Available online at www.sciencedirect.com
ScienceDirect
Electronic Notes in Theoretical Computer Science
ELSEVIER
Electronic Notes in Theoretical Computer Science 198 (2008) 3–17
www.elsevier.com/locate/entcs

Generating Minimum Transitivity Constraints in P-time for Deciding Equality Logic

Mirron Rozanov and Ofer Strichman

Information Systems Engineering, IE, Technion, Israel.
mirron@tx.technion.ac.il ofers@ie.technion.ac.il

Abstract

In a CAV'05 paper [6] we introduced a new decision procedure for Equality Logic: each equality predicate is encoded with a Boolean variable, and then a set of transitivity constraints are added to compensate for the loss of transitivity of equality. The constraints are derived by analyzing *Contradictory Cycles*: cycles in the equality graph with exactly one disequality. Such a cycle is called *constrained under a formula* φ if φ is not satisfied with an assignment of TRUE to all equality edges and FALSE to the disequality edge. While we proved in [6] that it is sufficient to constrain all simple contradictory cycles, we left open the question of how to find the necessary constraints in polynomial time. Instead, we showed two possible compromises: an exponential algorithm, or, alternatively, a polynomial approximation that constrains *all* contradictory cycles rather than only the simple ones. In this article we show a polynomial algorithm that constrains only the simple contradictory cycles.

Keywords: Equalities with Uninterpreted Functions

1 Introduction

Equality Logic with Uninterpreted Functions is a major decidable logic used in verification of infinite-state systems. Well-formed expressions in this logic are Boolean combinations of equality predicates, where the equalities are defined between *term-variables* (variables with some infinite domain) and Uninterpreted Functions. The Uninterpreted Functions can be reduced to equalities via, e.g., Ackermann's reduction [1], hence the underling theory that is left to solve is that of Equality Logic. We refer the reader to [6] for a description of some of the usage cases of this logic and a survey of previous work on decision procedures for it.

doi:10.1016/j.entcs.2008.04.077

The following framework is used by [2,6] and the current work to reduce the problem of deciding whether an Equality Logic formula φ^{E} is satisfiable, to the problem of deciding a propositional formula:

(i) Let E denote the set of equality predicates appearing in φ^{E}. Derive a Boolean formula $\mathcal{B}$ by replacing each equality predicate $(x_i = x_j) \in E$ with a new Boolean variable $e_{i,j}$. Encode disequality predicates with negations, e.g., encode $i \neq j$ with $\neg e_{i,j}$.

(ii) Recover the lost transitivity of equality by conjoining $\mathcal{B}$ with explicit *transitivity constraints* jointly denoted by $\mathcal{T}$ ($\mathcal{T}$ for $\mathcal{T}$ransitivity). $\mathcal{T}$ is a formula over $\mathcal{B}$'s variables and, possibly, auxiliary variables.

The Boolean formula $\mathcal{B} \wedge \mathcal{T}$ should be satisfiable if and only if φ^{E} is satisfiable. Further, it should be possible to construct a satisfying assignment to φ^{E} from an assignment to the $e_{i,j}$ variables.

In an earlier work [6] this framework was instantiated as follows (the terms that follow will be formally defined in Section 2). The transitivity constraints are derived by analyzing *Contradictory Cycles*: cycles in the Equality Graph with exactly one disequality. Such a cycle is called *constrained under a formula* φ if φ is not satisfied with an assignment of TRUE to all equality edges and FALSE to the disequality edge. While it was proven in [6] that it is sufficient to constrain all simple contradictory cycles, the question of how to find the necessary constraints in polynomial time was left open. Instead, two possible compromises were suggested: an exponential algorithm [5], or, alternatively, a polynomial approximation that constrains *all* contradictory cycles rather than only the simple ones. In this article we show a polynomial algorithm that constrains only the simple contradictory cycles.

While this article only replaces one (crucial) component in a previously-published decision procedure [6], it is written with the goal of being self-contained, assuming most readers are not familiar with the previous work [1]. In the next section we list several basic definitions that are necessary for understanding the setting; in Section 3 the main theorem on which this work (as well as [6]) is based on is re-presented; in Section 4 we describe the new decision procedure, and we conclude with a list of experiments in Section 5. A detailed comparison to [6,5] appears in Section 4.2.

2 Basic Definitions

The equality formula φ^{E} is assumed to be given in Negation Normal Form (NNF), which means that negations are only applied to atoms, or equality

[1] Some of the definitions and examples from [6] are in fact repeated here without change.

