Semantic Foundations of Equality Saturation

Dan Suciu
University of Washington, Seattle, WA, USA

Yisu Remy Wang
University of California Los Angeles, CA, USA

Yihong Zhang
University of Washington, Seattle, WA, USA

Abstract

Equality saturation is an emerging technique for program and query optimization developed in the programming language community. It performs term rewriting over an E-graph, a data structure that compactly represents a program space. Despite its popularity, the theory of equality saturation lags behind the practice. In this paper, we define a fixpoint semantics of equality saturation based on tree automata and uncover deep connections between equality saturation and the chase. We characterize the class of chase sequences that correspond to equality saturation. We study the complexities of terminations of equality saturation in three cases: single-instance, all-term-instance, and all-E-graph-instance. Finally, we define a syntactic criterion based on acyclicity that implies equality saturation termination.

2012 ACM Subject Classification Theory of computation → Equational logic and rewriting; Theory of computation → Rewrite systems

Keywords and phrases the chase, equality saturation, term rewriting, tree automata, query optimization

Digital Object Identifier 10.4230/LIPIcs.ICDT.2025.11

Related Version *Full Version*: `https://arxiv.org/abs/2501.02413` [27]

Funding This work is supported by NSF IIS 2314527 and NSF SHF 2312195, and is conducted while some of the authors participated in the Simons Program on Logic and Algorithms in Databases and AI.

1 Introduction

Given a set of identities between terms, the word problem asks whether the identities imply two ground terms t_1, t_2 are equivalent, i.e. $t_1 \approx t_2$. This fundamental problem has applications including automated theorem proving, program verification, and query equivalence checking. In his Ph.D. thesis, Nelson [22] introduced a data structured called *E-graph* for efficiently answering the word problem. At the core, an E-graph is a compact representation of an equivalence relation over a possibly infinite set of ground terms. During the 2000s, researchers applied E-graphs to program optimization [14, 28]. The compiler populates an E-graph with many equivalent programs, using axiomatic rewrites, then extracts the best program from the equivalent ones. In particular, Tate et.al. [28] coined the term *equality saturation (EqSat)* and gave a procedural description of the algorithm. In 2021, Willsey et al. [32] proposed `egg`, which introduced important algorithmic improvements and made EqSat practical. Since 2021, EqSat has been applied to a wide range of topics in domain-specific program optimization, including floating-point computation [24] computational fabrication [20], machine learning systems [33], and hardware design [29, 4]. There is also a growing interest in using EqSat for query optimization in data management. For example, EqSat is used to optimize queries in OLAP [6], linear algebra [30], tensor algebra [25], and Datalog [31].

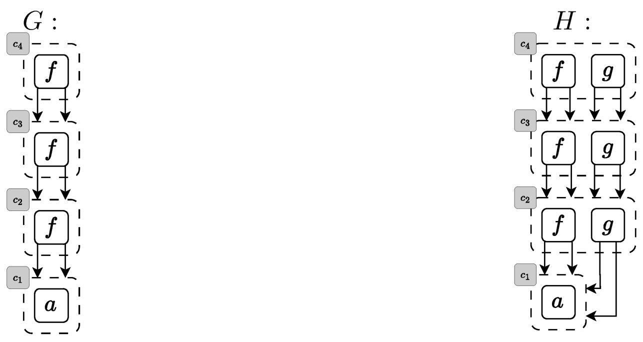


Figure 1 Two E-graphs G, H, before and after EqSat.

The equality saturation procedure consists of repeatedly selecting an identity $u \approx v$ from the given set, matching the term u with the E-graph, then adding the term v to the E-graph, if it wasn't already there. Equality saturation terminates when no new terms can be added. There are striking connections between equality saturation and database concepts. Zhang et al. [35] observed that the *matching* step is the same as conjunctive query evaluation, and described significant speedups in `egg` by using a Worst Case Optimal Join algorithm [23] for matching. A recent system, `egglog` [34], unified EqSat and Datalog to improve `egg`'s support for program optimization and program analysis.

In this paper we study another deep connection between equality saturation and the chase procedure for Tuple Generating Dependencies (TGDs) and Equality Generating Dependencies (EGDs) [8]. Our hope is that these results will help solve some of the open problems in equality saturation by using techniques and results for the chase procedure. Before describing our results we give a gentle introduction to EqSat and describe some of its open problems.

▶ **Example 1.** *Consider a simple language with two binary operators f, g and constant a. We want to optimize the following term t (the "8th power" of f on a):*

$$t = f(f(f(a,a), f(a,a)), f(f(a,a), f(a,a))) \tag{1}$$

We are given a single identity, $f(x,x) \approx g(x,x)$, which says two terms $f(t_1,t_2)$ and $g(t_1,t_2)$ are equivalent, provided that t_1, t_2 are equivalent. Starting with the initial term t, EqSat constructs an E-graph G and grows it to represent all terms equivalent to t. The literature defines an E-graph as a set of E-classes, *where each E-class is a set of* E-nodes, *and each E-node is labeled with a function symbol and has a number of E-class children equal to the arity of the symbol. EqSat starts by constructing an E-graph G representing t, shown on the left in Figure 1. There are 4 E-classes, each consisting of one single E-node; the E-class c_4 represents precisely the term t. Next, EqSat repeatedly applies the identity $f(x,x) \approx g(x,x)$, by matching the left-hand side $f(x,x)$ to the E-graph, then adding the right-hand side $g(x,x)$ to the E-graph: we formalize this in Sec. 3. The resulting E-graph H is on right of Figure 1. There are 4 E-classes, $c_1, \dots, c_4$, each consisting of 1 or 2 E-nodes. For example, c_4 has two E-nodes, and represents two equivalent terms, $f(t_1,t_2) \approx g(t_1,t_2)$, where t_1, t_2 are any terms represented by c_3. By continuing this reasoning, we observe that c_4, represents a total of 2^7 possible terms, namely all terms of the form: $h(h(h(a,a), h(a,a)), h(h(a,a), h(a,a)))$ where each h can be either f or g.*

Open problems about EqSat. We still understand very little about equality saturation. Most descriptions of EqSat focus on an imperative understanding[1] of equality saturation and E-graphs. E-graphs are described by their individual components (e.g., a hashconsing data structure, a union-find, etc.), and EqSat is commonly defined in pseudocode as a sequence of operations. In other words, the semantics of EqSat is the output of the specific algorithm, if it terminates; if the algorithm diverges, the semantics is undefined.

We also do not know much about when EqSat terminates. The termination problem asks, given a set of rewrite rules, whether EqSat terminates on a given input E-graph, or whether it terminates on all input E-graphs. This is a fundamental problem of EqSat and has applications in program/query optimization and equivalence checking: If EqSat terminates on the symmetric closure of a set of (variable-preserving) rewrite rules $\mathcal{R}$, it decides the word problem of the equational theory defined by $\mathcal{R}$ (Lemma 22). With an appropriate cost model, EqSat can further pick the optimal program among all programs equivalent to the input, e.g. by using Knuth's algorithm [15].

Our contribution. After a review of some background material in Sec. 2, we introduce E-graphs and EqSat in Sec. 3. Our definition, in Sec. 3.1, applies even to the cases when equality saturation does not terminate and, for that purpose, we define the E-graph to be a reachable, deterministic tree automaton, with possibly infinitely many states. By explicitly allowing infinite E-graphs we can define a formal semantics even when EqSat does not terminate. We show that concepts in tree automata are in 1-1 correspondence with those in E-graphs: the automaton states correspond to E-classes, and the transitions correspond to E-nodes. A term is represented by an E-graph iff it is accepted by the E-graph viewed as a standard tree automaton. We prove that, for any two E-graphs there exists at most one homomorphism between them, and, therefore, E-graphs are *rigid* tree automata. Next, in Sec. 3.2, we define a few basic operations on E-graphs, such as E-matching, insertion, congruence closure, and least upper bounds, by relying on tree-automata concepts. Using these operations, we define in Sec. 3.3 an *immediate consequence operator (ICO)*, and define EqSat formally as the least fixpoint of the ICO. The least fixpoint always exists and is unique, even if the fixpoint procedure does not terminate, in which case the least fixpoint may be infinite. Finally, we prove an important lemma, called the *Finite Convergence Lemma*, stating that, if the least fixpoint is finite, then equality saturation procedure converges in finitely many steps. This is not immediately obvious because, while E-matching and insertion strictly increase the size of the E-graph, congruence closure may decrease it. A similar proposition fails for TGDs and EGDs: there exists an infinite chase where all instances have bounded size, hence its "limit" is finite.

Next, in Sec. 4 we describe the connection between EqSat and the chase. After a brief review of the chase in Sec. 4.1, we start by presenting a reduction from the Skolem chase to equality saturation, denoted $\textsc{SklCh} \Rightarrow \textsc{EqSat}$ (Sec. 4.2), then from equality saturation to the standard chase, denoted $\textsc{EqSat} \Rightarrow \textsc{StdCh}$ (Sec. 4.3). For $\textsc{SklCh} \Rightarrow \textsc{EqSat}$, given a set of dependencies, we show there exists a set of rewrite rules where EqSat produces an encoded result of the Skolem chase and has the same termination behavior. For $\textsc{EqSat} \Rightarrow \textsc{StdCh}$, we show that, given a set of rewrite rules, there exists a set of dependencies where the standard chase produces an encoded result of EqSat (whether it terminates). Since the standard chase

[1] A notable exception is `egglog` [34], whose semantics is based on fixpoints instead of implementation details. Some early works also define E-graphs (under a different name like abstract congruence closure) as tree automata similar to ours [26, 2, 12].

is a non-deterministic process, we characterize the type of chase sequences that terminate when EqSat terminates (Theorem 30). We call them *EGD-fair* chase sequences. Intuitively, a chase sequence is called EGD-fair if it applies EGDs to a fixpoint frequently enough. We show in Theorem 30 that,

$$\begin{aligned} \text{EqSat terminates} \quad &\Leftrightarrow \text{one chase sequence terminates} \\ &\Leftrightarrow \text{all EGD-fair chase sequences terminate.} \end{aligned}$$

The notion of EGD-fair chase sequences is of independent interest.

