Exact Pattern-Aware Extraction for Equality Saturation via Bounded-Depth Tree Covering

Zi Cheng , Mengting Yuan * , Lefei Zhang *

Posted Date: 9 April 2026

doi: 10.20944/preprints202604.0583.v1

Keywords: equality saturation; e-graph extraction; tree covering; dynamic programming; pattern matching; AND-OR DAG

Article

Exact Pattern-Aware Extraction for Equality Saturation via Bounded-Depth Tree Covering

Zi Cheng, Mengting Yuan * and Lefei Zhang *

School of Computer Science, Wuhan University, Wuhan 430072, China
* Correspondence: ymt@whu.edu.cn (M.Y.); zhanglefei@whu.edu.cn (L.Z.)

Abstract

Equality saturation explores equivalent program expressions via e-graphs, and its final step—extraction—selects one representative per equivalence class to form an output tree. Standard extraction minimizes a decomposable, single-node cost function that cannot capture multi-node structural patterns exploited by downstream systems such as SMT preprocessors and compiler backends. We formalize pattern-aware extraction as a weighted pattern cover problem on AND-OR directed acyclic graphs and establish its correspondence to tree covering in compiler instruction selection. Three challenges arise: annotation ambiguity from multiple candidates per class, context-dependent selection from depth-2 templates, and DAG sharing conflict. We show that the coupled selection-tiling problem reduces to a tree DP with three mutually exclusive tile-role states—independent, tile-root, and tile-leaf—generalizing BURS tree covering from fixed trees to AND-OR DAGs. A bottom-up pass computes optimal DP values; a top-down pass traces back decisions to produce the output tree. For template depth at most two, the algorithm computes an exact optimum in $O(N \cdot K \cdot |\mathcal{P}| \cdot C_{\max})$ time. Experiments on SMT-COMP hardware verification benchmarks show up to $31\times$ higher weighted pattern coverage than standard extraction, with depth-2 tiling contributing 45–51% additional improvement and overhead remaining within 2–$3\times$, demonstrating exact, context-sensitive extraction at practical cost.

Keywords: equality saturation; e-graph extraction; tree covering; dynamic programming; pattern matching; AND-OR DAG

1. Introduction

Equality saturation explores the space of equivalent program expressions by encoding them in an e-graph (Willsey, Nandi, Wang, Flatt, Tatlock & Panchekha, 2021; Tate, Stepp, Tatlock & Lerner, 2009; Nelson & Oppen, 1980). Its final step—*extraction*—selects one representative per equivalence class to form a concrete output tree, directly determining the quality of the optimization pipeline.

Downstream consumers of e-graph output—including SMT solver preprocessors (De Moura & Bjørner, 2008) and compiler instruction selectors (Blindell, 2016; Aho, Ganapathi & Tjiang, 1989)—perform their most valuable simplifications through predefined pattern-matching passes that recognize specific multi-node operator configurations. However, these systems apply rewrites through a fixed sequence of local passes, each transforming expressions one at a time, limiting their ability to reshape an expression into the triggering form for their own heuristics. Equality saturation can bridge this gap by exploring equivalent expressions simultaneously and locating pattern-satisfying variants that no fixed sequence of local rewrites would produce. Extraction must therefore select among equivalents by downstream pattern value rather than by a generic proxy such as expression size.

Standard extraction discards this opportunity. The widely used egg framework (Willsey, Nandi, Wang, Flatt, Tatlock & Panchekha, 2021) employs a decomposable, single-node cost function—typically AST size—that evaluates each candidate independently of its parent context and commits to a global selection reused wherever the class is referenced. This mechanism is structurally unable to encode

 doi:10.20944/preprints202604.0583.v1

the joint value of multi-node pattern matches: two candidates of equal AST size that differ only in pattern-matching value receive identical scores (Section 3).

We formalize pattern-aware extraction as a weighted pattern cover problem on AND-OR DAGs, establishing its correspondence to tree covering in compiler instruction selection (Blindell, 2016; Pelegri-Llopart & Graham, 1988; Aho, Ganapathi & Tjiang, 1989). Three challenges arise when migrating tree covering to e-graphs: annotation ambiguity from multiple candidates per equivalence class, context-dependent selection caused by depth-2 pattern templates, and DAG sharing conflict among parent references. Fully general extraction with arbitrary non-decomposable cost functions is NP-hard (Tate, Stepp, Tatlock & Lerner, 2009); however, practical pattern templates have bounded depth—typically at most two—and this structural restriction admits a specialized, exact, polynomial-time algorithm.

To exploit this observation, we propose an extraction algorithm based on tree DP with three mutually exclusive tile-role states—independent, tile-root, and tile-leaf—that jointly capture the coupling between candidate selection and tile assignment. The algorithm rests on the insight that the extraction output is a tree: each reference to a shared equivalence class corresponds to an independent tree position resolvable with its own parent context. A bottom-up pass computes structural compatibility signatures and optimal DP values for each state; a top-down pass traces back the optimal decisions to produce the output tree. We prove that for template sets with depth at most two, this yields an exact optimum in $O(N \cdot K \cdot |\mathcal{P}| \cdot C_{\max})$ time. The main contributions are:

1. We formalize pattern-aware extraction from e-graphs as a weighted pattern cover problem on AND-OR DAGs and establish its structural correspondence to tree covering in compiler instruction selection.
2. We identify three challenges—OR-node annotation ambiguity, context-dependent selection, and DAG sharing conflict—that arise when extending tree covering to e-graphs, and demonstrate how DAG-to-tree unfolding resolves them.
3. We show that the coupled selection-tiling problem reduces to a tree DP with three mutually exclusive tile-role states, and design an exact algorithm generalizing BURS tree covering from fixed trees to AND-OR DAGs in $O(N \cdot K \cdot |\mathcal{P}| \cdot C_{\max})$ time.
4. We present experiments comparing the proposed algorithm against standard egg extraction on weighted pattern coverage, exactness, and runtime.

