overline{y_1 o_{11}}$ $\overline{x_1} y_1 \overline{o_{11}}$ $o_{10}\overline{e_{x=y}}$ $o_{11}\overline{e_{x=y}}$

for $y = z$:

$y_0\overline{z_0 o_{20}}$ $\overline{y_0} z_0 \overline{o_{20}}$ $y_1 \overline{z_1 o_{21}}$ $\overline{y_1} z_1 \overline{o_{21}}$ $o_{20}\overline{e_{y=z}}$ $o_{21}\overline{e_{y=z}}$

for $z = x$:

$x_0 z_0 o_{30}$ $\overline{x_0 z_0} o_{30}$ $x_1 z_1 o_{31}$ $\overline{x_1 z_1} o_{31}$ $\overline{o_{30} o_{31}} e_{z=x}$

for $\varphi_{sk}$:

$e_{x=y}$ $e_{y=z}$ $\overline{e_{z=x}}$

The variables $o_{10}$, $o_{11}$, and so on are auxiliary variables added for Tseitin's encoding of the conjunction required for decomposing the equalities.

### C. Lifting Bit-Level Proofs for Equality Logic

We represent propositional resolution proofs $P$ as a binary tree, where the nodes represent the clauses that were resolved from their antecedent nodes using the RES rule (see (1) in Section II-A). For instance, Figure 1(a) shows the resolution tree for the set of clauses given in Example 2. We write $root(P)$ for the root-node of $P$. We write $leaf(n)$ if $n$ is a leaf node. Otherwise, we write $L(n)$ and $R(n)$ for the left and right antecedent nodes, respectively. We write $\mathcal{L}(n)$ to denote the fact at node $n \in P$ (the *label* of node $n$). For leaf-nodes that are generated for a specific atom (as opposed to the skeleton), we write $\mathcal{A}^T(n)$ to denote the theory atom that the clause was generated for. A proof $P$ shows unsatisfiability of a formula $\varphi$ if its leaves are trivially implied by $\varphi$ and $\mathcal{L}(root(P)) = \Box$. We write $\varphi \vdash_P \Box$ in this case.

Our proof-lifting algorithms take a propositional resolution proof $P$ for $\varphi_{enc}$ as input, and lift it up to a new proof for $\varphi$ that uses a word-level logic by replacing the labels of the nodes. The structure of the graph is not changed, up to a final minimization step.

For the theory of equality and the encoding described above, the lifted propositions have one of the following three forms, where $T_i$ denotes a theory predicate:

$F1$ $T_1 \wedge \ldots \wedge T_k$, or
$F2$ $T_1 \wedge \ldots \wedge T_k \vee \Theta$, where $\Theta$ is a propositional clause, or
$F3$ a propositional CNF-clause, including the empty clause.

We define the following *lifting-function* $\lambda(n)$ for the propositions that are used to label the leaf-nodes of $P$ by means of a case-split on the clause $\mathcal{L}(n)$.

$$\lambda(n) \quad := \quad \begin{cases} x \neq y \vee o_i & : \quad \text{for } \mathcal{L}(n) = (\overline{x_i y_i} o_i) \\ x \neq y \vee o_i & : \quad \text{for } \mathcal{L}(n) = (x_i y_i o_i) \\ x = y \vee \overline{o_i} & : \quad \text{for } \mathcal{L}(n) = (\overline{x_i} y_i \overline{o_i}) \\ x = y \vee \overline{o_i} & : \quad \text{for } \mathcal{L}(n) = (x_i \overline{y_i o_i}) \\ \mathcal{L}(n) & : \quad \text{otherwise} \end{cases} \tag{5}$$

We note that the clauses generated by $\lambda(n)$ for leaf-nodes have either one of the forms $F2$ or $F3$. It is easy to see that the label returned by $\lambda$ is trivially implied by $\varphi$ for all leaf-nodes $n$ of $P$.