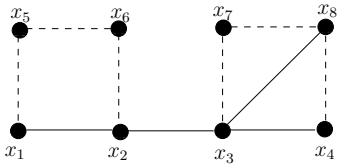


Fig. 1. An Equality Graph $G^{\mathrm{E}}(\varphi^{\mathrm{E}})$

predicates in our case. Every formula can be transformed to this form in linear time in the size of the formula. Given an NNF formula, denote by $E_=$ the set of (unnegated) equality predicates, and by $E_\neq$ the set of disequalities (negated) equality predicates. The Reduced Transitivity Constraints (RTC) method of [6] relies on graph-theoretic concepts.

Definition 2.1 [Equality Graph] Given an Equality Logic formula φ^{E}, the *Equality Graph* corresponding to φ^{E}, denoted by $G^{\mathrm{E}}(\varphi^{\mathrm{E}})$, is an undirected graph $(V, E_=, E_\neq)$ where each node $v \in V$ corresponds to a variable in φ^{E}, and each edge in $E_=$ and $E_\neq$ corresponds to an equality or disequality from the respective equality predicates sets $E_=$ and $E_\neq$. By convention $E_=$ edges are dashed and $E_\neq$ edges are solid.

Every edge in the Equality Graph corresponds to a variable $e_{i,j} \in \mathcal{B}$. It follows that when we refer to an assignment of an edge, it should be understood as an assignment to the *variable* that corresponds to this edge. Also, we will simply write G^{E} to denote an Equality Graph when not referring to a specific formula.

Note that Equality Graphs abstract the formulas from which they are built: they ignore the Boolean connectives. Hence, an Equality Graph $G^{\mathrm{E}}(\varphi^{\mathrm{E}})$ represent all formulas that have the same predicate sets as φ^{E}.

Example 2.2 Figure 1 shows an Equality Graph $G^{\mathrm{E}}(\varphi^{\mathrm{E}})$ for some equality formula φ^{E} for which $E_= : \{(x_1 = x_5), (x_5 = x_6), (x_6 = x_2), (x_3 = x_7), (x_7 = x_8), (x_8 = x_4)\}$ and $E_\neq : \{(x_1 \neq x_2), (x_2 \neq x_3), (x_3 \neq x_4), (x_3 \neq x_8)\}$. An assignment TRUE to an edge (regardless whether it is an $E_=$ or $E_\neq$ edge), means that the *equality* is satisfied. Hence, to satisfy an $E_\neq$ edge an assignment FALSE is required.

Transitivity of equality can be enforced for every three variables in φ^{E}:

Definition 2.3 [Transitivity Constraint] For variables x_i, x_j, x_k, the constraint

$$e_{i,j} \wedge e_{j,k} \rightarrow e_{i,k}$$

is called a *transitivity constraint*.

Such constraints can be added to $\mathcal{T}$ for every three variables in φ (in fact, this was one of the methods suggested by Bryant and Velev in [2]), although typically it is possible to find efficiently a small subset of them that is still sufficient for the reduction, as shown in [2,6] and in this article.

Definition 2.4 [Equality Path] An *Equality Path* in an Equality Graph G^{E} is a path made of $E_{=}$ (dashed) edges. Denote by $x =^* y$ the fact that x has an Equality Path to y in G^{E}, where $x, y \in V$.

Definition 2.5 [Disequality Path] A *Disequality Path* in an Equality Graph G^{E} is a path made of $E_{=}$ (dashed) edges and a single $E_{\neq}$ (solid) edge. Denote by $x \neq^* y$ the fact that x has a Disequality Path to y in G^{E}, where $x, y \in V$.

Equality and Disequality paths are called *simple* if no vertex in the path is repeated. In Figure 1 it holds, for example, that $x_2 =^* x_5$ due to the simple path x_2, x_6, x_5; $x_2 \neq^* x_5$ due to the simple path x_2, x_1, x_5; and $x_5 \neq^* x_7$ due to the simple path x_5, x_6, x_2, x_3, x_7.

Intuitively, an Equality Path $x_i =^* x_j$ in G^{E} implies that x_i and y_i are possibly required to be equal in order to satisfy the formula from which G^{E} was built. A Disequality Path $x_i \neq^* x_j$ implies the opposite: x_i and x_j are possibly required to be different in order to satisfy this formula. More formally, if $x_i =^* x_j$ in some equality graph G_1^{E}, then there exists a satisfiable equality formula φ^{E} such that $G^{\mathrm{E}}(\varphi^{\mathrm{E}}) \equiv G_1^{\mathrm{E}}$ and in every satisfying assignment to φ^{E}, $x_i = x_j$. The formal description for $x_i \neq^* x_j$ is similar.