2. Preliminaries and Problem Formulation

2.1. E-Graphs and Extraction

An *e-graph* $\mathcal{G} = (\mathcal{C}, \mathcal{N})$ consists of a finite set of *equivalence classes* (e-classes) $\mathcal{C}$ and a finite set of *e-nodes* $\mathcal{N}$ (Willsey, Nandi, Wang, Flatt, Tatlock & Panchekha, 2021; Nelson & Oppen, 1980). Each e-node $n \in \mathcal{N}$ has the form $\mathrm{Op}(c_1, \ldots, c_k)$, where Op is an operator symbol with arity $k \geq 0$ and $c_1, \ldots, c_k \in \mathcal{C}$ are its child e-classes. Every e-node belongs to exactly one e-class, and all e-nodes within the same e-class represent expressions that have been proven equivalent by the rewrite rules applied during saturation. Each e-class additionally carries *analysis data*—a summary of semantic properties (such as constant values, bit-widths, or variable occurrences) computed by a lattice-based fixed-point procedure over the e-graph (Willsey, Nandi, Wang, Flatt, Tatlock & Panchekha, 2021).

An e-graph admits a natural interpretation as an AND-OR directed acyclic graph. Each e-class acts as an *OR node*—exactly one of its member e-nodes must be selected—while each e-node acts as an *AND node*—once selected, all of its child e-classes must be recursively resolved. *Extraction* is the process of traversing this AND-OR DAG to produce a concrete expression tree. In standard extraction (Willsey, Nandi, Wang, Flatt, Tatlock & Panchekha, 2021), a *selection function* $\phi \colon \mathcal{C} \to \mathcal{N}$ maps each e-class c to a single representative $\phi(c) \in c$. The resulting output $T(\phi)$ is a tree obtained by recursively expanding from a designated root e-class. Because ϕ assigns a single e-node to each e-class *globally*, every reference to the same e-class in the DAG resolves to the same candidate.

 doi:10.20944/preprints202604.0583.v1

2.2. Pattern Templates

To formalize the notion of valuable multi-node structure that a downstream system can exploit, we introduce *pattern templates*.

Definition 1. *A* pattern template *is a four-tuple* $p = (S_p, \Pi_p, d_p, w_p)$ *where:*

- S_p *is a* structural schema*—a rooted operator tree of depth* d_p *in which each position is annotated with either a required operator type or a wildcard* $*$*;*
- Π_p *is a set of* predicate constraints *over the analysis data of the e-classes at the schema positions (e.g., "the second child is a constant shift amount," or "the sibling operand is a contiguous low-bit mask");*
- $d_p \in \{1, 2\}$ *is the* template depth*;*
- $w_p \in \mathbb{R}^+$ *is a positive* weight *reflecting the simplification value of the pattern.*

Template depth is the key parameter that governs algorithmic difficulty. A *depth-1* template constrains only a single e-node and its immediate children; its match status can be determined from the e-node and its children's analysis data alone. Structural primitives such as bit-field extraction (`Extract`) or zero-extension (`ZeroExtend`) are recognized as depth-1 templates, since the downstream system processes them directly without further transformation. A *depth-2* template additionally constrains the parent operator. The canonical example is $\texttt{Eq}\big(\texttt{BvAdd}(x, c),\ k\big)$ with joint predicates "`BvAdd`'s first operand is a variable" and "k is a constant." The downstream simplifier can then isolate the variable ($x = k - c$), triggering variable elimination. However, this applicability cannot be assessed without examining the parent–child pair together, since neither a standalone `BvAdd` nor a standalone `Eq` (whose arguments are not bare variables) satisfies any depth-1 template.

The weight w_p reflects how much simplification value a matched pattern contributes. Patterns that directly eliminate variables or collapse multiple operations into a single primitive receive higher weights, while patterns that merely improve encoding efficiency receive lower weights. For depth-2 templates, a strict super-additivity constraint is imposed: the joint weight satisfies $w_p > w_1^{\max}(\mathrm{Op}_{\mathrm{parent}}) + w_1^{\max}(\mathrm{Op}_{\mathrm{child}})$, where $w_1^{\max}(\mathrm{Op})$ denotes the best depth-1 weight achievable by operator type Op. This constraint reflects a structural property of downstream simplification systems such as SMT solver preprocessors (De Moura & Bjørner, 2008): the heuristic trigger patterns recognized by depth-2 templates—variable isolation, bit-field decomposition, and similar compound simplifications—yield simplification value that is inherently greater than the sum of their individual components, because the joint recognition enables a qualitatively different transformation (e.g., variable elimination) that neither component alone can trigger. Super-additivity also ensures that the three-state DP strictly prefers state $\downarrow$ whenever a depth-2 template applies, enabling the ablation experiment (Section 6) to cleanly isolate the incremental contribution of depth-2 tiling. The full set of pattern templates $\mathcal{P}$ is derived systematically from the simplification rules of the target downstream system; we assume $|\mathcal{P}|$ is a moderate constant in practice.

2.3. Optimization Objective

Given an e-graph $\mathcal{G}$ and a pattern template set $\mathcal{P}$, the extraction objective is to select a tree $T(\phi)$ that maximizes the total weighted pattern coverage. We formulate this objective as a tree covering problem, directly adopting the classical framework from compiler instruction selection (Blindell, 2016; Pelegri-Llopart & Graham, 1988; Aho, Ganapathi & Tjiang, 1989). Let $\mu(v, p)$ be a Boolean predicate that evaluates to *true* if and only if tree node v—together with its parent in $T(\phi)$ when $d_p = 2$—satisfies the structural schema S_p and all predicate constraints Π_p of template p.