Finally, we present our main decidability results for EqSat in Sec. 5: we show that the single-instance termination problem of EqSat, denoted as $\mathcal{T}_G^{\text{EqSat}}$, is R.E.-complete, and the all-term-instance termination problem of EqSat, denoted as $\mathcal{T}_{\forall t}^{\text{EqSat}}$, is Π_2-complete. Our proof is based on a non-trivial reduction from the Turing machine, first presented in the undecidability proof of the finiteness of congruence classes defined by string rewriting systems [21]. While the single-instance case easily follows from the undecidability of Skolem chase termination, our approach allows us to also prove the Π_2-completeness of the all-term-instance termination case by a reduction from the universal halting problem. We also show the all-E-graph-instance termination problem of EqSat, denoted as $\mathcal{T}_{\forall G}^{\text{EqSat}}$, is undecidable, although the exact upper bound is open.

We contrast the termination problems of EqSat with those of the Skolem chase and the standard chase. The single-instance termination problems are R.E.-complete in all three cases [18, 5], and the all-instance termination of the Skolem chase ($\mathcal{T}_{\forall}^{\text{SklCh}}$) is R.E.-complete as well [18, 10]. This shows that $\mathcal{T}_{\forall}^{\text{SklCh}}$ is easier than $\mathcal{T}_{\forall t}^{\text{EqSat}}$. The case for the standard chase is more interesting. There are two all-instance termination problems of the standard chase: for all database instances, whether all chase sequences terminate in finitely many steps ($\mathcal{T}_{\forall,\forall}^{\text{StdCh}}$), and whether there exists at least one chase sequence that terminate ($\mathcal{T}_{\forall,\exists}^{\text{StdCh}}$). It has been shown $\mathcal{T}_{\forall,\exists}^{\text{StdCh}}$ is Π_2-complete [11], but the exact complexity of $\mathcal{T}_{\forall,\forall}^{\text{StdCh}}$ is open. Grahne and Onet showed if we allow one *denial constraint*, $\mathcal{T}_{\forall,\forall}^{\text{StdCh}}$ is Π_2-complete [11], although Gogacz and Marcinkowski [10] conjectured that this problem is indeed in R.E.

In Sec. 6 we propose a sufficient syntactic criterion that guarantees EqSat termination, called *weak term acyclicity*, which is based on the classic notion of *weak acyclicity* [8]. If a set of rewrite rules is weakly term acyclic, then EqSat terminates for all input E-graphs.

2 Background

2.1 Term Rewriting Systems

We review briefly the standard definition of a term rewriting system from [1]. A *signature* is a finite set Σ of function symbols with given arities. If V is a set of variables, then $T(\Sigma, V)$ denotes the set of terms constructed inductively using symbols from Σ and variables from V. Members of $T(\Sigma, V)$ are called *patterns*, and members of $T(\Sigma) \stackrel{\text{def}}{=} T(\Sigma, \emptyset)$ are called *ground terms*, or simply *terms* thereafter. A *substitution* is a function $\sigma : V \to T(\Sigma)$; if u is a pattern, then we denote by $u[\sigma]$ the term obtained by applying the substitution σ to u. A *rewrite rule* r has the form $lhs \to rhs$ where lhs and rhs are patterns and the variables in rhs are a subset of those lhs, $\mathit{Var}(rhs) \subseteq \mathit{Var}(lhs)$. A *term rewriting system* (TRS), $\mathcal{R}$, is a set of rewrite rules. $\mathcal{R}$ defines a *rewrite relation* $\to_{\mathcal{R}}$ as follows: $lhs[\sigma] \to_{\mathcal{R}} rhs[\sigma]$ for any substitution σ and rule $lhs \to rhs$ in $\mathcal{R}$, and, if $u \to_{\mathcal{R}} v$ then $f(w_1, \ldots, w_{i-1}, u, w_{i+1}, \ldots w_k) \to_{\mathcal{R}} f(w_1, \ldots, w_{i-1}, v, w_{i+1}, \ldots w_k)$ for any function symbol $f \in \Sigma$ of arity k, and any terms w_j, $j = 1, k; j \neq i$. Let $\to_{\mathcal{R}}^*$ be the reflexive and transitive closure of $\to_{\mathcal{R}}$. We define $(\leftarrow_{\mathcal{R}}) \stackrel{\text{def}}{=} (\to_{\mathcal{R}})^{-1}$, $(\leftrightarrow_{\mathcal{R}}) \stackrel{\text{def}}{=} (\to_{\mathcal{R}}) \cup (\leftarrow_{\mathcal{R}})$, and $(\approx_{\mathcal{R}}) \stackrel{\text{def}}{=} (\leftrightarrow_{\mathcal{R}}^*)$.

$\approx_{\mathcal{R}}$ is a congruence relation. We define the set of reachable terms $R^*(t) = \{t' \mid t \rightarrow^*_{\mathcal{R}} t'\}$. If a term rewriting system $\mathcal{R}$ is variable-preserving (i.e., $\mathit{Var}(\mathit{lhs}) = \mathit{Var}(\mathit{rhs})$ for all rules), we define $\mathcal{R}^{-1} = \{\mathit{rhs} \rightarrow \mathit{lhs} \mid (\mathit{lhs} \rightarrow \mathit{rhs}) \in \mathcal{R}\}$. It follows that $(\rightarrow_{(\mathcal{R}^{-1})}) = (\leftarrow_{\mathcal{R}})$.

2.2 Tree automata

Let Σ be a signature. A (bottom-up) tree automaton is a tuple $\mathcal{A} = \langle Q, \Sigma, Q_{\mathit{final}}, \Delta\rangle$, where Q is a (potentially infinite)[2] set of states, $Q_{\mathit{final}} \subseteq Q$ is a set of final states, and Δ is a set of transitions of the form $f(q_1, \ldots, q_n) \rightarrow q$ where $q, q_1, \ldots, q_n \in Q$, and $f \in \Sigma$. Denote by $\Sigma \cup Q$ the signature Σ extended with Q where each state is viewed as a symbol of arity 0. Then Δ is a term rewriting system for $\Sigma \cup Q$, and we will denote by $\rightarrow^*_{\mathcal{A}}$ (rather than $\rightarrow^*_{\Delta}$) the rewrite relation defined by Δ. A term $t \in T(\Sigma)$ is accepted by a state q if $t \rightarrow^*_{\mathcal{A}} q$, and we write $\mathcal{L}(q)$ for the set of terms accepted by q. The language accepted by $\mathcal{A}$ is $\mathcal{L}(\mathcal{A}) \overset{\text{def}}{=} \{t \in T(\Sigma) \mid \exists q_f \in Q_{\mathit{final}}, t \rightarrow^*_{\mathcal{A}} q_f\}$. A tree language $L \subseteq T(\Sigma)$ is called regular if it is accepted by some *finite* tree automaton.

Fix two tree automata $\mathcal{A} = \langle Q, \Sigma, Q_{\mathit{final}}, \Delta\rangle, \mathcal{B} = \langle Q', \Sigma, Q'_{\mathit{final}}, \Delta'\rangle$. A *homomorphism*, $h : \mathcal{A} \rightarrow \mathcal{B}$, is a function $h : Q \rightarrow Q'$ that maps final states to final states, and, for every transition $f(c_1, \ldots, c_k) \rightarrow c$ in $\mathcal{A}$ there exists a transition $f(h(c_1), \ldots, h(c_k)) \rightarrow h(c)$ in $\mathcal{B}$. An *isomorphism*[3] is a homomorphism $h : \mathcal{A} \rightarrow \mathcal{B}$ for which there exists an inverse homomorphism $h^{-1} : \mathcal{B} \rightarrow \mathcal{A}$ such that $h^{-1} \circ h = id_{\mathcal{A}}$ and $h \circ h^{-1} = id_{\mathcal{B}}$. The following holds:

▶ **Lemma 2.** *Let $h : \mathcal{A} \rightarrow \mathcal{B}$ be a homomorphism, $t \in T(\Sigma)$, and c be a state of $\mathcal{A}$. If $t \rightarrow^*_{\mathcal{A}} c$, then $t \rightarrow^*_{\mathcal{B}} h(c)$. In particular, $\mathcal{L}(\mathcal{A}) \subseteq \mathcal{L}(\mathcal{B})$.*

Proof. We prove the statement by induction on the structure of the term $t \in T(\Sigma)$. Assuming $t = f(t_1, \ldots, t_k)$ for $k \geq 0$[4] and $t \rightarrow^*_{\mathcal{A}} c$, then there exists states $c_1, \ldots, c_k$ such that $t_i \rightarrow^*_{\mathcal{A}} c_i$ and $\mathcal{A}$ contains the transition $f(c_1, \ldots, c_k) \rightarrow c$. By induction hypothesis $t_i \rightarrow^*_{\mathcal{B}} h(c_i)$ for $i = 1, \ldots, k$, and since h is a homomorphism, there exists a transition $f(h(c_1), \ldots, h(c_k)) \rightarrow h(c)$ in $\mathcal{B}$, proving that $t \rightarrow^*_{\mathcal{B}} h(c)$. ◀

We write $\mathcal{A} \sqsubseteq \mathcal{B}$ whenever there exists a homomorphism $\mathcal{A} \rightarrow \mathcal{B}$. Observe that $\sqsubseteq$ is a preorder relation. In the next section, we show that this preorder relation $\sqsubseteq$ becomes a partial order when restricted to E-graphs (Lemma 12).