Definition 2.6 [Contradictory Cycle] A Contradictory Cycle in an Equality Graph is a cycle with exactly one disequality (solid) edge.

Several characteristics of contradictory cycles are:

(i) For every pair of nodes x, y in a Contradictory Cycle, it holds that $x =^* y$ and $x \neq^* y$.

(ii) For every Contradictory Cycle C, either C is *simple* or a subset of its edges forms a Simple Contradictory Cycle. It is sufficient, therefore, to refer only to simple contradictory cycles.

(iii) It is impossible to satisfy simultaneously all the predicates that correspond to edges of a Contradictory Cycle. Further, this is the only type of subgraph with this property.

Let the *positive set* S of α be the positive literals in ϕ assigned TRUE and the negative literals in ϕ assigned FALSE. The polarity information (whether the edge represents an equality or disequality) in the equality graph is useful due to the following property of NNF formulas.

Theorem 2.7 (Monotonicity of NNF) *Let ϕ be an NNF formula and α be an assignment such that $\alpha \models \phi$. Every assignment α' with a positive set S' such that $S \subseteq S'$ satisfies ϕ as well.*

The same theorem was used, for example, in [7].

Two graph-theoretical concepts that are used by our algorithm are:

Definition 2.8 [Chord] A *chord* in a cycle is an edge between two non-adjacent vertices.

Definition 2.9 [Chordal graphs] A graph is called *chordal* if no cycle of size four or more in the graph is chord-free.

Every graph can be made chordal in polynomial time by adding edges. The following procedure returns a set of chords sufficient for making a graph $G(V, E)$ chordal:

(i) While $V \neq \emptyset$:
  (a) Choose a vertex $v \in V$;
  (b) Add to E an edge between every two neighbors of v (if it was not already in E).
  (c) Remove v and its incident edges.

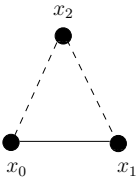
(ii) Return the set of edges that were added in line b.

The order by which vertices are chosen in line a affects the number of added chords. A simple greedy criterion is to choose the vertex that adds the least number of edges (finding the smallest set of edges that make a graph chordal is NP-hard).

Chordal graphs were used by [2] in the context of their observation that transitivity should be enforced only on chord-free cycles (which, in a chordal graph, are only the triangles). In [6], as well as in this paper, it is used in a different context, but with a similar purpose: it enables the algorithm to constrain only triangles. More specifically, although a graph (V, E) can contain an exponential number of contradictory cycles in $|V|$, it can only contain a polynomial number of triangles. Yet, enforcing transitivity on triangles (whether they are contradictory or not) is sufficient for enforcing it on all contradictory cycles, as shown in [6] and in Section 4.

3 Main Theorem

The key idea that is formulated by Theorem 3.4 below and later exploited by our algorithm RTC^S, can first be demonstrated by a simple example.



	α	α'
$e_{0,1}$	TRUE	TRUE
$e_{1,2}$	TRUE	TRUE
$e_{0,2}$	FALSE	TRUE

Fig. 2. An equality graph for Example 3.1, demonstrating how an assignment that contradicts transitivity, can be changed to one that respects transitivity.

Example 3.1 For the Equality Graph in Figure 2(left), the single transitivity constraint $\mathcal{T} = (e_{0,2} \wedge e_{1,2} \rightarrow e_{0,1})$ is sufficient.
To justify this claim, it is sufficient to show that for every assignment α that satisfies $\mathcal{B} \wedge \mathcal{T}$, there exists an assignment α' that satisfies $\mathcal{B}$ and transitivity of equality. Since this, in turn, implies that φ^{E} is satisfiable as well, then it is implied that φ^{E} is equisatisfiable to $\mathcal{B} \wedge \mathcal{T}$.

It is possible to construct such an assignment α' because of the monotonicity of NNF (recall that the polarity of the edges in the Equality Graph are according to their polarity in the NNF representation of φ^{E}). There are only two satisfying assignments to $\mathcal{T}$ that do not satisfy transitivity. One of these assignments is shown in the α column in the table to the right of the drawing. The second column shows a corresponding assignment α', which clearly satisfies transitivity. It is left to prove that every formula $\mathcal{B}$ that corresponds to the above graph, is still satisfied by α' if it was satisfied by α. For example, for $\mathcal{B} = (\neg e_{0,1} \vee e_{1,2} \vee e_{0,2})$, both $\alpha \models \mathcal{B} \wedge \mathcal{T}$ and $\alpha' \models \mathcal{B}$ and respects transitivity. Intuitively, this is guaranteed to be true because α' is derived from α by flipping an assignment of a positive (un-negated) predicate ($e_{0,2}$) from FALSE to TRUE. Similarly, we can flip an assignment to a negated predicate ($e_{0,1}$ in this case) from TRUE to FALSE.