A *covering* of a tree T partitions its nodes into non-overlapping *tiles*, each matching a template from $\mathcal{P}$. A *depth-1 tile* covers a single node v with value equal to the weight of the best matching depth-1 template, or zero if no template matches. A *depth-2 tile* covers a parent–child pair (u, v) with value w_p from the matching depth-2 template; this weight represents the *joint* simplification value of

 doi:10.20944/preprints202604.0583.v1

the two-node combination and subsumes both individual depth-1 contributions that the tile replaces. The *cover value* of a tree is the total weight under its optimal covering:

$$\text{CoverValue}(T) = \max_{\mathcal{T} \in \text{Cov}(T)} \sum_{t \in \mathcal{T}} w(t), \tag{1}$$

where $\text{Cov}(T)$ denotes the set of valid coverings of T and $w(t)$ is the weight of tile t. A structural constraint on the template set is *mutual exclusivity*: for any tree node v with determined parent context, at most one $p \in \mathcal{P}$ satisfies $\mu(v, p) = \textit{true}$. This is natural since the downstream simplification rules apply disjointly; any residual overlap is resolved by retaining only the highest-weight template for each schema. Because a depth-2 tile covers a parent–child pair, each parent node can participate in at most one depth-2 tile. When depth-2 templates apply at multiple child positions of the same parent, the algorithm selects the single most beneficial option (Section 4).

Under mutual exclusivity, the tile assignment at each tree node reduces to selecting one of three mutually exclusive roles: using a depth-1 tile independently, serving as root of a depth-2 tile with one child, or being subsumed as leaf of a parent's depth-2 tile. This three-way choice with local parent–child coupling admits a tree DP formulation (Section 4). The key difference from classical tree covering is that our problem must simultaneously *select* the tree from the AND-OR DAG and *optimize* its covering—the tree itself is a decision variable, introducing the challenges analyzed in Section 3.

3. Challenges: Beyond Classical Tree Covering

Classical tree covering (Pelegri-Llopart & Graham, 1988; Aho, Ganapathi & Tjiang, 1989) operates on a fixed expression tree where every node has a predetermined operator, enabling straightforward bottom-up annotation and top-down selection. The AND-OR DAG structure of an e-graph violates this premise in three ways.

3.1. OR-Node Annotation Ambiguity

In a fixed expression tree, each node carries a single operator, and the set of patterns applicable at that node is deterministic. In an e-graph, each e-class (OR node) contains multiple semantically equivalent e-nodes whose operator types may differ entirely. When a bottom-up pass reaches an e-class c, there is no single operator to annotate: one e-node in c may be $\texttt{BvAdd}(c_1, c_2)$, another $\texttt{BvOr}(c_1, c_2)$ (semantically equivalent when the operands have non-overlapping bit ranges). Each exposes a different operator to any parent referencing c—the former can participate in variable-isolation templates under an equality parent, while the latter cannot. Consequently, the pattern-match status of c is not a fixed attribute but a function of the *selection decision* $\phi(c)$—a variable that has not yet been determined at annotation time. A bottom-up pass must therefore retain annotation information for *every* candidate in every e-class rather than committing to a single best choice, multiplying the state space relative to classical tree covering.

3.2. Context-Dependent Selection

The ambiguity described above could, in principle, be resolved by a single bottom-up pass that evaluates all candidates and selects the one maximizing the depth-1 contribution. Depth-2 pattern templates defeat this strategy by coupling the child's value to the parent's identity.

Consider an e-class c containing two candidates: $n_1 = \texttt{BvAdd}(c_x, c_{\mathit{off}})$ and $n_2 = \texttt{BvOr}(c_x, c_{\mathit{off}})$, both semantically equivalent because the operands occupy non-overlapping bit ranges. Evaluating depth-1 templates alone, n_1 receives $w_1(n_1) = 0$ (addition does not match any depth-1 template), while n_2 matches a depth-1 bitwise-OR simplification template with $w_1(n_2) = w_1 > 0$—bitwise OR avoids the carry-chain propagation inherent in addition, making it the preferred operand form; a single bottom-up pass would therefore commit to n_2. Now suppose a parent e-node $n_{\text{par}} = \texttt{Eq}(c, c_k)$ exists, where c_k holds a constant and neither argument of the equality is a bare variable ($w_1(n_{\text{par}}) = 0$ at depth-1). If c selects $n_1 = \texttt{BvAdd}$ instead of the depth-1-optimal n_2, the parent–child pair jointly satisfies the

 doi:10.20944/preprints202604.0583.v1

depth-2 variable-isolation template described in Section 2.2: the downstream simplifier recognizes $\texttt{Eq}(\texttt{BvAdd}(x, c_{off}), k)$ and derives $x = k - c_{off}$, triggering variable elimination with weight $w_p > w_1$. Under a different parent—say $\texttt{BvUle}(c, c')$—no depth-2 template applies, and the depth-1 preference for n_2 remains optimal.

The crux is that a bottom-up pass processes c before any parent information is available, yet the optimal selection depends on the parent operator—breaking the optimal substructure that single-pass dynamic programming requires. Neither a pure bottom-up pass (lacking parent context) nor a pure top-down pass (lacking subtree costs) can resolve this dependency alone.