The inner nodes of $P$ (i.e., those that are the result of a resolution step) are also lifted. Let

$$\begin{aligned}\lambda(L(n)) &= T_1^L \wedge \ldots \wedge T_j^L \vee \Theta, \quad \text{and} \\ \lambda(R(n)) &= T_1^R \wedge \ldots \wedge T_k^R \vee \Theta'\end{aligned}$$

be the lifted labels of the two antecedent nodes of $n$, where $\Theta$ and $\Theta'$ stand for (possibly empty) propositional clauses. We define $T^n$ to denote the conjunction

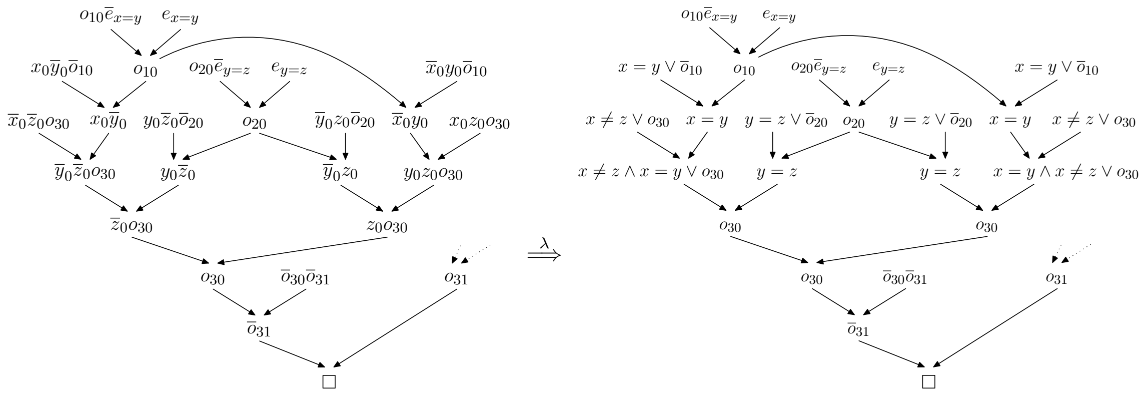
$$T_1^L \wedge \ldots \wedge T_j^L \wedge T_1^R \wedge \ldots \wedge T_k^R .$$

There are two cases, depending on whether the resolution that results in node $n$ is performed on a vector variable:

- If the resolution for node $n$ is performed by resolving on a vector variable, we define $\lambda(n) := T^n \vee \Theta \vee \Theta'$ if $T^n$ is satisfiable, and $\lambda(n) := \Theta \vee \Theta'$ otherwise.
- Otherwise, the resolution for node $n$ is performed on a non-vector variable. Let $\text{RES}(\Theta, \Theta')$ denote the resolvent of $\Theta$ and $\Theta'$ according to Rule (1) in Section II-A. Then, $\lambda(n) := T_1^L \wedge \ldots \wedge T_j^L \vee T_1^R \wedge \ldots \wedge T_k^R \vee \text{RES}(\Theta, \Theta')$.

Note that the second transformation may violate the assumption that the lifted clauses are always of form $F1$, $F2$, or $F3$.

[2] In order to simplify the presentation, the clauses that are trivially satisfied after propagation of the unit-clauses $e_{x=y}$, $e_{y=z}$, and $\overline{e_{z=x}}$ are omitted.


$o_{10}\overline{e}_{x=y}$
$e_{x=y}$
$x_0\overline{y}_0\overline{o}_{10}$
$o_{10}$
$o_{20}\overline{e}_{y=z}$
$e_{y=z}$
$\overline{x}_0y_0\overline{o}_{10}$
$\overline{x}_0\overline{z}_0o_{30}$
$x_0\overline{y}_0$
$y_0\overline{z}_0\overline{o}_{20}$
$o_{20}$
$\overline{y}_0z_0\overline{o}_{20}$
$\overline{x}_0y_0$
$x_0z_0o_{30}$
$\overline{y}_0\overline{z}_0o_{30}$
$y_0\overline{z}_0$
$\overline{y}_0z_0$
$y_0z_0o_{30}$
$\overline{z}_0o_{30}$
$z_0o_{30}$
$o_{30}$
$\overline{o}_{30}\overline{o}_{31}$
$o_{31}$
$\overline{o}_{31}$
$\Box$
$\overset{\lambda}{\Longrightarrow}$
$o_{10}\overline{e}_{x=y}$
$e_{x=y}$
$x = y \vee \overline{o}_{10}$
$o_{10}$
$o_{20}\overline{e}_{y=z}$
$e_{y=z}$
$x = y \vee \overline{o}_{10}$
$x \neq z \vee o_{30}$
$x = y$
$y = z \vee \overline{o}_{20}$
$o_{20}$
$y = z \vee \overline{o}_{20}$
$x = y$
$x \neq z \vee o_{30}$
$x \neq z \wedge x = y \vee o_{30}$
$y = z$
$y = z$
$x = y \wedge x \neq z \vee o_{30}$
$o_{30}$
$o_{30}$
$o_{30}$
$\overline{o}_{30}\overline{o}_{31}$
$o_{31}$
$\overline{o}_{31}$
$\Box$

(a) Resolution proof

(b) Lifted proof

Fig. 1. Resolution graph and the corresponding lifted proof for the clauses in Example 2. The right part of the respective graphs is symmetric to to the left part and has therefore been omitted.