A formalization of this argument requires a reference to the monotonicity of NNF (Theorem 2.7): Let S and S' denote the positive sets of α and α' respectively. Then in this case $S = \{e_{1,2}\}$ and $S' = \{e_{1,2}, e_{0,2}\}$. Thus $S \subset S'$ and hence, according to Theorem 2.7, $\alpha \models \mathcal{B} \rightarrow \alpha' \models \mathcal{B}$.

Several definitions are needed in order to generalize this example into a theorem.

Definition 3.2 [A constrained Contradictory Cycle] Let $C = (e_s, e_1, \ldots, e_n)$ be a Contradictory Cycle where e_s is the solid edge. Let ψ be a formula over the Boolean variables in $\mathcal{B}$ that encodes the edges of C. C is said to be *constrained in* ψ if the assignment $(e_s, e_1, \ldots, e_n) \leftarrow (F, T, \ldots, T)$ contradicts ψ.

Definition 3.3 [A Reduced Transitivity Constraints formula $\mathcal{T}$] A Reduced Transitivity Constraints (RTC) formula $\mathcal{T}$ for an equality graph G^{E} is a conjunction of transitivity constraints that constrains all the simple contradictory cycles in G^{E} [2].

Consider, for example, an Equality Graph in which all edges are solid (disequalities): in such a graph there are no contradictory cycles and hence no constraints are required: $\mathcal{T} = \text{TRUE}$.

Theorem 3.4 (Main) *Let φ^E be an equality formula, and let $\mathcal{T}$ be an RTC formula for $G^E(\varphi^E)$. Then φ^E is satisfiable if and only if $\mathcal{B} \wedge \mathcal{T}$ is satisfiable.*

The proof of this theorem appears in [6] and [5]. Since $\mathcal{T}$ is a conjunction of transitivity constraints, the proof of the 'only if' direction ($\Rightarrow$) is trivial. To prove the other direction it is shown in [5] that there exists an algorithm for reconstructing an assignment that satisfies all transitivity constraints from a given assignment α that only satisfies $\mathcal{T}$.

Given Theorem 3.4, it is left to show an algorithm that generates a formula that constrains all simple contradictory cycles. In [6] we presented the RTC algorithm for this purpose, parts of which are re-used here in the description of the new algorithm RTC^S. The latter only constrains *simple* contradictory cycles, as it should according to Theorem 3.4, hence the superscript S. It is also simpler to describe and implement than RTC.

4 The RTC^S algorithm

The RTC^S algorithm processes *Biconnected Components* (BCC) [3] in the given Equality Graph.

Definition 4.1 [Maximal Biconnected Component] A *Biconnected Component* of an undirected graph is a maximal set of edges such that any two edges in the set lie on a common simple cycle.

It is sufficient to focus on BCCs because only cycles need to be constrained (more specifically, contradictory cycles). Each considered BCC contains a solid edge e_s and all the contradictory cycles that it is part of. In line 4, RTC^S makes the BCC chordal, by adding edges. After the graph is chordal RTC^S calls GENERATE-CONSTRAINTS^S, which strengthens $\mathcal{T}$ with all the transitivity constraints that are necessary for constraining all the contradictory cycles in this BCC with respect to e_s.

[2] The definition of this term in [6] includes an additional requirement, that it is not more restrictive than the constraint generated by Bryant and Velev's Sparse method technique. This restriction is not necessary in our context.

Algorithm 1 RTC^S returns a formula $\mathcal{T}$, which conjoins all the transitivity constraints that are sufficient and necessary in order to constrain all simple contradictory cycles in a given equality graph.