This context dependence is a structural limitation of decomposable, single-node cost functions, which satisfy three invariants: (i) *decomposability*—cost decomposes additively over individual nodes; (ii) *context independence*—evaluation does not depend on the parent operator; and (iii) *global commitment*—each equivalence class receives a single selection reused at every reference site. Invariant (ii) prevents encoding the joint value of a parent–child pair, and invariant (iii) prevents executing different selections at different tree positions. As Figure 1 illustrates, the two candidates in c have identical AST size, and a depth-1 evaluation globally prefers n_2 ($w_1 > 0$); yet only `BvAdd` enables the depth-2 match under `Eq`—neither a decomposable cost function nor a depth-1-only evaluation captures this position-specific opportunity.

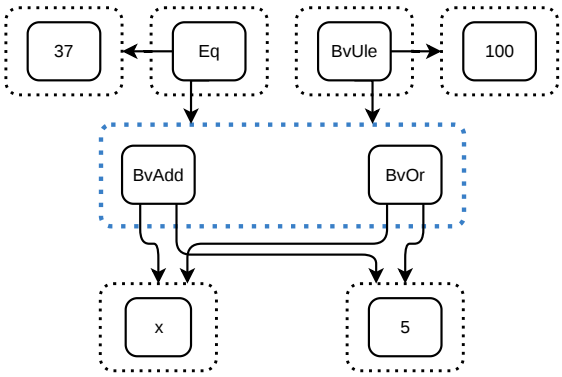

Figure 1. Running example illustrating context-dependent selection. E-class c contains two equivalent candidates $n_1 = \texttt{BvAdd}(x, 5)$ and $n_2 = \texttt{BvOr}(x, 5)$. The dashed blue region marks the depth-2 template linking `Eq` with `BvAdd`; under the `BvUle` parent no depth-2 template applies, and n_2 is preferred for its carry-free depth-1 match. A decomposable cost function must commit to one global selection; the proposed algorithm resolves each tree position independently.

3.3. DAG Sharing Conflict

Context-dependent selection is further complicated by DAG sharing—a characteristic that renders optimal covering on shared DAGs NP-complete (Blindell, 2016; Koes & Goldstein, 2008). A single e-class may be referenced by multiple parents imposing different depth-2 requirements. As in Figure 1, $\texttt{Eq}(c, c_k)$ prefers c to select `BvAdd` for variable isolation, while $\texttt{BvUle}(c, c')$ prefers `BvOr`. Under global selection, optimizing for one parent context necessarily sacrifices the other.

These three challenges are absent from classical tree covering (Pelegri-Llopart & Graham, 1988; Aho, Ganapathi & Tjiang, 1989) and collectively preclude direct migration of instruction-selection algorithms. In Section 4, we show that the tree-shaped nature of the extraction output resolves these difficulties: each reference to a shared e-class corresponds to an independent tree position resolvable with its own parent context.

 doi:10.20944/preprints202604.0583.v1

4. Extraction Algorithm

4.1. Key Insight: Extraction Output Is a Tree

The extraction output is a concrete expression tree: when the same e-class is referenced at multiple positions, each reference corresponds to an independent tree node. Because all candidates within an e-class are semantically equivalent, different positions may select different candidates without affecting correctness. Unfolding the AND-OR DAG into the output tree ensures that every node has exactly one parent, making the tile-role assignment at each position unambiguous—mirroring the standard DAG-to-tree unfolding in compiler instruction selection (Blindell, 2016; Koes & Goldstein, 2008).

This insight enables a two-pass solution: a bottom-up pass computes optimal subtree values under every possible tile-role assignment, and a top-down pass traces back these pre-computed values to construct the output tree. Table 1 summarizes the design rationale.

Table 1. Each algorithmic mechanism resolves a specific structural limitation of decomposable cost functions.

Limitation	Resolution	Algorithm component
Context independence	Three-state tile-role DP	$V^{\uparrow}$ separates committed value
Global commitment	Position-dependent selection	DAG-to-tree unfolding

4.2. Three-State Tree-Covering DP

Tile-role states. The coupling between candidate selection and tile assignment (Section 3) is resolved by decomposing each node's role into three mutually exclusive states:

- **State $-$ (independent)**: the node forms a depth-1 tile with weight $w_1(n)$; all children are free (state $-$ or $\downarrow$).
- **State $\downarrow$ (tile-root)**: the node forms a depth-2 tile with one selected child; the chosen child enters state $\uparrow$, all others remain free.
- **State $\uparrow$ (tile-leaf)**: the node is committed by its parent's depth-2 tile and forms no tile of its own; all children are free.

A node cannot simultaneously be tile-root and tile-leaf, as this would place it in two overlapping tiles. The assignment of state $\uparrow$ is decided by the *parent*: the parent evaluates whether committing the child improves the total covering value. This asymmetry resolves depth-2 context dependence (Section 3.2) without violating optimal substructure. The three states with parent–child coupling ($\downarrow\leftrightarrow\uparrow$) form a tree DP with constrained state assignment—a well-studied problem class encompassing maximum weighted matching and maximum independent set on trees. Each state's optimal value depends only on the subtree below the node; the parent's influence is limited to *which state* the node enters.

Template compilation and structural information. Before the DP begins, the template set $\mathcal{P}$ is compiled offline into two hash-indexed lookup tables: $\mathcal{T}_1[\mathrm{Op}]$ maps each operator to its depth-1 templates, and $\mathcal{T}_2[\mathrm{Op}_p, \mathrm{pos}, \mathrm{Op}_c]$ maps each (parent operator, child position, child operator) triple to its depth-2 templates. Both support $O(1)$ lookup; a single key may index multiple templates differing only in predicate constraints Π_p.