We call an automaton $\mathcal{A}$ *deterministic* if $t \rightarrow^*_{\mathcal{A}} q_1$ and $t \rightarrow^*_{\mathcal{A}} q_2$ implies $q_1 = q_2$ for states q_1, q_2. We call $\mathcal{A}$ *reachable* if every state q accepts some ground term: $\exists t \in T(\Sigma), t \rightarrow^*_{\mathcal{A}} q$.

3 E-graphs and Equality Saturation

Most papers discussing E-graphs use an operational definition not suitable for a theoretical analysis. We introduce an equivalent definition of E-graphs in terms of tree automata, similar to Kozen's partial tree automata [17]. Throughout this section we fix the signature Σ.

[2] In this paper, we allow tree automata (and thus E-graphs) to have an infinite number of states and transitions. Talking about infinite E-graphs allow us to define the semantics of equality saturation even when the algorithm does not terminate.
[3] Notice that a bijective homomorphism is not necessarily an isomorphism.
[4] The base case is covered by the case $k = 0$.

3.1 E-graphs

▶ **Definition 3.** *An* E-graph *is a deterministic and reachable tree automaton $G = \langle Q, \Sigma, \Delta \rangle$ (without a set of final state Q_{final}).*

Our definition maps one-to-one to the classical definition of E-graphs: An E-class is a state $c \in Q$ of the tree automaton, and an E-node is a transition $f(c_1, \ldots, c_k) \to c$. A term t is *represented* by the E-class c if t is accepted by c, i.e. $t \to_G^* c$. In the literature, the sets of E-classes and E-nodes are denoted C and N respectively. We will use states/E-classes and transitions/E-nodes interchangeably in this paper. E-graphs do not define a set of "final" E-classes, and for that reason we omit the final states Q_{final} from Definition 3[5], similarly to [17].

▶ **Example 4.** *The E-graph H in Figure 1 is the automaton $\langle Q, \Sigma, \Delta \rangle$, where $\Sigma = \{a, f(\cdot,\cdot), g(\cdot,\cdot)\}$, there are four states $Q = \{c_1, \ldots, c_4\}$, and Δ consists of seven transitions:*

$$a() \to c_1 \qquad f(c_1, c_1) \to c_2 \qquad g(c_1, c_1) \to c_2 \qquad \ldots \qquad f(c_3, c_3) \to c_4 \qquad g(c_3, c_3) \to c_4$$

An example of rewritings is $f(a, a) \to_H f(c_1, a) \to_H f(c_1, c_1) \to_H c_2$, showing that the term $f(a, a)$ is represented by the E-class c_2.

It is folklore that E-graphs represent equivalences of terms. We make this observation formal, by defining the semantics of an E-graph to be a certain partial congruence. A *partial equivalence relation*, or PER, on a set A is a binary relation $\approx$ that is symmetric and transitive. Its *support* is the set $\mathit{supp}(\approx) \overset{\text{def}}{=} \{x \mid x \approx x\} \subseteq A$. Equivalently, a PER can be described by its support and an equivalence relation on the support. A PER on the set of terms $T(\Sigma)$ is *congruent* if $s_i \approx t_i$ for $i = 1, \ldots, n$ and $f(s_1, \ldots, s_n) \in \mathit{supp}(\approx)$ implies $f(s_1, \ldots, s_n) \approx f(t_1, \ldots, t_n)$. A PER is reachable if $f(s_1, \ldots, s_n) \in \mathit{supp}(\approx)$ implies $s_i \in \mathit{supp}(\approx)$, for $i = 1, \ldots, n$. A *Partial Congruence Relation (PCR)*[6] on $T(\Sigma)$ is a congruent and reachable PER.

An E-graph G induces a PCR $\approx_G$ defined as follows: $t_1 \approx t_2$ if there exists some E-class c in G that accepts both t_1 and t_2, i.e. $t_1 \to_G^* c \;{}_G^*\!\!\leftarrow t_2$. We check that $\approx_G$ is a PCR: $\approx_G$ is symmetric by definition, and transitivity follows from determinacy, because $t_1 \to_G^* c \;{}_G^*\!\!\leftarrow t_2$ and $t_2 \to_G^* c' \;{}_G^*\!\!\leftarrow t_3$ implies $t_1 \to_G^* c = c' \;{}_G^*\!\!\leftarrow t_3$. Suppose $f(s_1, \ldots, s_n) \to_G^* c$: then there exists states c_i s.t. $s_i \to_G^* c_i$, and a transition $f(c_1, \ldots, c_n) \to c$, proving reachability; if, in addition, $s_i \approx_G t_i$ for $i = 1, \ldots, n$, then $t_i \to_G^* c_i$, which implies $f(t_1, \ldots, t_n) \to_G^* c$, proving congruence, $f(s_1, \ldots, s_n) \approx_G f(t_1, \ldots, t_n)$.

▶ **Definition 5.** *The* semantics *of an E-graph G is the PCR $\approx_G$.*

▶ **Theorem 6.** *For any PCR $\approx$ over $T(\Sigma)$ there exists a unique G such that $(\approx_G) = (\approx)$.*

Proof sketch. The states of G are the equivalence classes of $\approx$, denoted as $[t]$ for $t \in \mathit{supp}(\approx)$, and the transitions are $f([t_1], \ldots, [t_n]) \to [f(t_1, \ldots, t_n)]$ for all $t_1, \ldots, t_n, f(t_1, \ldots, t_n)$ in the support of $\approx$. One can check by induction on the size of t that $t \in \mathit{supp}(\approx)$ iff $t \in \mathit{supp}(\approx_G)$, and $t \to_G^* [s]$ iff $t \approx s$, proving that $(\approx_G) = (\approx)$. ◀

Thus, the semantics of an E-graph G is a PCR $\approx_G$, which is a congruence on $\mathcal{L}(G) \overset{\text{def}}{=} \mathit{supp}(\approx_G)$. We say that G *represents* the set of terms $\mathcal{L}(G)$.

[5] Alternatively, consider $Q_{\mathit{final}} = Q$.
[6] PCRs are studied in the literature as congruences on partial algebras (e.g., [17]).

▶ **Example 7.** *Continuing Example 1, the semantics of the E-graph H in Figure 1 is the PCR $\approx_H$ that equates $a \approx_H a$ (witnessed by state c_1), $f(a,a) \approx_H g(a,a)$ (by state c_2), $f(f(a,a), g(a,a)) \approx_H g(f(a,a), f(a,a))$ (by state c_3), etc.*

▶ **Example 8.** *Let $\Sigma = \{a, f(\cdot)\}$. Consider the E-graph G with a single state c and transitions $a() \to c$, $f(c) \to c$. It represents infinitely many terms, $f^{(k)}(a)$, for $k \geq 0$, and its semantics is the PCR $a \approx_G f(a) \approx_G f(f(a)) \approx_G \cdots$*

▶ **Example 9.** *Let $\Sigma = \{a, f(\cdot), g(\cdot)\}$ and consider the infinite E-graph G with states $c, c_0, c_1, c_2, \ldots$ and transitions*

$$a \to c_0 \qquad\qquad f(c_i) \to c_{i+1} \qquad\qquad g(c_i) \to c \qquad\qquad i = 0, 1, 2, \ldots$$

The PCR consists of $g(a) \approx_G g(f(a)) \approx_G g(f(f(a))) \approx_G \ldots$, defined by the state c. No other distinct terms are in $\approx_G$, for example $f(a) \not\approx_G f(f(a))$ because they are represented by the distinct states c_1 and c_2 respectively. Although G represents a regular language, $\{f^{(k)}(a) \mid k \geq 0\} \cup \{g(f^{(k)}(a)) \mid k \geq 0\}$, its semantics $\approx_G$ cannot be captured by a finite E-graph. This example shows that $\approx_G$ differs from the Myhill-Nerode equivalence relation [16], under which all terms $f^{(k)}(a)$ would be equivalent. It also illustrates the subtle distinction between tree automata and E-graphs. An optimizer that wants to use the identity $g(x) = g(f(x))$, but not $x = f(x)$, needs this E-graph to represent all terms equivalent to $g(a)$, and cannot use the finite tree automaton accepting the regular language $\mathcal{L}(c)$ because that would incorrectly equate all terms $f^{(k)}(a)$.

Recall the definitions of tree automata homomorphisms in Sec. 2.2. When restricted to E-graphs, homomorphisms have some interesting properties:

▶ **Lemma 10.** *If $h : G \to H$ is a homomorphism, then $(\approx_G) \subseteq (\approx_H)$.*

Proof. Assume $t_1 \approx_G t_2$. Then there exists some E-class c where $t_1 \to^*_G c \; {}_G^*\!\!\leftarrow t_2$. By Lemma 2, $t_1 \to^*_H h(c) \; {}_H^*\!\!\leftarrow t_2$, implying $t_1 \approx_H t_2$. ◀

▶ **Lemma 11.** *There exists at most one homomorphism $h : G \to H$.*

Proof. Call the *weight* of a state c in G the size of the smallest term t such that $t \to^*_G c$. Since G is reachable, every state has a finite weight. Given two homomorphisms $h_1, h_2 : G \to H$, we prove by induction on the weight of c that $h_1(c) = h_2(c)$. Let t be a term of minimal size such that $t \to^*_G c$, and assume $t = f(t_1, \ldots, t_k)$, for $k \geq 0$. Then there exists states $c_1, \ldots, c_k$ such that $t_i \to^*_G c_i$, $i = 1, k$, and a transition $f(c_1, \ldots, c_k) \to c$ in G. By induction hypothesis $h_1(c_i) = h_2(c_i)$ for $i = 1, k$. By the definition of a homomorphism, H contains both transitions $f(h_1(c_1), \ldots, h_1(c_k)) \to h_1(c)$ and $f(h_2(c_1), \ldots, h_2(c_k)) \to h_2(c)$, and we conclude $h_1(c) = h_2(c)$ because H is deterministic. ◀

We call a tree automaton $\mathcal{A}$ rigid [13] if the identity mapping is the only homomorphism $\mathcal{A} \to \mathcal{A}$. It follows from Lemma 11 that every E-graph is a rigid tree automaton.