RTC^S (Equality Graph $G^{\mathrm{E}}(V, E_{=}, E_{\neq})$)
1: $\mathcal{T}$ = TRUE
2: **for all** $e_s \in E_{\neq}$ **do**
3: Find $B(e_s)$, the maximal BCC in G^{E} that is made of e_s and $E_{=}$ edges;
4: Make $B(e_s)$ chordal; ▷ This step adds new dashed edges.
5: GENERATE-CONSTRAINTS^S $(B(e_s))$; ▷ see Algorithm 2.
6: **end for**
7: return $\mathcal{T}$;

A possible optimization to RTC^S is to reuse chords: denote by E_p the union of chords that were added in previous iterations of the algorithm (when other BCCs were considered), and $G^{\mathrm{E}}_{\neq}$ edges. The greedy criterion by which vertices are chosen (see the algorithm for making graphs chordal after Definition 2.9) should be changed as follows: rather than counting the number of added edges, count only those edges that are added and are also not in E_p (since E_p edges are already represented in the resulting formula). This optimization reduces the number of added chords and, consequently, the number of variables and constraints.

4.1 Deriving transitivity constraints in P time

Let B be a chordal biconnected component in which there is a single solid edge e_s adjacent to vertices x_s,x'_s. The algorithm in Fig. 2 finds the necessary and sufficient constraints for constraining all the simple contradictory cycles with respect to e_s. We will use a convention by which removing a vertex implies removing its incident edges. Also, we will use set notation for graph elements when the meaning is clear from the context, for example:

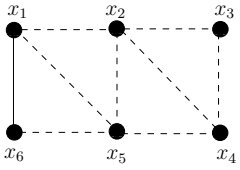
- $(x_i, x_j) \in B$ means that the graph B has an edge (x_i, x_j),
- $B' \subseteq B$ means that B' is a subgraph of B,
- $B \setminus v$ is the graph B after the removal of the vertex v and its incident edges from B.

Two comments about GENERATE-CONSTRAINTS^S:

- An optimization for line 5 is to add the constraint only if it was not added before. Our implementation in fact maintains the constraints as a set, and generates $\mathcal{T}$ only in the end.
- The condition in line 4.1 can be checked in polynomial time, by, for example,

Algorithm 2 GENERATE-CONSTRAINTS^S adds transitivity constraints to a (global) formula $\mathcal{T}$, that are sufficient and necessary for constraining all the simple contradictory cycles in a given bi-connected component with a single solid edge $e_s = (x_s, x'_s)$.

1: **procedure** GENERATE-CONSTRAINTS^S(Chordal BCC $B(V, E)$)
2: **for** each vertex $v \in \{V \setminus \{x_s, x'_s\}\}$ **do**
3: Let $B' = B \setminus v$.
4: **for** every (x_i, x_j) that
4.1 is on a simple cycle with e_s in B' (or $e_s \equiv (x_i, x_j)$), and
4.2 $\{(v, x_i), (v, x_j)\} \in B$
do
5: $\mathcal{T} = \mathcal{T} \wedge (e_{v,x_i} \wedge e_{v,x_j} \rightarrow e_{x_i,x_j})$
6: **end for**
7: **end for**
8: **end procedure**



Examining...	Added constraints
x_2	$e_{1,2} \wedge e_{2,5} \rightarrow e_{1,5}$
x_3	$e_{2,3} \wedge e_{3,4} \rightarrow e_{2,4}$
x_4	$e_{2,4} \wedge e_{4,5} \rightarrow e_{2,5}$
x_5	$e_{1,5} \wedge e_{5,6} \rightarrow e_{1,6}$

Fig. 3. An Equality Graph for Example 4.2.

building a maximal BCC B'' around e_s in B'. Every edge in B'' is on a simple cycle with e_s in B'.

Example 4.2 Consider the Equality Graph in Figure 3. Assume that the vertices are examined in line 2 in an order corresponding to the variable index. For this graph, x_1, x_6 are those vertices called x_s, x'_s in GENERATE-CONSTRAINTS^S. The first vertex examined in line 2 is therefore x_2. The edge (x_1, x_5) is the only one fulfilling the two conditions: $(x_2, x_1), (x_2, x_5)$ are edges in B, and it is on a simple cycle with e_s in B'. Indeed, (x_1, x_5) is an edge in $B'' = (x_1, x_6, x_5)$, the maximal BCC that contains e_s after the removal of x_2 and its incident edges. Therefore the constraint $e_{1,2} \wedge e_{2,5} \rightarrow e_{1,5}$ is the only one added in this iteration. The table below shows the constraints added in each iteration.

Theorem 4.3 *For a chordal BCC B with a single solid edge e_s, the constraints added by* GENERATE-CONSTRAINTS$^S(B)$ *are sufficient and necessary for*

constraining all simple contradictory cycles in B.