For each e-node $n = \mathrm{Op}(c_1, \ldots, c_k)$, the bottom-up pass first computes three structural quantities. The *depth-1 match weight* is:

$$w_1(n) = \begin{cases} \max\limits_{\substack{p \in \mathcal{T}_1[\mathrm{Op}] \\ \Pi_p \text{ satisfied}}} w_p & \text{if some depth-1 template matches,} \\ 0 & \text{otherwise.} \end{cases} \tag{2}$$

The *predicate profile* $\pi(n)$ is a bitvector summarizing child properties: $\pi(n)[i] = 1$ if child e-class c_i contains only constants or variables. The *compatibility signature* $\sigma(n)$ is a bitvector over operator types, where $\sigma(n)[f_p] = 1$ iff there exists a position i and a template $p \in \mathcal{T}_2[f_p, i, \mathrm{Op}]$ whose predicates Π_p are satisfied. Informally, $\sigma(n)$ encodes which parent operators could form a depth-2 tile with n as

 doi:10.20944/preprints202604.0583.v1

tile-leaf. Because the input space (Op, π) is finite, the mapping $(\mathrm{Op}, \pi) \to \sigma$ can be pre-compiled into a static lookup table—analogous to BURS automaton state compilation (Pelegri-Llopart & Graham, 1988; Proebsting, 1995)—yielding $O(1)$ per e-node. The per-e-class aggregate $\sigma^{\cup}(c) = \bigvee_{n \in c} \sigma(n)$ enables pruning: if $\sigma^{\cup}(c)[\mathrm{Op}] = 0$, no candidate in c can serve as tile-leaf for parent operator Op, and the depth-2 evaluation for that child position is skipped entirely.

Bottom-up DP values. The bottom-up pass requires an acyclic e-graph; cyclic e-nodes are removed by a standard preprocessing step (Willsey, Nandi, Wang, Flatt, Tatlock & Panchekha, 2021). For each e-node $n = \mathrm{Op}(c_1, \ldots, c_k)$ in topological order, after computing structural information, three DP values are computed. In state $\uparrow$ (tile-leaf), n is committed by its parent and forms no tile:

$$V^{\uparrow}(n) = \sum_{i=1}^{k} V^{*}(c_i). \tag{3}$$

In state $-$ (independent), n uses its depth-1 tile:

$$V^{-}(n) = w_1(n) + V^{\uparrow}(n). \tag{4}$$

In state $\downarrow$ (tile-root), n selects child position j, template p, and candidate $n'_j \in c_j$ to form a depth-2 tile; n'_j enters state $\uparrow$, other children are free:

$$V^{\downarrow}(n) = \max_{\substack{j,\, n'_j \in c_j,\, p \in \mathcal{T}_2 \\ \Pi_p \text{ satisfied}}} \Big(w_p + V^{\uparrow}(n'_j) + \textstyle\sum_{i \neq j} V^{*}(c_i) \Big). \tag{5}$$

If no depth-2 template matches, $V^{\downarrow}(n) = -\infty$. After all e-nodes are processed, each e-class aggregates the best free value:

$$V^{*}(c) = \max_{n \in c} \max\big(V^{-}(n),\, V^{\downarrow}(n)\big). \tag{6}$$

$V^{*}(c)$ does not bind a specific candidate; candidate selection is deferred to trace-back. The complete bottom-up procedure is given in Algorithm 1.

Algorithm 1: Bottom-Up DP Pass

Input: E-graph $\mathcal{G}$ (acyclic); template set $\mathcal{P}$
Output: $w_1(n)$, $\sigma(n)$, $V^{-}(n)$, $V^{\downarrow}(n)$, $V^{\uparrow}(n)$ for all n;; $V^{*}(c)$ for all c

$\mathcal{T}_1,\ \mathcal{T}_2 \leftarrow \mathrm{CompileTemplates}(\mathcal{P})$;
for all *e-node* $n = \mathrm{Op}(c_1, \ldots, c_k)$ *in topological order* **do**
  Compute $w_1(n)$ via Eq. (2); $\pi(n)$; $\sigma(n)$;
  $V^{\uparrow}(n) \leftarrow \sum_{i=1}^{k} V^{*}(c_i)$;
  $V^{-}(n) \leftarrow w_1(n) + V^{\uparrow}(n)$;
  $V^{\downarrow}(n) \leftarrow -\infty$;
  for $j \leftarrow 1$ **to** k **do**
    if $\sigma^{\cup}(c_j)[\mathrm{Op}] = 0$ **then continue**;
    for all $p \in \mathcal{T}_2[\mathrm{Op}, j, *]$ **do**
      for all $n' \in c_j$ *with* $\mathrm{Op}(n')$ *matching* p *and* Π_p *satisfied* **do**
        $v \leftarrow w_p + V^{\uparrow}(n') + \sum_{i \neq j} V^{*}(c_i)$;
        if $v > V^{\downarrow}(n)$ **then** $V^{\downarrow}(n) \leftarrow v$;;
        record (j, p, n');
  end
for all *e-class* $c \in \mathcal{G}$ **do**
  $V^{*}(c) \leftarrow \max_{n \in c} \max\big(V^{-}(n),\, V^{\downarrow}(n)\big)$;

 doi:10.20944/preprints202604.0583.v1

Top-down trace-back. Starting from the root e-class, the trace-back recursively selects candidates and assigns tile roles using pre-computed DP values. When an e-class c is *free* (not committed by its parent), the procedure selects the candidate n^* achieving $V^*(c)$ and compares $V^{\downarrow}(n^*)$ against $V^{-}(n^*)$. If $V^{\downarrow}(n^*) > V^{-}(n^*)$, n^* enters state $\downarrow$: it forms a depth-2 tile with the recorded best child n'_j, which enters state $\uparrow$, while all other children are recursed as free. Otherwise, n^* enters state $-$ and all children are free. When c is *committed* with a specific candidate n_c, that candidate enters state $\uparrow$: no tile is formed and all children are recursed as free.