▶ **Lemma 12.** *$\sqsubseteq$ over E-graphs forms a partial order up to isomorphism.*

Proof. Obviously $\sqsubseteq$ is reflexive and transitive. To prove anti-symmetry, assume two homomorphisms $h : G \to H$, $h' : H \to G$. The composition $h' \circ h$ is a homomorphism $G \to G$, and, by uniqueness, it must be the identity on G; similarly, $h \circ h'$ is the identity on H, proving that h is an isomorphism, thus G, H are isomorphic. ◀

Next, we define models for term rewriting systems.

▶ **Definition 13.** *We say that an E-graph $H = \langle Q, \Sigma, \Delta\rangle$ is a* model *of a TRS $\mathcal{R}$ if, for every rule $lhs \to rhs$ in $\mathcal{R}$ and any substitution $\sigma : \mathsf{Var}(lhs) \to Q$, if $lhs[\sigma] \to^*_G c$ then $rhs[\sigma] \to^*_G c$. If G is another E-graph, then we say that H is a* model *for the pair $\mathcal{R}, G$ if $G \sqsubseteq H$ and H is a model of $\mathcal{R}$. H is a* universal model *if for any other model H', it holds that $H \sqsubseteq H'$.*

When it exists, the universal model is unique up to isomorphism, because $H \sqsubseteq H'$ and $H' \sqsubseteq H$ implies H, H' are isomorphic.

Continuing Example 7, let $\mathcal{R}$ consists of the rule $f(x, x) \to g(x, x)$, and let G, H be the E-graphs in Figure 1. G is not a model of $\mathcal{R}$, because for the substitution $\sigma(x) = c_1$ we have $lhs[\sigma] = f(c_1, c_1) \to_G c_2$, but $rhs[\sigma] = g(c_1, c_1) \not\to^*_G c_2$. On the other hand, one can check that H is a model of $\mathcal{R}$; in fact it is a model of $\mathcal{R}, G$, because $G \sqsubseteq H$.

Given an E-graph G and a TRS $\mathcal{R}$, equality saturation constructs a universal model H of $\mathcal{R}, G$, by repeatedly applying some simple operations on G, which we define next.

3.2 Operations over E-graphs

E-matching, Insertion, and Rebuilding are the building blocks of equality saturation. They defined in the literature operationally [19]. We provide here a formal definition, using tree automata terminology. Throughout this section we fix an E-graph $G = \langle Q, \Sigma, \Delta\rangle$.

E-matching a rule $lhs \to rhs \in \mathcal{R}$ in G returns the set of pairs (σ, c), where $\sigma : \mathsf{Var}(lhs) \to Q$ is a substitution such that $lhs[\sigma] \to^*_G c$. For example, considering the E-graph on the left of Figure 1 and the rule $f(x, x) \to g(x, x)$, E-matching returns $(\{x \mapsto c_i, y \mapsto c_i\}, c_{i+1})$ for $i = 1, 2, 3$. E-matching is analogous to computing the triggers for a TGD or EGD (Sec. 4.1).

The **Insertion** of a pair (t, c) into G, where $t \in T(\Sigma \cup Q)$ and $c \in Q$, returns an automaton $\mathcal{A}$ such that $G \sqsubseteq \mathcal{A}$ such that $t \to^*_{\mathcal{A}} c$. To define $\mathcal{A}$, we need the following:

▶ **Definition 14.** *Fix a term $t \in T(\Sigma \cup Q)$ and a state $c \in Q$. The* flattening *for t with root c, in notation $\mathsf{FL}(t \to^* c)$, or just FL when t, c are clear from the context, is an E-graph that has one distinct state q_u for each subterm u of t, and has a transition $f(q_{u_1}, \ldots, q_{u_k}) \to q_u$, for all subterms u of the form $u = f(u_1, \ldots, u_k)$ and $f \in \Sigma$. Moreover, it is enforced that the state of the root node is c (i.e., $q_t = c$). One can check that $t \to^*_{\mathsf{FL}} c$, and that $\approx_{\mathsf{FL}}$ is the identity on all subterms of t. Flattening is also called* normalization *[9].*

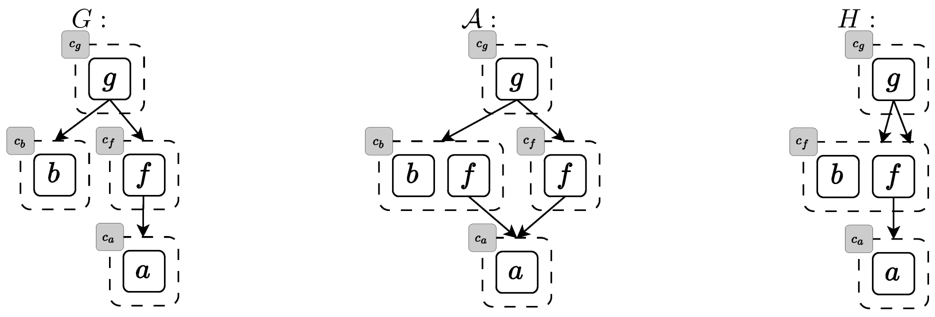


Figure 2 Example E-graphs on insertion and rebuilding.

The result of *inserting* (t, c) in G is $\mathcal{A} \overset{\text{def}}{=} G \cup \mathsf{FL}(t \to^* c)$ (i.e. we take the set-union of all states and all transitions). The result $\mathcal{A}$ is a reachable tree automaton, but it is non-deterministic in general, thus it is not an E-graph; the next operation, rebuilding, converts it back into an E-graph. $G \sqsubseteq \mathcal{A}$ holds, because the inclusion $G \to \mathcal{A}$ is a homomorphism.

As an example, if we insert the pair $(f(c_a), c_b)$ in the E-graph G in Figure 2, the result is $\mathcal{A}$ in the center of the figure; flattening $\mathsf{FL}(f(c_a) \to c_b)$ has a single transition $f(c_a) \to c_b$.

Rebuilding converts $\mathcal{A}$ into a deterministic automaton. Formally, let $\mathcal{A}$ be any reachable tree automaton, and recall that $\mathcal{A}$ may be infinite. The *congruence closure* $\mathsf{CC}(\mathcal{A})$ is an E-graph (i.e. deterministic, reachable automaton) such that $\mathcal{A} \sqsubseteq \mathsf{CC}(\mathcal{A})$ and, for any other E-graph G', if $\mathcal{A} \sqsubseteq G'$, then $\mathsf{CC}(\mathcal{A}) \sqsubseteq G'$. We prove the following in the full version:

▶ **Lemma 15.** *For any reachable tree automaton $\mathcal{A}$, $CC(\mathcal{A})$ exists and is unique.*

The procedure of computing $\mathsf{CC}(\mathcal{A})$, also known as *rebuilding* in EqSat literature [32], can be done efficiently in the finite case, for instance with Tarjan's algorithm [7]. The idea is to repeatedly find violating transitions $f(c_1, \ldots, c_k) \to c$ and $f(c_1, \ldots, c_k) \to c'$ with $c \neq c'$, and replace every occurrence of c with c', until fixpoint. This is similar to determinizing $\mathcal{A}$, but instead of constructing powerset states like $\{c_1, c_4, c_7\}$, we equate states $c_1 = c_4 = c_7$; thus, $\mathsf{CC}(\mathcal{A})$ has at most as many states as $\mathcal{A}$, and the procedure always terminates for finite $\mathcal{A}$, as merging shrinks the number of states.

▶ **Example 16.** *The tree automaton $\mathcal{A}$ in Figure 2 is non-deterministic, because $f(a) \to^* c_b$ and $f(a) \to^* c_f$. The congruence closure algorithm merges c_b and c_f, and produces the E-graph H in Figure 2. Notice that H represents strictly more terms than $\mathcal{A}$. For example, H represents $g(b, b)$, because $g(b, b) \to^*_H c_g$, but $\mathcal{A}$ does not represent $g(b, b)$.*

Least upper bound of E-graphs. Let $(G_i)_{i \in I}$ be a (possibly infinite) family of E-graphs. Recall that their least upper bound G is an E-graph such that G is an upper bound for every E-graph in the set, $G_i \sqsubseteq G$ for all $i \in I$, and for any other upper bound G', it holds that $G \sqsubseteq G'$. We prove the following in the full version:

▶ **Lemma 17** (Least upper bound)**.** *The least upper bound exists and is given by $CC(\mathcal{A})$, where $\mathcal{A}$ is the automaton consisting of the disjoint union of the states and the disjoint union of the transitions of all E-graphs G_i.*

We will denote the least upper bound by $\bigsqcup_{i \in I} G_i$. It is also possible to show that every family of E-graphs admits a greatest lower bound, by using a product construction [12], but we do not need it in this paper.