Proof. (Sufficiency) We first prove the following:

Lemma 4.4 *If for every triangle (x_i, v, x_j) in B such that $(x_i, v), (v, x_j) \neq e_s$ and $(x_i, v), (v, x_j)$ is part of a simple contradictory cycle, the transitivity constraint $e_{i,v} \wedge e_{v,j} \rightarrow e_{i,j}$ is in $\mathcal{T}$, then $\mathcal{T}$ constrain all the simple contradictory cycles in B.*

Proof. Let C be a simple contradictory cycle in B. By induction on the size of C:

Base: Let C be a triangle (x_i, v, x_j), where (x_i, x_j) is the solid edge e_s. Since $(x_i, v), (v, x_j)$ are part of a contradictory cycle, the constraint $e_{i,v} \wedge e_{v,j} \rightarrow e_{i,j}$ is in $\mathcal{T}$. Thus, C is constrained in C.

Step: Assume the Proposition holds for C of size n ($n \geq 3$), and consider C with size $n+1$. Since C is chordal, it can be decomposed into a triangle, say x_i, v, x_j, and another contradictory cycle $C' = C \setminus \{(x_i, v), (v, x_j)\} \cup (x_i, x_j)$ (this observation is proven as Proposition 2 in [5]). By the induction hypothesis, C' is constrained by $\mathcal{T}$ (since $|C'| = n$). The constraint $e_{i,v} \wedge e_{v,j} \rightarrow e_{i,j}$ is in $\mathcal{T}$ because $(x_i, v), (v, x_j)$ is part of C, which, recall, is a simple contradictory cycle. Now, assume that C is not constrained by $\mathcal{T}$, i.e. an assignment α that assigns TRUE to all dashed edges and FALSE to e_s still satisfies $\mathcal{T}$. In this assignment $e_{i,v}, e_{v,j}$ are assigned TRUE, but then $e_{i,j}$ is assigned TRUE as well due to the constraint mentioned above. Hence, in C' all dashed edges are assigned TRUE whereas e_s is assigned FALSE, which contradicts the induction hypothesis. □

It is left to show that the constraints added by RTC^S satisfy the premise of Lemma 4.4, i.e., that it adds a constraint $e_{i,v} \wedge e_{v,j} \rightarrow e_{i,j}$ for every triangle (x_i, v, x_j) in B such that $(x_i, v), (v, x_j) \neq e_s$ and $(x_i, v), (v, x_j)$ are part of a simple contradictory cycle.

Lemma 4.5 *Let (x_i, v, x_j) be a dashed triangle in B such that there exists a simple contradictory cycle through $(x_i, v), (v, x_j)$. Then there exists a simple contradictory cycle in B through (x_i, x_j) that does not go through v.*

Proof. Let C be a simple contradictory cycle that goes through $(x_i, v), (v, x_j)$, and let $C' = C \setminus \{(x_i, v), (v, x_j)\} \cup (x_i, x_j)$. Observe that in a simple cycle C, the degree of each vertex (counting only C edges) is 2. It is easy to see that the degree of each vertex in C' is the same as in C, other than v for which the degree is reduced from 2 to 0 (which means that is not part of C'). Hence C' is simple and does not go through v. □

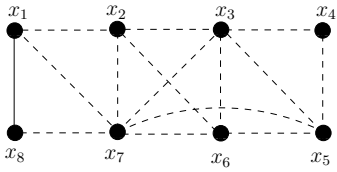


Fig. 4. An Equality Graph that demonstrates the difference between RTC and RTC^S.

The contra-positive conclusion from Lemma 4.5 is that if (x_i, x_j) is not part of a simple contradictory cycle with e_s in B', then $(x_i, v), (v, x_j)$ is also not in a simple contradictory cycle with e_s. But since this is the only case in which GENERATE-CONSTRAINTS^S does *not* add a constraint, we conclude that it adds the constraints as required by the premise of Lemma 4.4, i.e. it adds a constraint $e_{i,v} \wedge e_{v,j} \rightarrow e_{i,j}$ for every triangle (x_i, v, x_j) in B such that $(x_i, v), (v, x_j) \neq e_s$ and $(x_i, v), (v, x_j)$ are part of a simple contradictory cycle. Hence, by Lemma 4.4, $\mathcal{T}$ constrains all simple contradictory cycles in B.

(Necessity) Falsely assume that there is a redundant constraint, e.g. there exists a constraint $e_{x_i,v} \wedge e_{v,x_j} \rightarrow e_{x_i,x_j}$ although $(x_i, v), (v, x_j)$ is not part of a simple contradictory cycle or that $(x_i, x_j) \notin B$. If this constraint is added (in line 5), it means that (x_i, x_j) is part of a simple contradictory cycle C' with e_s not through v. But this means that $C = C' \setminus (x_i, x_j) \cup \{(x_i, v), (v, x_j)\}$ must be simple as well (it adds a vertex v with degree 2, and does not change the degree of the other vertices), which contradicts the assumption.