Therefore, it is only performed if either $T_1^L \wedge \ldots \wedge T_j^L$ is equal to $T_1^R \wedge \ldots \wedge T_k^R$, or at least one of these terms is false. In all other cases, the lifting of the propositional proof fails. This case occurs in none of our benchmarks (see Section IV). The satisfiability of $T^n$ can be checked efficiently using a union-find data structure.

*Theorem 1:* The new labels $\lambda(n)$ imply the old labels $\mathcal{L}(n)$.

*Proof:* (By induction) For the leaf nodes, one can easily show that $\lambda(n)$ implies $\mathcal{L}(n)$ by flattening the theory atom in $\mathcal{L}(n)$ according to Definition 6. This constitutes the base case.

It remains to show that $\lambda(n)$ implies $\mathcal{L}(n)$ for an inner node $n$. By our induction hypothesis, $\lambda(L(n))$ implies $\mathcal{L}(L(n))$ (and $\lambda(R(n)) \Rightarrow \mathcal{L}(R(n))$, respectively). Observe that the lifting function $\lambda$ preserves the propositional structure of $\Theta$, the clause over the non-vector variables: If $\mathcal{L}(L(n))$ is $\bigvee_{i=1}^k L_i^L \vee \Theta$, where each $L_i^L$ denotes a literal for a vector variable, and $\Theta$ contains no vector variables, then $\lambda(L(n))$ is of the form $T_1^L \wedge \ldots \wedge T_j^L \vee \Theta$.

First we consider the case that resolution is performed on a vector variable. We prove

$$(T_1^L \wedge \ldots \wedge T_j^L) \wedge (T_1^R \wedge \ldots \wedge T_k^R) \;\vee\; (\Theta \vee \Theta') \Rightarrow \\ \textsc{Res}(\bigvee_{i=1}^{l} L_i^L, \bigvee_{i=1}^{m} L_i^R) \vee (\Theta \vee \Theta') \tag{6}$$

by performing a case split over values of the elements of the disjunction on the left side of the implication: If $(\Theta \vee \Theta')$ is true, or the expression on the left side is false, then the implication holds trivially. In the remaining case, $\Theta \vee \Theta'$ is false, and $(T_1^L \wedge \ldots \wedge T_j^L) \wedge (T_1^R \wedge \ldots \wedge T_k^R)$ holds. Then $T_1^L \wedge \ldots \wedge T_j^L$ implies $\bigvee_{i=1}^l L_i^L$, and $T_1^R \wedge \ldots \wedge T_k^R$ implies $\bigvee_{i=1}^m L_i^R$ by our induction hypothesis, and (6) holds.

For the remaining case (i.e., when we perform resolution on a non-vector variable), we have to show that

$$(T_1^L \wedge \ldots \wedge T_j^L) \vee (T_1^R \wedge \ldots \wedge T_k^R) \;\vee\; \textsc{Res}(\Theta, \Theta') \Rightarrow \\ \bigvee_{i=1}^{l} L_i^L \vee \bigvee_{i=1}^{m} L_i^R \vee \textsc{Res}(\Theta \vee \Theta')$$

This is trivial, since the new label $\lambda(n)$ as well as the old label $\mathcal{L}(n)$ is obtained from the predecessors of $n$ by resolution. ■

It is left to show that the lifted labeling actually corresponds to a proof.

*Theorem 2:* For all inner nodes $n$ of $P$, the new label is implied by the conjunction of the new antecedent labels:

$$\lambda(L(n)) \wedge \lambda(R(n)) \;\Rightarrow\; \lambda(n) \tag{7}$$

*Proof:* In the case that the resolution is performed on a vector variable, we have to show that

$$\big((T_1^L \wedge \ldots \wedge T_j^L) \vee \Theta\big) \wedge \big((T_1^R \wedge \ldots \wedge T_k^R) \vee \Theta'\big) \Rightarrow \\ (T_1^L \wedge \ldots \wedge T_j^L) \;\wedge\; (T_1^R \wedge \ldots \wedge T_k^R) \;\vee\; \Theta \vee \Theta'$$

holds. This can be easily achieved by applying the distributive law to the left side of the implication.