Recursive calls carry no visit markers: each tree position makes an independent decision with its own parent context, resolving sharing conflicts through DAG-to-tree unfolding (Section 3.3). The complete procedure is given in Algorithm 2.

Algorithm 2: Top-Down Pass: Trace-Back

Input: E-graph $\mathcal{G}$;;
results of Algorithm 1
Output: Output expression tree

Function `Extract`(c, state):
  /* `Entry: Extract`(c_{root}, FREE) */
  if state $=$ FREE **then**
    $n^* \leftarrow \arg\max_{n \in c}\ \max\big(V^{-}(n),\ V^{\downarrow}(n)\big)$;
    if $V^{\downarrow}(n^*) > V^{-}(n^*)$ **then**
      $(j,\ p,\ n'_j) \leftarrow$ recorded best-$\downarrow$ for n^*;
      for all $(i,\ c_i) \in \text{children}(n^*)$ **do**
        if $i = j$ **then** `Extract`(c_i, COMMITTED(n'_j));
        else `Extract`(c_i, FREE);
      emit depth-2 tile p covering $(n^*,\ n'_j)$;
    else
      for all $(i,\ c_i) \in \text{children}(n^*)$ **do**
        `Extract`(c_i, FREE);
      if $w_1(n^*) > 0$ **then** emit depth-1 tile for n^*;
    end
  else /* state $=$ COMMITTED(n_c) */
    for all $(i,\ c_i) \in \text{children}(n_c)$ **do**
      `Extract`(c_i, FREE);
  end
  return TreeNode(...);

4.3. Illustrative Example

We revisit the running example from Section 3 (Figure 2a). E-class c contains $n_1 = \texttt{BvAdd}(c_x,\ c_{off})$ and $n_2 = \texttt{BvOr}(c_x,\ c_{off})$. Candidate n_1 does not match any depth-1 template, so $w_1(n_1) = 0$; candidate n_2 matches a depth-1 bitwise-OR simplification template with $w_1(n_2) = w_1 > 0$. Denoting $S = V^*(c_x) + V^*(c_{off})$, the bottom-up pass yields $V^{\uparrow}(n_1) = V^{\uparrow}(n_2) = S$, $V^{-}(n_1) = S$, and $V^{-}(n_2) = w_1 + S$. Neither serves as tile-root, so $V^{\downarrow}(n_1) = V^{\downarrow}(n_2) = -\infty$ and $V^*(c) = w_1 + S$, with n_2 as the best free candidate.

The depth-2 variable-isolation template p fires when an `Eq` parent pairs with a `BvAdd` child, so $\sigma(n_1)[\texttt{Eq}] = 1$ while $\sigma(n_2)[\texttt{Eq}] = 0$.

At Position Eq (Figure 2b, left), the parent $n_{\text{par}} = \texttt{Eq}(c,\ c_k)$ computes $V^{\downarrow}(n_{\text{par}}) = w_p + V^{\uparrow}(n_1) + V^*(c_k) = w_p + S + V^*(c_k)$ and $V^{-}(n_{\text{par}}) = V^*(c) + V^*(c_k) = w_1 + S + V^*(c_k)$ (since $w_1(n_{\text{par}}) = 0$). Since $V^{\downarrow} - V^{-} = w_p - w_1 > 0$, the trace-back selects state $\downarrow$, committing c to candidate n_1 in state $\uparrow$: the depth-2 gain w_p outweighs the forgone depth-1 weight w_1 of n_2.

 doi:10.20944/preprints202604.0583.v1

At Position BvUle (Figure 2b, right), the parent $n_{\text{par}'} = \texttt{BvUle}(c, c')$ finds no applicable depth-2 template ($V^{\downarrow} = -\infty$), so it selects state $-$, leaving c free. The trace-back selects $n_2 = \texttt{BvOr}$, which contributes its depth-1 weight w_1. The two decisions are independent: Position Eq selects `BvAdd` via state $\downarrow$ for the depth-2 match, while Position BvUle retains `BvOr` via state $-$ for the depth-1 match. This outcome arises from DAG-to-tree unfolding, where each tree position is a separate subproblem with its own parent context.

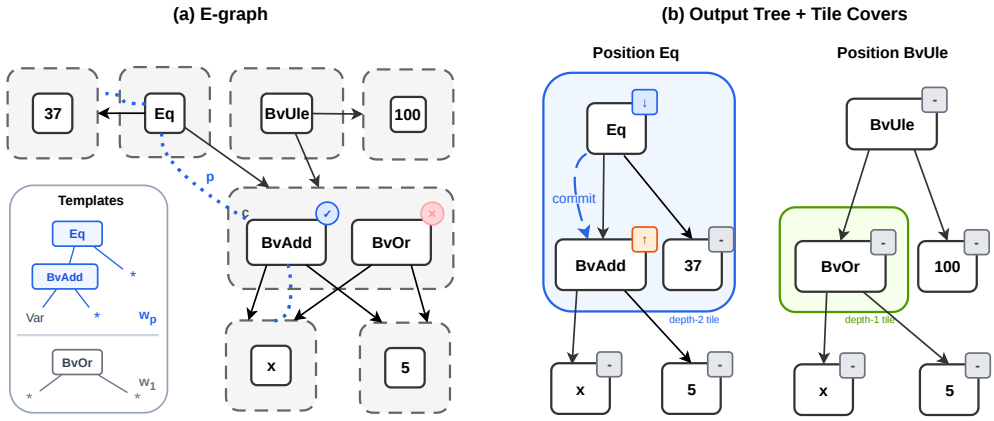

Figure 2. Algorithm walkthrough on the running example. (a) E-graph with template annotations: depth-2 template p (`Eq`–`BvAdd`) with weight w_p and depth-1 template for `BvOr` with weight w_1. (b) Output trees at two positions: Position Eq applies the depth-2 tile (blue), committing c to `BvAdd`; Position BvUle applies the depth-1 tile on `BvOr` (green).