3.3 Equality saturation

The standard definition of equality saturation in the literature is procedural: given an E-graph $G = \langle Q, \Sigma, \Delta \rangle$ and TRS $\mathcal{R}$, equality saturation repeatedly applies matching/insertion/rebuilding. EqSat is undefined when this process does not terminate. We provide here an alternative definition, as the least fixpoint of an *immediate consequence operator* (ICO), and prove that it always exists. We start by introducing the ICO:

$$\mathsf{ICO}_{\mathcal{R}} \stackrel{\text{def}}{=} \mathsf{CC} \circ T_{\mathcal{R}} \tag{2}$$

$T_{\mathcal{R}}$ is the match/apply operator: it computes all E-matches then inserts the *rhs*'s into G:

$$T_{\mathcal{R}}(G) \stackrel{\text{def}}{=} G \cup \bigcup \{\mathsf{FL}(\mathit{rhs}[\sigma] \to^*_G c) \mid (\mathit{lhs} \to \mathit{rhs}) \in \mathcal{R}, \sigma : \mathit{Var}(\mathit{lhs}) \to Q, \mathit{lhs}[\sigma] \to^*_G c\}$$

CC is the rebuilding operator of Lemma 15.

▶ **Lemma 18.** *$ICO_{\mathcal{R}}$ is inflationary ($G \sqsubseteq ICO_{\mathcal{R}}(G)$ for all G) and monotone.*

Proof. That $\mathsf{ICO}_\mathcal{R}$ is inflationary follows from $G \sqsubseteq T_\mathcal{R}(G) \sqsubseteq \mathsf{CC}(T_\mathcal{R}(G))$. We prove that both $T_\mathcal{R}$ and CC are monotone. Let $H \overset{\text{def}}{=} \bigcup\{\mathsf{FL}(\mathit{rhs}[\sigma] \to^* c) \mid (\mathit{lhs} \to \mathit{rhs}) \in \mathcal{R}, \sigma : \mathit{Var}(\mathit{lhs}) \to Q, \mathit{lhs}[\sigma] \to_G^* c\}$, thus $T_\mathcal{R}(G) = G \cup H$. Any homomorphism $G \to G'$ can be extended to a homomorphism $G \cup H \to G' \cup H$, which proves that $T_\mathcal{R}$ is monotone. Consider two automata $\mathcal{A}, \mathcal{A}'$ and assume $\mathcal{A} \sqsubseteq \mathcal{A}'$, i.e. there exists a homomorphism from $\mathcal{A}$ to $\mathcal{A}'$. Denote $G \overset{\text{def}}{=} \mathsf{CC}(\mathcal{A})$, $G' \overset{\text{def}}{=} \mathsf{CC}(\mathcal{A}')$. Then, $\mathcal{A} \sqsubseteq \mathcal{A}' \sqsubseteq G'$, which implies $\mathcal{A} \sqsubseteq G'$. By the definition of $G = \mathsf{CC}(\mathcal{A})$, we have $G \sqsubseteq G'$, proving that CC is monotone. ◀

With Lemmas 17 and 18, we show the following in the full version:

▶ **Theorem 19.** *Fix an E-graph G, and consider the class $\mathcal{C}_G$ of E-graphs $G' \sqsupseteq G$. Then $\mathit{ICO}_\mathcal{R} : \mathcal{C}_G \to \mathcal{C}_G$ has a least fixpoint, given by*

$$\textsc{EqSat}(\mathcal{R}, G) \overset{\text{def}}{=} \bigsqcup_{i \geq 0} \mathit{ICO}_\mathcal{R}^{(i)}(G) \tag{3}$$

Furthermore, $\textsc{EqSat}(\mathcal{R}, G)$ is a universal model of $\mathcal{R}, G$; we call it equality saturation.

Given $G, \mathcal{R}$, our semantics of EqSat is the least fixpoint in (3), which is also the unique universal model of $G, \mathcal{R}$. When $\textsc{EqSat}(\mathcal{R}, G)$ is finite, then this coincides with the standard procedural definition in the literature. A common case (e.g., in program and query optimization settings) is when G represents a single term t, more precisely $G = \mathsf{FL}(t \to^* c)$ with fresh state c; in that case we denote $\textsc{EqSat}(\mathcal{R}, G)$ as $\textsc{EqSat}(\mathcal{R}, t)$.

Properties of equality saturation. We establish several basic facts of $\textsc{EqSat}$.

▶ **Lemma 20** (Inflationary). *$G \sqsubseteq \textsc{EqSat}(\mathcal{R}, G)$.*

▶ **Corollary 21.** *$\mathcal{L}(G) \subseteq \mathcal{L}(\textsc{EqSat}(\mathcal{R}, G))$ and $(\approx_G) \subseteq (\approx_{\textsc{EqSat}(\mathcal{R},G)})$.*

Lemma 20 follows from Lemma 18, while Corollary 21 follows from Lemma 2 and Lemma 10. Thus, $\textsc{EqSat}(\mathcal{R}, G)$ represents more terms than G, and identifies more pairs of terms than G. Next, we examine the relationship between the PCR defined by $\textsc{EqSat}(\mathcal{R}, G)$ and the relations $\mathcal{R}^*$ and $\approx_\mathcal{R}$ defined by the TRS $\mathcal{R}$ (see Sec. 2.1). If $\approx_1, \approx_2$ are two PCRs, then we denote by $\approx_1 \vee \approx_2$ the smallest PCR that contains both. We prove:

▶ **Lemma 22** (Representation). *Let $w \in T(\Sigma)$ be a term represented by some state of the E-graph $H \overset{\text{def}}{=} \textsc{EqSat}(\mathcal{R}, G)$. The following hold: $\mathcal{R}^*(w) \subseteq [w]_{\approx_H} \subseteq [w]_{\approx_\mathcal{R} \vee \approx_G}$.*

Proof. The definitions of E-matching and insertion imply that, if $u \to_\mathcal{R} v$ and u is represented by some state c of some E-graph K, then v is represented by the same state of the E-graph $\mathsf{ICO}(K) = \mathsf{CC}(T_\mathcal{R}(K))$. Therefore, if $u \to_\mathcal{R} v$ and u is represented by some state of H, then v is represented by the same state of $\mathsf{ICO}(H) = H$ (because H is a fixpoint of ICO). This implies that $\mathcal{R}^*(w) \subseteq [w]_{\approx_H}$.

For the second part, we denote by $G_k = \mathsf{ICO}^{(k)}(G)$, and check by induction on k that $[w]_{\approx_{G_k}} \subseteq [w]_{\approx_\mathcal{R} \vee \approx_G}$. When $k = 0$ then $G_0 = G$ and the claim is obvious. For the inductive step we observe that the only new identities introduced by G_{k+1} are justified by $\mathcal{R}$. ◀

In other words, if $w \to_\mathcal{R}^* v$, then $\textsc{EqSat}(\mathcal{R}, G)$ will equate w with v; and if $\textsc{EqSat}(\mathcal{R}, G)$ equates w with v then this can be derived from $\approx_\mathcal{R}$ and $\approx_G$. In general, $w \approx_\mathcal{R} v$ does not imply $w \approx_H v$. For a simple example, let $\mathcal{R} = \{a \to b\}$, thus $a \approx_\mathcal{R} b$, and let G represent only the term b. Then $H = \textsc{EqSat}(\mathcal{R}, G) = G$ represents only the term b, thus $a \not\approx_H b$.

▶ **Example 23.** *For some TRS, the starting E-graph G determines whether EqSat terminates in a finite number of steps. For a simple example, consider $\Sigma = \{f(\cdot), g(\cdot), a\}$, $\mathcal{R} = \{f(g(x)) \to g(f(x))\}$. If the initial E-graph G represents only the term $f(g(a))$ (and its subterms), then EqSat terminates, and the resulting $H \overset{def}{=} \textsc{EqSat}(\mathcal{R}, G)$ represents a PCR where $f(g(a)) \approx_H g(f(a))$. On the other hand, if G is the E-graph with states c_f, c_g and transitions $\{g(c_f) \to c_g, f(c_g) \to c_f, a \to c_f\}$, then G already represents infinitely many terms $\mathcal{L}(c_f) \cup \mathcal{L}(c_g)$, where $\mathcal{L}(c_f) = \{a, f(g(a)), f(g(f(g(a)))), \ldots\}$ and $\mathcal{L}(c_g) = \{g(a), g(f(g(a))), \ldots\}$. After equality saturation, $H = \textsc{EqSat}(\mathcal{R}, G)$ represents all terms in $T(\Sigma)$ where the numbers of occurrences $\#f, \#g$ of f, g satisfy $\#f \le \#g \le \#f + 1$. This is not a regular language, hence H is infinite, and EqSat will not terminate in a finite number of steps.*

▶ **Example 24.** *We show that both inclusions in Lemma 22 can be strict. Let $\mathcal{R} = \{a \to b, c \to b\}$ and let G be the E-graph representing a single term $f(a, b)$ with transitions $\{\ a \to c_a, b \to c_b, f(c_a, c_b) \to c_f\ \}$. Then $H = \textsc{EqSat}(\mathcal{R}, G)$ has transitions: $\{\ a \to c_a, b \to \underline{c_a}, f(c_a, \underline{c_a}) \to c_f\}$. We have:[7] $\mathcal{R}^*(f(a, b)) = f(a|b, b)$, $[f(a, b)]_{\approx_H} = f(a|b, a|b)$, and $[f(a, b)]_{\approx_{\mathcal{R}} \vee \approx_G} = [f(a, b)]_{\approx_{trs}} = f(a|b|c, a|b|c)$. All three sets are different.*

However, the three expressions in Lemma 22 are equal in an important special case:

▶ **Corollary 25.** *Suppose $\mathcal{R}$ is a variable-preserving term rewriting system. Let $Sym(\mathcal{R}) = \mathcal{R} \cup \mathcal{R}^{-1}$, and let $w \in T(\Sigma)$ be a term represented by some state of the E-graph $H^{\leftrightarrow} \overset{def}{=} \textsc{EqSat}(Sym(\mathcal{R}), G)$. The following hold: $(Sym(\mathcal{R}))^*(w) = [w]_{\approx_{H^{\leftrightarrow}}} = [w]_{\approx_{\mathcal{R}} \vee \approx_G}$.*

Let G be a finite E-graph. If $\textsc{EqSat}(\mathcal{R}, G)$ is infinite, then EqSat does not terminate in a finite number of steps. Somewhat surprisingly, the converse does hold: if $\textsc{EqSat}(\mathcal{R}, G)$ is finite, then EqSat terminates in a finite number of steps. This follows from the next lemma, whose proof is deferred to the full version.