The remaining case is trivial, since the $\lambda(n)$ is obtained by performing resolution on the labels of $L(n)$ and $R(n)$. ■

The lifting-function $\lambda$ can be applied recursively to a proof $P$ together with labels $\mathcal{L}$, beginning with the leaf-nodes, to produce a new set of labels. We write $\lambda(P)$ for this new proof.

*Theorem 3:* Let $P$ denote a proof with $\varphi_{enc} \vdash_P \Box$, let $\lambda$ denote a lifting-function, and $P' := \lambda(P)$ a proof generated by lifting $P$ using $\lambda$. If the lifting function has the following properties, then $P'$ shows unsatisfiability of $\varphi$:

1) The lifted leaf-node labels are implied by $\varphi$,
2) the lifted inner node labels imply the old labels.

*Proof:* Observe that $\mathcal{L}(\mathit{root}(P)) = \Box$. Because of the second premise, $\lambda(\mathit{root}(P)) = \Box$. As the leaves $\lambda(\mathit{leaf}(n))$ are implied by $\varphi$, and $P'$ is a proof, we have $\varphi \vdash_{P'} \Box$. ■

### D. Interpolation using Lifted Proofs

In this section we explain how an interpolant can be derived from the lifted proof $\lambda(P)$. Note that the all lifted labels are of form $F1$, $F2$ or $F3$ (see Section III-C). Our implementation guarantees that the formula $\mathcal{P}(T_1, \ldots, T_k)$ over the theory predicates is always of form $T_1 \wedge \ldots \wedge T_k$.

The decision procedure for equality logic (as presented in [4]) makes use of transitivity, reflexivity, and contradiction.

The following two rules are sufficient to decide pure equality logic:

$$\textsc{Trans}\,\frac{t_1 = t_2 \quad t_2 = t_3}{t_1 = t_3} \qquad \textsc{EqNeq}\,\frac{t_1 = t_2}{\textsf{false}}\,\neg(t_1 = t_2)$$

If the formula is unsatisfiable, then a contradictory fact $(t_i = t_j) \wedge (t_i \neq t_j)$ can be inferred for some $i, j$.

A chain of equalities $(x = t_1) \wedge (t_1 = t_2) \wedge \cdots \wedge (t_n = y)$ can be partitioned into maximal sub-chains $(t_i = t_{i+1}) \cdots (t_{j-1} = t_j)$, $i < j$ consisting of either only equalities from $A$, or only equalities from $B$. Each subchain can then be summarized by $(t_i = t_j)$. We maintain such summaries for each node of the lifted proof. By constructing summaries from $A$ such that the terms $t_i$ and $t_j$ also occur in $B$ (i.e., are *global*) one can obtain an interpolant for $A$ and $B$.

*Example 3:* Consider the lifted proof in Figure 1(b), and the partitioning $A = (x = y) \wedge (y = z)$ and $B = (z \neq x)$. The variable $y$ is local to $A$, while $x$ and $z$ are global.

Consider the resolvent of $x \neq z \wedge x = y \vee o_3 0$ and $y = z$. Using transitivity, we derive

$$\textsc{Trans}\,\frac{x = y \quad y = z}{x = z}\,\{(x = y), (y = z)\} \subseteq A$$

and obtain a maximal chain $x = z$ as summary for $A$. We obtain a contradiction by applying the EQNEQ rule:

$$\textsc{EqNeq}\,\frac{x = z}{\textsf{false}}\,\neg(z = x) \in B$$

This yields $(x = z)$ as an interpolant for $(x = y) \wedge (y = z) \wedge (z \neq x)$. We apply the rules presented in [4] to the remaining propositional resolution steps of the lifted proof. Since the propositional structure of $\varphi$ contains no disjunction, this results in $(x = z)$ as the interpolant for $\langle A, B \rangle$.

## IV. Experimental Results

We have implemented the techniques described above in a tool called LIFTER. It uses MiniSat [2] as the solver for the encoding $\varphi_{enc}$.