5. Theoretical Analysis

This section establishes the two properties promised in Section 4: exactness of the optimum for template depth at most two, and polynomial time complexity.

5.1. Exactness for Bounded-Depth Templates

Proposition 1. *For any e-graph $\mathcal{G}$ and pattern template set $\mathcal{P}$ satisfying mutual exclusivity (Section 2.3) with* $\max_{p\in\mathcal{P}} d_p \leq 2$, *the output tree T produced by Algorithm 2 maximizes* $\text{CoverValue}(T)$ *as defined in Equation (1).*

Proof. The argument establishes that the three-state DP computes an optimal tile covering on the output tree.

Step 1 (Exhaustive state coverage). Every output-tree node is assigned exactly one of the three states $\{-, \downarrow, \uparrow\}$. The DP computes the optimal subtree value under each state (Equations (3)–(5)), and the trace-back (Algorithm 2) selects the best.

Step 2 (Mutual exclusivity). When n enters state $\downarrow$ with child n'_j, the recurrence (Equation (5)) uses $V^{\uparrow}(n'_j)$, which excludes any tile for n'_j itself. Conversely, when n is committed to state $\uparrow$ by its parent, $V^{\uparrow}(n)$ excludes any tile-root role for n. The $\downarrow\leftrightarrow\uparrow$ coupling thus correctly enforces non-overlapping tiles.

Step 3 (Optimal substructure). Processing in topological order ensures that all $V^{*}(c_i)$ values are finalized before any e-node referencing c_i is processed. Each state's value depends only on the subtree below n—the parent's identity does not appear in Equations (3)–(5). The parent determines which state n enters, but not the optimal value within each state.

Step 4 (Trace-back exactness). Algorithm 2 selects at each e-class the candidate and state achieving the pre-computed optimum. DAG unfolding ensures that each tree position is an independent subproblem with its own parent context. No heuristic is involved. □

 doi:10.20944/preprints202604.0583.v1

The depth bound $d_p \leq 2$ is essential: depth-3 templates would couple a node's matching status to both its parent and grandparent, introducing inter-position dependencies that the three-state formulation cannot capture. This restriction aligns with observed practice: in compiler instruction selection, depth-two patterns constitute the dominant category (Blindell, 2016; Aho, Ganapathi & Tjiang, 1989), and in SMT preprocessing no template in our evaluation exceeds depth two.

5.2. Time Complexity

Let N denote the number of e-nodes, K the maximum arity, $|\mathcal{P}|$ the template set size, $C_{\max}$ the maximum e-class size, and M the output tree size. Table 2 summarizes the cost.

Table 2. Time complexity of each phase. N: e-nodes; K: max arity; M: output tree size; $C_{\max}$: max e-class size.

Phase	Domain	Complexity	Notes
Template compilation	Offline	$O(\|\mathcal{P}\|)$	One-time
Bottom-up pass	E-graph	$O(N \cdot K \cdot \|\mathcal{P}\| \cdot C_{\max})$	Structural info + DP
Top-down pass	Output tree	$O(M \cdot K)$	Read decisions

The bottom-up pass visits each e-node once. For each e-node n, computing $w_1(n)$ and $\sigma(n)$ inspects templates in $\mathcal{T}_1$ and $\mathcal{T}_2$. Computing $V^{\uparrow}(n)$ and $V^{-}(n)$ costs $O(K)$. Computing $V^{\downarrow}(n)$ iterates over child positions (K), applicable templates ($|\mathcal{P}|$), and candidates within each child e-class ($C_{\max}$), yielding $O(K \cdot |\mathcal{P}| \cdot C_{\max})$ per e-node. Standard egg extraction runs in $O(N \cdot K)$ (Willsey, Nandi, Wang, Flatt, Tatlock & Panchekha, 2021); the overhead is a factor of $|\mathcal{P}| \cdot C_{\max}$, both moderate constants in practice.

The top-down pass visits each of the M output-tree nodes once, reading the pre-computed decision at $O(K)$ cost per node. No template evaluation occurs during trace-back. The combined worst-case complexity is $O(N \cdot K \cdot |\mathcal{P}| \cdot C_{\max} + M \cdot K)$, dominated by the bottom-up pass. When $C_{\max}$ is bounded, this simplifies to $O(N \cdot K \cdot |\mathcal{P}|)$.

6. Experiments

6.1. Experimental Setup

The evaluation targets the extraction algorithm in isolation—its coverage quality, correctness, and runtime—rather than end-to-end application performance. All experiments are implemented in Rust atop egg 0.11 (Willsey, Nandi, Wang, Flatt, Tatlock & Panchekha, 2021); the three extraction strategies share the same e-graph representation and template infrastructure so that observed differences are attributable solely to the extraction logic. The DP computation uses fixed-point iteration, equivalent to topological-order processing, which naturally handles cycles arising from identity-absorbing rewrites.

Benchmarks. Two benchmark categories are used. *SMT-COMP verification formulas* (21 instances) are drawn from the QF_BV and QF_ABV divisions of the 2024 SMT-COMP library (Barrett, Fontaine & Tinelli, 2017), originating from hardware verification workloads (PicoRV32, VexRiscv, ZipCPU, arbiter). Each formula is saturated with an iteration budget of 10 and a node limit of 10^5; resulting e-graphs range from 10^4 to 2.4×10^5 e-classes with $C_{\max} = 2$–4. Of these, 13 instances have operator compositions intersecting the template set (CoverValue > 0); Table 3 reports the 10 distinct coverage profiles (three further instances from the same families yield identical ratios and are omitted). The remaining 8 instances contain exclusively non-template operators. *Synthetic benchmarks* (3 instances, 15–80 assertions) embed known depth-2 patterns (mask-shift, high-mask equality, addition-equality isolation) alongside depth-1 patterns in QF_BV formulas; after saturation they contain 88–378 e-classes. A third set of *micro instances* (29–35 e-classes) supports brute-force correctness verification (Section 6.3).