▶ **Lemma 26** (Finite convergence). *Let $\mathcal{G} : G_1 \sqsubset G_2 \sqsubset \ldots$ be an ascending sequence of finite E-graphs. If $G_\infty = \bigsqcup_i G_i$ is finite, then the sequence $\mathcal{G}$ is finite.*

▶ **Corollary 27** (Finite convergence of EqSat). *Let $\mathcal{R}$ be a term rewriting system and G be a finite E-graph. If $\textsc{EqSat}(\mathcal{R}, G)$ is finite, EqSat converges in a finite number of steps.*

4 Equality saturation and the chase

In this section, we briefly review necessary background on databases and the chase. Then, we will show the fundamental connections between equality saturation and the chase.

4.1 The Chase Procedure

Databases and conjunctive queries. A relational database *schema* is a tuple of relation names $\mathcal{S} = (R_1, \ldots, R_m)$ with associated arities $ar(R_i)$. A database instance is a tuple of relation instances $I = (R_1^I, \ldots, R_m^I)$, where $R_i^I \subseteq \mathsf{Dom}^{ar(R_i)}$ for some domain Dom. We allow an instance to be infinite. We often view a tuple $\vec{a}$ in R_i^I as an atom $R_i(\vec{a})$, and view the instance I as a set of atoms. The domain Dom is the disjoint union of set of *constants* and a set of *marked nulls*.

[7] We use $f(a|b, b)$ as a shorthand for $\{f(t, b) \mid t = a \vee t = b\}$ (as in regular expressions) and similarly for other terms.

A *conjunctive query* $\lambda(\vec{x})$ is a formula with free variables $\vec{x}$ of the form $R_1(\vec{x_1}) \wedge \ldots \wedge R_k(\vec{x_k})$, where each $\vec{x}_i$ is a tuple of variables from $\vec{x}$. The *canonical database* of a conjunctive query consists of all the tuples $R_i(\vec{x_i})$, where the variables $\vec{x}$ are considered marked nulls.

Let I, J be two database instances. A *homomorphism* from I to J, in notation $h : I \to J$, is a a function $h : \mathsf{Dom}(I) \to \mathsf{Dom}(J)$ that is the identity on the set of constants, and maps each atom $R(\vec{a}) \in I$ to an atom $R(h(\vec{a})) \in J$. The notion of homomorphism immediately extends to conjunctive queries and/or database instances. The output of a conjunctive query $\lambda(\vec{x})$ on a database I is defined as the set of homomorphisms from $\lambda(\vec{x})$ to I. We say that a database instance I *satisfies* a conjunctive query $\lambda(\vec{x})$, denoted by $I \models \exists \vec{x} \lambda(\vec{x})$, if there exists a homomorphism $\lambda(\vec{x}) \to I$.

Dependencies. TGDs and EGDs describe semantic constraints between relations. A TGD is a first-order formula of the form $\lambda(\vec{x}, \vec{y}) \to \exists \vec{z}. \rho(\vec{x}, \vec{z})$ where $\lambda(\vec{x}, \vec{z})$ and $\rho(\vec{x}, \vec{y})$ are conjunctive queries with free variables in $\vec{x} \cup \vec{y}$ and $\vec{x} \cup \vec{z}$. An EGD is a first-order formula of the form $\lambda(\vec{x}) \to x_i = x_j$ where $\lambda(\vec{x})$ is a conjunctive query with free variables in $\vec{x}$ and $\{x_i, x_j\} \subseteq \vec{x}$.

Fix a set of TGDs and EGDs Γ. If I is a database instance and $d \in \Gamma$, then a *trigger* for d in I is a homomorphism from $\lambda(\vec{x}, \vec{y})$ (resp. $\lambda(\vec{x})$) to I. An *active trigger* is a trigger h such that, if d is a TGD, then no extension h to a homomorphism $h' : \rho(\vec{x}, \vec{z}) \to I$ exists, and, if d is an EGD, then $h(x_i) \neq h(x_j)$. We say that I is model for Γ, and write $I \models \Gamma$, if it has no active triggers.

Given Γ and I we say that some database instance J is a *model* for Γ, I, if $J \models \Gamma$ and there exists a homomorphism $I \to J$. J is called *universal model* if there is a homomorphism from J to every model of Γ and I. Universal models are unique up to homomorphisms.

The chase. The chase is a fixpoint algorithm for computing universal models. We consider two variants of the chase here: the *standard chase* and *Skolem chase*. Both the standard chase and the Skolem chase produce a universal model of Γ, I [8, 3, 18]. The standard chase computes answers by deriving a sequence of *chase steps* until all dependencies are satisfied. A chase step, denoted as $I \xrightarrow{d,h} J$, takes as inputs an instance I, a homomorphism h, and a dependency d, where h is an active trigger of d in I, and produces an output instance J by adding some tuples (for TGDs) or collapsing some elements (for EGDs). Specifically, if d is a TGD, the chase step extends I with the tuple $h'(\rho(\vec{x}, \vec{z}))$, where h' is an extension of h that maps the variables $\vec{z}$ on which h is undefined to fresh marked nulls. If d is an EGD, if $h(x_i)$ (or $h(x_j)$) is a marked null, a chase step replaces in I every occurrence of $h(x_i)$ with $h(x_j)$ (or $h(x_j)$ with $h(x_i)$). If neither $h(x_i)$ nor $h(x_j)$ is a marked null and $h(x_i) \neq h(x_j)$, then the chase fails.

A standard chase sequence starting at I_0 is a sequence of successful chase steps $I_0 \xrightarrow{d_1,h_1} I_1 \xrightarrow{d_2,h_2} \cdots$ that is *fair*: for all $i \geq 0$, for each dependency d and active trigger h of d in I_i, some $j \geq i$ must exist such that h is no longer an active trigger of d in I_j. The *result* of a (possibly infinite) chase sequence is $\bigcup_{i \geq 0} \bigcap_{j \geq i} I_j$ [3]. A chase sequence is *terminating* if it ends with I_n and $I_n \models \Gamma$, in which case I_n is the result of the chase sequence. The standard chase is non-deterministic: depending on the order of firing, the chase sequence can be different. Different chase sequences can even differ on whether they terminate.

The Skolem chase, discussed in [18], differs from the standard chase in several ways. It first *skolemizes* each TGD $d : \lambda(\vec{x}, \vec{y}) \to \exists \vec{z}. \rho(\vec{x}, z_1, \ldots, z_k)$ to $\lambda(\vec{x}, \vec{y}) \to \rho(\vec{x}, f^d_{z_1}(\vec{x}), \ldots, f^d_{z_k}(\vec{x}))$, where each $f^d_{z_j}$ is an uninterpreted function from $\mathsf{Dom}^{|\vec{x}|}$ to Dom. The result of the Skolem chase, denoted as $\textsc{SklCh}(\Gamma, I)$, is the least fixpoint of the immediate consequence operator (ICO) of the Skolemized TGDs. Note that the Skolem chase does not directly handle EGDs but uses a technique called *singularization* [18] to simulate EGDs with TGDs.

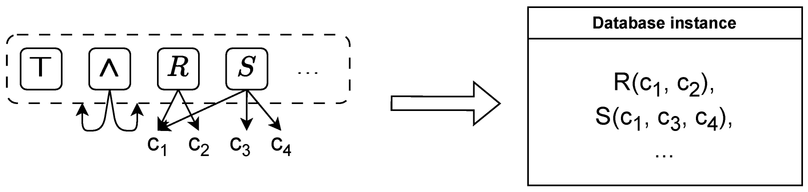


Figure 3 Mapping results of encoded EqSat back to database instances.

4.2 Reducing the Skolem chase to equality saturation

In this section, we show how to reduce the Skolem chase to EqSat. We only consider TGDs, since in the Skolem chase, EGDs are modeled as TGDs using singularization [18].

We show an encoding where there exists a simple mapping from E-graphs to database instances, defined by

$$\xi(G) = \{R(c_1, \ldots, c_k) \in \mathcal{L}(G) \mid R \text{ is a relation symbol in } \mathcal{S}\}$$

such that, given a set of dependencies Γ, running EqSat on an encoded term rewriting system from Γ corresponds to running the Skolem chase on the set of dependencies via ξ. Intuitively, given an E-grpah, ξ collects every term that corresponds to a tuple from the language of G. An illustration of ξ is shown in Figure 3.