We compare the performance of LIFTER with the SMT solver Yices [8] and with FOCI [4]. Of these provers, only FOCI and LIFTER are interpolating. We nevertheless include Yices in the comparison as a reference point.[3]

The benchmarks we use have been suggested in [6], [9], and are available for download. We remove 'easy' benchmarks from the set by eliminating a benchmark if all solvers are able to solve it within $0.1s$ or less. All formulas we consider are unsatisfiable. For the purpose of computing an interpolant, we split the formulas into two parts. We plot the run-time of LIFTER and Yices in Fig. 2. We omit the comparison with FOCI, as Yices outperforms FOCI on most of these queries. The experiments show that on this set of benchmarks, the range encoding is typically faster than the DPLL(T)-based solver. The run-time of LIFTER is dominated by the time taken to build $\varphi_{enc}$ and transform it into CNF. Both the time taken by MiniSat and the lifting-procedure are negligible. The lifting-procedure was always able to generate a word-level proof.

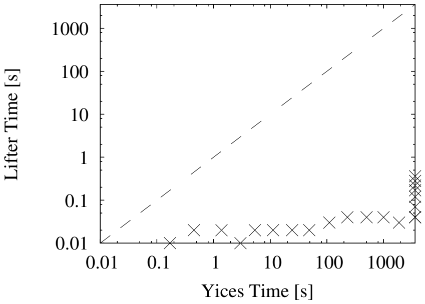


Fig. 2. Comparison of LIFTER and Yices on Equality Logic formulas

## V. Conclusion

We present a procedure for constructing a word-level proof for equality logic. We do not rely on a deductive engine to build a proof for the original formula, but instead transform an existing bit-level proof to the word-level. Thus, the deductive engine only has to work on a restricted set of theory-literals, and does not perform any case-splitting. From the resulting proofs, word-level Craig interpolants can be obtained. The procedure generates a word-level interpolant for all of our equality logic benchmarks. As future work, we plan to lift proofs obtained from highly optimized incremental encodings for bit-vector arithmetic, e.g., the one described in [10].

## Acknowledgments

We would like to thank Angelo Brillout and Christoph Wintersteiger for their helpful comments on this paper.

## References

[1] W. Craig, "Linear reasoning. A new form of the Herbrand-Gentzen theorem." *Journal of Symbolic Logic*, vol. 22, no. 3, pp. 250–268, 1957.

[2] N. Eén and N. Sörensson, "An extensible SAT-solver," in *Theory and Applications of Satisfiability Testing (SAT)*, 2003, pp. 502–518.

[3] P. Pudlák, "Lower bounds for resolution and cutting plane proofs and monotone computations," *The Journal of Symbolic Logic*, vol. 62, no. 3, pp. 981–998, 1997.

[4] K. L. McMillan, "An interpolating theorem prover," *Theoretical Computer Science*, vol. 345, no. 1, pp. 101–121, 2005.

[5] ——, "Interpolation and SAT-based model checking," in *Computer Aided Verification (CAV)*, ser. LNCS, vol. 2725. Springer, 2003, pp. 1–13.

[6] A. Pnueli, Y. Rodeh, O. Strichman, and M. Siegel, "The small model property: How small can it be?" *Information and Computation*, vol. 178, no. 1, pp. 279–293, 2002.

[7] G. Tseitin, "On the complexity of proofs in poropositional logics," in *Automation of Reasoning: Classical Papers in Computational Logic 1967–1970*, J. Siekmann and G. Wrightson, Eds., vol. 2. Springer, 1983, originally published 1970.

[8] B. Dutertre and L. de Moura, "The Yices SMT solver," Available at `http://yices.csl.sri.com/tool-paper.pdf`, 2006.

[9] Y. Rodeh and O. Strichman, "Building small equality graphs for deciding equality logic with uninterpreted functions." *Information and Computation*, vol. 204, no. 1, pp. 26–59, 2006.

[10] R. E. Bryant, D. Kroening, J. Ouaknine, S. A. Seshia, O. Strichman, and B. Brady, "Deciding bit-vector arithmetic with abstraction," in *Tools and Algorithms for the construction and Analysis of Systems (TACAS)*, ser. LNCS, vol. 4424. Springer, 2007, pp. 358–372.

[3] Yices won the most recent SMT competition in all divisions. We also ran CVC3 on all benchmarks, and found that Yices always outperforms CVC3. We therefore do not include CVC3 in the comparison.