□

4.2 The differences between RTC *and* RTC^S

As was mentioned earlier, the original GENERATE-CONSTRAINTS procedure that appeared in [6] added enough transitivity constraints to constrain all contradictory cycles, and not just the simple ones as required by Theorem 3.4. The graph in Figure 4 demonstrates the difference between the results of the two algorithms. Consider the constraints that are added when removing x_7 in line 2 of RTC^S: the edge (x_1, x_8) is e_s itself and hence the constraint $e_{1,7} \wedge e_{7,8} \rightarrow e_{1,8}$ is added. No other edge fulfills the condition in line 4.1. GENERATE-CONSTRAINTS, on the other hand, adds, for example, also the constraint $e_{6,7} \wedge e_{5,7} \rightarrow e_{5,6}$, because of the non-simple cycle $(x_1, x_2, x_3, x_5, x_7, x_6, x_7, x_8, x_1)$. All together Algorithm GENERATE-CONSTRAINTS^S generates 16 constraints for this graph, whereas GENERATE-CONSTRAINTS generates 26 constraints.

Algorithm GENERATE-CONSTRAINTS of [6] traverses the BCC, each time expanding the contradictory cycle while adding transitivity constraints. It starts from each triangle that one of its edges is e_s. From there it gradually increases the cycle it examines (at each step it replaces an edge with two edges that lean

% dashed	Run time		# Constraints		Constraints
	RTC	RTC^S	RTC	RTC^S	ratio
10	3.37	0.4608	208.7	153.5	0.73
30	195.81	120.15	170308.7	117832.9	0.69
50	339.29	300.23	299937.8	240075.7	0.8
70	419.94	236.83	355631.3	328359.1	0.92
Average	239.6	164.4	206521.6	171605.3	0.83

Table 1
Experimental results of RTC^S vs. RTC on randomly generated graphs. Each line in the table corresponds to an average on (the same) 10 graphs, but with a varying percentage of dashed vs. solid edges.

on that edge) while adding constraints. To avoid traversing an exponential number of paths, it uses a cache of constraints, and stops traversing the graph in a direction that results in a constraint that already appears in the cache. This, in turn, requires that all cycles are traversed, rather than only the simple ones. In [5] we also showed an algorithm that does not use the cache, and hence generates the requested formula (i.e. not overly constrained), but the algorithm there is worst-case exponential.

5 Experiments and conclusions

We re-ran the experiments with random equality graphs, which were first presented in [6]. Table 1 presents the comparison. Each line in the table corresponds to the same 10 randomly generated topologies, with 1% of 'double edges' (edges that are both solid and dashed), the % of dashed edges as specified in the first column, and the rest are solid. The number of vertices is 200, and the number of edges is 800.

Overall, there is a decrease of 17% in the number of transitivity constraints, and 32% decrease in the run time of the algorithm (not including SAT time). For comparison, on the same graphs the sparse method [2], which generates three constraints for each triangle in the graph regardless of their polarity, generates 390165 constraints on average (regardless of the ratio between solid and dashed edges). For most of these graphs the formula could not be generated by the exponential method of [5].

As expected, RTC^S is better in practice than RTC, both in terms of the size of the generated formula, and the overall running time. The difference in the run time of SAT between the formulas generated by RTC and RTC^S was quite

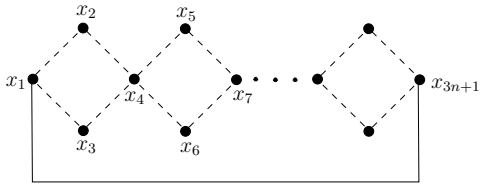


Fig. 5. An equality formula corresponding to the crafted examples.

negligible in these cases.

Is RTC^S *competitive with lazy-style solvers?* Although we did not check it systematically, we expect that on instances with many uninterpreted functions (as most of the benchmarks in the SMT-LIB [8] suite are), the answer is no. As was noted in [6], lazy solvers are likely to perform better in such cases, since the reduction to equality logic using Bryant's reduction creates graphs in which most of the edges are both dashed and solid, a case in which RTC^S has no advantage in comparison to the sparse method of Bryant and Velev [2] (which by itself, as far as we know, has never been compared experimentally to some of the modern implementations based on congruence closure, such as [4]). We also noticed that large graphs (with a 150 nodes or more) with a high degree of connectivity, as the ones created when reducing uninterpreted functions, frequently lead to an excessive run time in making them chordal, despite the polynomial upper-bound on the running time of this operation.