▶ **Theorem 28.** *Given a database schema $\mathcal{S} = (R_1, \ldots, R_m)$, a set of TGDs Γ, and an initial database I, it is possible to define a signature Σ, a term rewriting system $\mathcal{R}$ over Σ, and an initial term t such that*

$$\xi(\textsc{EqSat}(\mathcal{R}, t)) = \textsc{SklCh}(\Gamma, I).$$

Moreover, the Skolem chase terminates if and only if equality saturation terminates.

The intuition for the construction is that we can uniformly treat relational atoms as E-nodes contained in a special E-class, and Skolem functions naturally correspond to terms in EqSat. More specifically,

- Add symbols $\{\top, \wedge(\cdot,\cdot)\}$ to the signature Σ. Add rewrite $r_\top : \top \to \wedge(\top, \top)$ to $\mathcal{R}$. Let the initial term t be $\top$.
- Add every Skolem function symbol to Σ, and for every n-ary relational symbol $R \in \mathcal{S}$, add a n-ary function symbol to Σ, and add rewrite rule $r_R : R(x_1, \ldots, x_n) \to \top$.
- For every Skolemized TGD

  $$d:\ R_1\left(\vec{x_1}, \vec{y_1}\right) \wedge \ldots \wedge R_n\left(\vec{x_n}, \vec{y_n}\right) \to R_1'\left(\vec{x_1'}, \vec{f^d_z}_1\right) \wedge \ldots \wedge R_m'\left(\vec{x_m'}, \vec{f^d_z}_m\right),$$

  replace the conjunctions in the head and body with nested applications of $\wedge$ and $\top$:

  $$r_d:\ \wedge\left(R_1\left(\vec{x_1}, \vec{y_1}\right), \wedge\left(\ldots \wedge\left(R_n\left(\vec{x_n}, \vec{y_n}\right), \top\right)\right)\right) \\ \to \wedge\left(R_1'\left(\vec{x_1'}, \vec{f^d_z}_1\right), \wedge\left(\ldots \wedge\left(R_m'\left(\vec{x_m'}, \vec{f^d_z}_m\right), \top\right)\right)\right)$$

  and add r_d to $\mathcal{R}$.
- For each constant c in the input database I, add a nullary function symbol c to Σ. For each tuple $t = R(c_1, \ldots, c_n)$ in the input database I, add rewrite $r_t : \top \to R(c_1, \ldots, c_n)$.

Proof of Theorem 28. See the full version. ◀

4.3 Reducing equality saturation to the standard chase

We show how to reduce equality saturation to the standard chase. The encoding itself is straightforward. However, the standard chase is non-deterministic and can have different chase sequences, so a natural question is what kind of the chase sequence will converge finitely, given that EqSat terminates, and vice versa. We show that as long as the chase sequence applies EGDs frequent enough, the chase sequence will always converge. We capture this notion as EGD-fairness.

▶ **Definition 29.** *Given a database schema $\mathcal{S}$, a set of dependencies Γ over $\mathcal{S}$, and an initial database I_0. We call a chase sequence $I_0, I_1, \ldots$ of Γ and I EGD-fair if for every i, either I_i is a model of Γ and the chase terminates, or there exists some $j > i$ such that I_j is a model of the EGD subset of Γ.*

Given that EqSat terminates, what can we say about chase sequences that are not EGD-fair? In fact, such chase sequences may not terminate. Despite this, it can be shown that the result of such chase sequences, terminating or not, is isomorphic to the result of equality saturation (when encoded as a database). On the other hand, to show that equality saturation terminates, it is sufficient to show an arbitrary chase sequence terminates.

The following theorem shows the connection between EqSat and the standard chase.

▶ **Theorem 30.** *Given signature Σ, a set of rewrite rules $\mathcal{R}$ over Σ, and an initial E-graph G, it is possible to define a relational schema $\mathcal{S}$, a set of dependencies Γ over $\mathcal{S}$, and an initial database I over $\mathcal{S}$. The following three statements are equivalent:*
1. *Equality saturation terminates for $\mathcal{R}$ and t.*
2. *There exists a terminating chase sequence of the standard chase for Γ and I.*
3. *All EGD-fair chase sequences of the standard chase terminate for Γ and I.*

Moreover, denote the result of an arbitrary chase sequence as I_∞. If equality saturation terminates, I_∞ is isomorphic to the database encoding the resulting E-graph of $\textsc{EqSat}(\mathcal{R}, G)$.

The encoding consists of two steps. First, we can encode an E-graph as a database. Second, we encode the match/apply operator and congruence closure operator as a set of TGDs and EGDs. To encode an E-graph as a database:
- Take the domain Dom to be the set of all E-classes, which are treated as marked nulls.
- For every function symbol f of arity n, add relation symbol R_f of arity $n+1$ to $\mathcal{S}$.
- For every E-node $f(c_1, \ldots, c_n) \to c$, add a tuple $R_f(c_1, \ldots, c_n, c)$ to the database I.

Under this encoding, each E-class is treated as a marked null, and each E-node is treated as a tuple.

The encoding of the match/apply operator and congruence closure operator is plain:
- For every function symbol f of arity n, add a functional dependency $R_f(x_1, \ldots, x_n, x) \wedge R_f(x_1, \ldots, x_n, x') \to x = x'$ to Γ.
- For every rewrite rule *lhs* $\to$ *rhs* in $\mathcal{R}$, flatten the left- and right-hand side into conjunctions of relational atoms, unify the variable denoting the root node of *lhs* with that of *rhs*, and add existential quantifiers to the head accordingly. For example, rule $f(f(x,y),z) \to f(x, f(y,z))$ is flattened into $R_f(x,y,w_1) \wedge R_f(w_1,z,r) \to \exists w_2, R_f(x,y,w_2) \wedge R_f(w_2,v,r)$. There are two corner cases to the above translations. First, if *lhs* is a single variable x, we need to introduce n rules of the form $R_f(y_1, \ldots, y_k, x) \to \ldots$, one for each function symbol, to "ground" x. For instance, suppose $\Sigma = \{f(\cdot,\cdot), g(\cdot)\}$, rewrite rule $x \to g(x)$ is flattened into two dependencies: $R_f(y_1, y_2, x) \to R_g(x,x)$ and $R_g(y_1, x) \to R_g(x,x)$. Second, in the case that the right-hand side is a single variable x, we need to add an EGD instead of a TGD. For example, rule $f(x,y) \to x$ is encoded as an EGD $R_f(x,y,r) \to x = r$.

Proof of Theorem 30. See the full version. ◀

5 The termination theorems of equality saturation

Finally, we present our main results here.

▶ **Theorem 31** (Single-instance termination). *The following problem is R.E.-complete:*

- *Instance: A term rewriting system R, a term t.*
- *Question: Does EqSat terminate with R and t?*

▶ **Theorem 32** (All-term-instance termination). *The following problem is Π_2-complete:*

- *Instance: A term rewriting system R.*
- *Question: Does EqSat terminate with R and t for all terms t?*

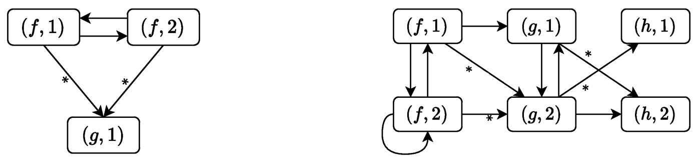
While Theorem 31 follows immediately from the fact the the Skolem chase is undecidable, our proof in the full version is based on Narendran et al. [21], which allows us to also show Theorem 32. We encode a Turing machine as a term rewriting system with the property that the congruence classes of initial configurations corresponds to traces of running such configurations, and that EqSat terminates if and only if congruence class is finite. For the all-term-instance case, we then show that the congruence class of an arbitrary term is infinite if and only if the congruence class of an initial configuration is. The actual proof is slightly more involved so we refer the reader to the full version for more details.

The technique above does not apply to the all-E-graph-instance case, however. The all-E-graph-instance termination can be thought of as having inputs both a term and a set of ground identities, and we have no control over the latter. Still, we are able to prove that this problem is undecidable by a reduction from the Post correspondence problem (see the full version), while the exact upper bound is unknown.

▶ **Theorem 33** (All-E-graph-instance termination). *The following problem is undecidable:*

- *Instance: A term rewriting system R.*
- *Question: Does EqSat terminate with R and G for all E-graphs G?*

6 Weak term acyclity for equality saturation termination



(a) Example 34. **(b)** Example 35.

Figure 4 Example weak term dependency graphs. Special edges are marked with $*$.

We can adapt the classic weak acyclicity criterion [8], which is used to show the termination of the chase algorithm, to equality saturation. The adapted criterion, which we call *weak term acyclicity*, is more powerful than simply translating EqSat rules to TGDs/EGDs and applying weak acyclicity. We demonstrate weak term acyclicity with two examples, and the full definition can be found at the full version.

▶ **Example 34.** *Consider $\mathcal{R} = \{f(f(x, y), z) \to g(f(z, x))\}$. This ruleset is weakly term acyclic, with the weak term dependency graph shown in Figure 4a. Note however if we derive the dependencies Γ using the method in Sec. 4.3 from $\mathcal{R}$, Γ is not weakly acyclic.*

▶ **Example 35.** *Consider $\mathcal{R} = \{g(f(x_1, y_1), f(z_1, x_1)) \to g(z_1, f(y_1, x_1)), g(x_2, y_2) \to h(y_2, g(y_2, x_2))\}$. This ruleset is weakly term acyclic. Its weak term dependency graph is shown in Figure 4b.*

7 Conclusion

We have presented a semantic foundation for E-graphs and EqSat: We identified E-graphs as reachable and deterministic tree automata and defined the result of EqSat as the least fixpoint according to E-graph homomoprhisms. We defined the universal model of E-graphs and showed the fixpoint EqSat produces is the universal model (Theorem 19). We showed several basic properties about E-graphs, including a finite convergence lemma (Lemma 26). We then established connections between EqSat and the chase in both directions (Sec. 4) and characterize chase sequences that correspond to EqSat with EGD-fairness (Definition 29). Our main results are on the terminations of EqSat in three cases: single-instance, all-term-instance, and all-E-graph-instance. Finally, adapting ideas from weak acyclicity for the chase, we defined weak term acyclicity which implies EqSat termination.

The correspondence between EqSat and the chase established in this paper may help further port the rich results of database theory to EqSat, as the current paper only scratches the surface of the deep literatures of the chase. Another direction is to use our better understanding of EqSat to design more efficient and expressive EqSat tools and better support downstream applications of EqSat. Finally, many problems about EqSat are still open. For example, the exact upper bound of the all-E-graph-instance termination is not known. Other problems include rule scheduling, evaluation algorithm, and E-graph extraction.

References

1 Franz Baader and Tobias Nipkow. *Term Rewriting and All That*. Cambridge University Press, USA, 1999.

2 Leo Bachmair, Ashish Tiwari, and Laurent Vigneron. Abstract congruence closure. *J. Autom. Reason.*, 31(2):129–168, 2003. doi:10.1023/B:JARS.0000009518.26415.49.

3 Michael Benedikt, George Konstantinidis, Giansalvatore Mecca, Boris Motik, Paolo Papotti, Donatello Santoro, and Efthymia Tsamoura. Benchmarking the chase. In *Proceedings of the 36th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, PODS '17, pages 37–52, New York, NY, USA, 2017. Association for Computing Machinery. doi:10.1145/3034786.3034796.

4 Samuel Coward, George A. Constantinides, and Theo Drane. Automating constraint-aware datapath optimization using e-graphs. *2023 60th ACM/IEEE Design Automation Conference (DAC)*, pages 1–6, 2023. URL: https://api.semanticscholar.org/CorpusID:257353847.

5 Alin Deutsch, Alan Nash, and Jeff Remmel. The chase revisited. In *Proceedings of the Twenty-Seventh ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS '08, pages 149–158, New York, NY, USA, 2008. Association for Computing Machinery. doi:10.1145/1376916.1376938.

6 RisingLight developers. RisingLight: An Educational OLAP Database System, December 2022. URL: https://github.com/risinglightdb/risinglight.

7 Peter J. Downey, Ravi Sethi, and Robert Endre Tarjan. Variations on the common subexpression problem. *Journal of the ACM*, 27(4):758–771, October 1980. doi:10.1145/322217.322228.

8 Ronald Fagin, Phokion G. Kolaitis, Renée J. Miller, and Lucian Popa. Data exchange: semantics and query answering. *Theoretical Computer Science*, 336(1):89–124, 2005. Database Theory. doi:10.1016/j.tcs.2004.10.033.

9 Thomas Genet. Termination criteria for tree automata completion. *Journal of Logical and Algebraic Methods in Programming*, 85(1, Part 1):3–33, 2016. Rewriting Logic and its Applications. doi:10.1016/j.jlamp.2015.05.003.

10 Tomasz Gogacz and Jerzy Marcinkowski. All–instances termination of chase is undecidable. In Javier Esparza, Pierre Fraigniaud, Thore Husfeldt, and Elias Koutsoupias, editors, *Automata, Languages, and Programming*, pages 293–304, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg. doi:10.1007/978-3-662-43951-7_25.

11 Gösta Grahne and Adrian Onet. Anatomy of the chase. *Fundam. Informaticae*, 157(3):221–270, 2018. doi:10.3233/FI-2018-1627.

12 Sumit Gulwani, Ashish Tiwari, and George C. Necula. Join algorithms for the theory of uninterpreted functions. In Kamal Lodaya and Meena Mahajan, editors, *FSTTCS 2004: Foundations of Software Technology and Theoretical Computer Science*, pages 311–323, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

13 Pavol Hell and Jaroslav Nesetril. Images of rigid digraphs. *Eur. J. Comb.*, 12(1):33–42, 1991. doi:10.1016/S0195-6698(13)80005-4.

14 Rajeev Joshi, Greg Nelson, and Keith Randall. Denali: A goal-directed superoptimizer. *SIGPLAN Not.*, 37(5):304–314, May 2002. doi:10.1145/543552.512566.

15 Donald E. Knuth. A generalization of dijkstra's algorithm. *Inf. Process. Lett.*, 6(1):1–5, 1977. doi:10.1016/0020-0190(77)90002-3.

16 Dexter Kozen. On the myhill-nerode theorem for trees. *Bulletin of the EATCS*, 47:170–173, 1992.

17 Dexter Kozen. *Partial Automata and Finitely Generated Congruences: An Extension of Nerode's Theorem*, pages 490–511. Birkhäuser Boston, Boston, MA, 1993. doi:10.1007/978-1-4612-0325-4_16.

18 Bruno Marnette. Generalized schema-mappings: From termination to tractability. In *Proceedings of the Twenty-Eighth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*, PODS '09, pages 13–22, New York, NY, USA, 2009. Association for Computing Machinery. doi:10.1145/1559795.1559799.

19 Leonardo Moura and Nikolaj Bjørner. Efficient e-matching for smt solvers. In *Proceedings of the 21st International Conference on Automated Deduction: Automated Deduction*, CADE-21, pages 183–198, Berlin, Heidelberg, 2007. Springer-Verlag. doi:10.1007/978-3-540-73595-3_13.

20 Chandrakana Nandi, Max Willsey, Adam Anderson, James R. Wilcox, Eva Darulova, Dan Grossman, and Zachary Tatlock. Synthesizing structured CAD models with equality saturation and inverse transformations. In *Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI 2020, pages 31–44, New York, NY, USA, 2020. Association for Computing Machinery. doi:10.1145/3385412.3386012.

21 Paliath Narendran, Colm Ó'Dúnlaing, and Heinrich Rolletschek. Complexity of certain decision problems about congruential languages. *Journal of Computer and System Sciences*, 30(3):343–358, 1985. doi:10.1016/0022-0000(85)90051-0.

22 Charles Gregory Nelson. *Techniques for Program Verification.* PhD thesis, Stanford University, Stanford, CA, USA, 1980. AAI8011683.

23 Hung Q. Ngo, Christopher Ré, and Atri Rudra. Skew strikes back: new developments in the theory of join algorithms. *SIGMOD Rec.*, 42(4):5–16, 2013. doi:10.1145/2590989.2590991.

24 Pavel Panchekha, Alex Sanchez-Stern, James R. Wilcox, and Zachary Tatlock. Automatically improving accuracy for floating point expressions. *SIGPLAN Not.*, 50(6):1–11, June 2015. doi:10.1145/2813885.2737959.

25 Maximilian Schleich, Amir Shaikhha, and Dan Suciu. Optimizing tensor programs on flexible storage, 2022. doi:10.48550/arXiv.2210.06267.

26 Wayne Snyder. A fast algorithm for generating reduced ground rewriting systems from a set of ground equations. *J. Symb. Comput.*, 15(4):415–450, April 1993. doi:10.1006/jsco.1993.1029.

27 Dan Suciu, Yisu Remy Wang, and Yihong Zhang. Semantic foundations of equality saturation, 2025. arXiv:2501.02413.

28 Ross Tate, Michael Stepp, Zachary Tatlock, and Sorin Lerner. Equality saturation: A new approach to optimization. In *Proceedings of the 36th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '09, pages 264–276, New York, NY, USA, 2009. ACM. doi:10.1145/1480881.1480915.

29 Alexa VanHattum, Rachit Nigam, Vincent T. Lee, James Bornholt, and Adrian Sampson. *Vectorization for Digital Signal Processors via Equality Saturation*, pages 874–886. Association for Computing Machinery, New York, NY, USA, 2021. doi:10.1145/3445814.3446707.

30 Yisu Remy Wang, Shana Hutchison, Jonathan Leang, Bill Howe, and Dan Suciu. SPORES: Sum-product optimization via relational equality saturation for large scale linear algebra. *Proceedings of the VLDB Endowment*, 2020.

31 Yisu Remy Wang, Mahmoud Abo Khamis, Hung Q Ngo, Reinhard Pichler, and Dan Suciu. Optimizing recursive queries with program synthesis. *arXiv preprint arXiv:2202.10390*, 2022. arXiv:2202.10390.

32 Max Willsey, Chandrakana Nandi, Yisu Remy Wang, Oliver Flatt, Zachary Tatlock, and Pavel Panchekha. Egg: Fast and extensible equality saturation. *Proc. ACM Program. Lang.*, 5(POPL), January 2021. doi:10.1145/3434304.

33 Yichen Yang, Phitchaya Mangpo Phothilimtha, Yisu Remy Wang, Max Willsey, Sudip Roy, and Jacques Pienaar. Equality saturation for tensor graph superoptimization. In *Proceedings of Machine Learning and Systems*, 2021. arXiv:2101.01332.

34 Yihong Zhang, Yisu Remy Wang, Oliver Flatt, David Cao, Philip Zucker, Eli Rosenthal, Zachary Tatlock, and Max Willsey. Better together: Unifying datalog and equality saturation. *Proc. ACM Program. Lang.*, 7(PLDI), June 2023. doi:10.1145/3591239.

35 Yihong Zhang, Yisu Remy Wang, Max Willsey, and Zachary Tatlock. Relational e-matching. *Proc. ACM Program. Lang.*, 6(POPL), January 2022. doi:10.1145/3498696.