Are there cases in which RTC^S *has an advantage over lazy-style solvers?* To test this question we compared RTC^S to Yices (version 1.09), where the propositional formulas generated by RTC^S were solved with Yices as well. The comparison was done on crafted examples without uninterpreted functions. It turns out that generating random CNF-s is rather meaningless in this context, because it is very unlikely that in such formulas every assignment that satisfies the skeleton will correspond to a contradictory cycle. Indeed, in all the experiments we made with random CNFs, the formulas were satisfiable and very easy to solve by both methods. We therefore crafted a set of formulas whose respective equality graph follow the pattern that appears in Figure 5. The number of 'diamonds' is denoted by n, hence the right most node is x_{3n+1}. The checked formula is

$$
\begin{aligned}
& x_1 \neq x_{3n+1} \wedge \\
& ((x_1 = x_2 \wedge x_2 = x_4) \vee (x_1 = x_3 \wedge x_3 = x_4)) \wedge \\
& \qquad\qquad \vdots \\
& ((x_{3n-2} = x_{3n-1} \wedge x_{3n-1} = x_{3n+1}) \vee (x_{3n-2} = x_{3n} \wedge x_{3n} = x_{3n+1})) \ ,
\end{aligned} \tag{1}
$$

which is unsatisfiable for all $n > 0$. There are $4n+1$ edges and hence Boolean

variables in the formula's skeleton. Satisfying either the top or bottom path of each diamond (or both), together with the disequality $x_1 \neq x_{3n+1}$, satisfies the Boolean skeleton of the formula. Yet each such satisfying assignment corresponds to a contradictory cycle, which makes the formula unsatisfiable.

This type of formula is expected to be hard for lazy-style solvers, because there is an exponential number of solutions that satisfy the skeleton, none of which is a real solution (it seems that theory propagation and learning cannot be effective in this case either). The results, in seconds, appear in the table below. TO denotes a timeout of 1 hour. It is clear that in such formulas indeed RTC^S has an advantage.

n	Yices	RTC^S
20	95.7	< 1
25	3210.2	< 1
30	TO	< 1
40	TO	< 1

To summarize, as indicated in the introduction, RTC^S dominates the two previously published alternatives: RTC^S is polynomial in contrast to the exponential algorithm described in [5], and it generates formulas that are guaranteed to be smaller and less constrained than the formulas generated by the polynomial approximation offered by RTC (or, if it happens to be that there are no non-simple cycles, it generates an equivalent formula). It is also simpler to implement and (subjectively) more elegant than RTC.

References

[1] Ackermann, W., "Solvable cases of the Decision Problem," Studies in Logic and the Foundations of Mathematics, North-Holland, Amsterdam, 1954.

[2] Bryant, R. and M. Velev, *Boolean satisfiability with transitivity constraints*, in: *Proc.* 12^{th} *Intl. Conference on Computer Aided Verification (CAV'00)*, Lect. Notes in Comp. Sci. **1855**, 2000.

[3] Cormen, T., C. Leiserson and R. Rivest, "Introduction to Algorithms," MIT press, 2000 p. 563.

[4] Ganzinger, H., G. Hagen, R. Nieuwenhuis, A. Oliveras and C. Tinelli, *DPLL(T): Fast decision procedures*, in: *Proc.* 16^{th} *Intl. Conference on Computer Aided Verification (CAV'04)*, number 3114 in Lect. Notes in Comp. Sci. (2004), pp. 175–188.

[5] Meir, O., "A decision procedure for equality logic," Master's thesis, Technion (2005), URL: ie.technion.ac.il/∼ofers/publications/theses/orly-meir.pdf.gz.

[6] Meir, O. and O. Strichman, *Yet another decision procedure for equality logic*, in: K. Etessami and S. Rajamani, editors, *Proc.* 17^{th} *Intl. Conference on Computer Aided Verification (CAV'05)*, Lect. Notes in Comp. Sci. **3576** (2005), pp. 307–320.

[7] Pnueli, A., Y. Rodeh, O. Strichman and M. Siegel, *The small model property: How small can it be?*, Information and computation **178** (2002), pp. 279–293.

[8] Ranise, S. and C. Tinelli, *The Satisfiability Modulo Theories Library (SMT-LIB)*, `www.SMT-LIB.org` (2007).