Templates and strategies. The template set comprises three depth-1 templates (weights 3–5) and three depth-2 templates (weights 12–14) satisfying the super-additivity constraint of Section 2.2; $|\mathcal{P}| = 6$. Three strategies are compared: *standard extraction* (AST-size minimization (Willsey, Nandi, Wang, Flatt, Tatlock & Panchekha, 2021)), *depth-1 only* (Algorithm 1 with $V^{\downarrow}$ disabled), and *full three-*

 doi:10.20944/preprints202604.0583.v1

state (Algorithms 1–2). ILP (Tate, Stepp, Tatlock & Lerner, 2009) is excluded for the structural reasons analyzed in Section 7.

6.2. Extraction Quality

Pattern awareness vs. standard extraction. Table 3 compares CoverValue(T) across the three strategies. On the VexRiscv and bv-format benchmark families, pattern-aware extraction (both depth-1-only and full three-state) yields 16×–31× higher CoverValue than standard AST-size extraction. On PicoRV32 instances, the dominant operators do not intersect any template, and all strategies yield identical CoverValue. These results confirm that pattern awareness captures optimization value inaccessible to decomposable size-based cost functions, with the magnitude scaling with the density of template-matchable operators in the workload.

Depth-2 ablation. Table 4 isolates the incremental contribution of the $V^{\downarrow}$ mechanism on the synthetic benchmarks, which are designed with sufficient depth-2 pattern density to exercise the three-state DP. The full strategy improves CoverValue by 45–51% over the depth-1-only baseline, with depth-2 tile hits increasing from 6–28 (incidental matches under depth-1 optimization) to 10–50 (intentional matches under the three-state DP). In the BURS instruction selection literature, tree covering algorithms are evaluated independently of the instruction set architecture (Blindell, 2016; Pelegri-Llopart & Graham, 1988; Aho, Ganapathi & Tjiang, 1989): correctness and complexity are algorithmic properties, while the coverage achieved on a given workload depends on the pattern vocabulary. Following the same separation, the synthetic benchmarks validate the $V^{\downarrow}$ mechanism itself; developing comprehensive depth-2 template vocabularies tuned to specific workloads is orthogonal to the algorithmic contribution and is left to future work.

Table 3. CoverValue on SMT-COMP benchmarks (pattern awareness). Instances sharing identical coverage ratios within a family are deduplicated; 8 further instances with exclusively non-template operators yield CoverValue = 0 across all strategies.

Family	Benchmark	$\lvert\mathcal{C}\rvert$	Standard	Pattern-aware	Ratio
VexRiscv	regch0-30-nomem	66 193	405	12 555	31×
	regch0-30-mem	56 242	105	3 255	31×
	regch0-20-nomem	44 933	405	8 505	21×
	regch0-20-mem	38 182	105	2 205	21×
	regch0-15-nomem	34 303	405	6 480	16×
	regch0-15-mem	29 152	105	1 680	16×
bv-format	hex	42 704	405	8 505	21×
	binary	42 704	405	8 505	21×
PicoRV32	check-mem	60 560	174	174	1×
	pcregs-mem	47 128	114	114	1×

Table 4. Depth-2 ablation on synthetic benchmarks. D1-only: depth-1 extraction with $V^{\downarrow}$ disabled; Full: three-state DP.

		D1-only		Full			
Instance	$\lvert\mathcal{C}\rvert$	CoverValue	D2	CoverValue	D2	Δ	Gain
synth_small	88	95	6	143	10	+48	50.5%
synth_medium	211	280	18	406	28	+126	45.0%
synth_large	378	488	28	712	50	+224	45.9%

6.3. Correctness Validation

A brute-force enumerator traverses all feasible extraction trees on the micro instances and computes the globally optimal CoverValue by exhaustive search. Table 5 reports the results: the three-state

 doi:10.20944/preprints202604.0583.v1

algorithm matches the brute-force optimum on all three instances, empirically confirming the exactness guarantee of Proposition 1 and validating the implementation.

Table 5. Brute-force correctness verification on micro instances.

Instance	$\|\mathcal{C}\|$	BF CoverValue	DP CoverValue	Match
micro_1	33	29	29	✓
micro_2	29	29	29	✓
micro_3	35	41	41	✓

6.4. Runtime Efficiency

Table 6 reports wall-clock extraction time. The full three-state strategy incurs 1.9–2.8× overhead relative to standard extraction on SMT-COMP benchmarks, with the fixed-point DP computation dominating (> 85% of total time). This is consistent with the theoretical $|\mathcal{P}| \cdot C_{\max}$ factor of Section 5.2: with $|\mathcal{P}| = 6$ and $C_{\max} \leq 4$, each child position inspects at most 24 candidate–template pairs, a modest cost amplified by the number of fixed-point iterations. On synthetic benchmarks, all strategies complete in under two milliseconds. The absolute time on the largest instance (2.7 s on `zipcpu-pfcache`, 2.4×10^5 e-classes, 500 roots) exceeds the saturation phase (89–265 ms on the same instances), reflecting the cost of multi-root formula processing; the DP phase itself is a one-time cost independent of the number of roots.

Table 6. Extraction time (ms). Sat.: equality saturation phase; Std.: standard AST-size extraction; D1: depth-1-only pattern-aware extraction; Full: full three-state extraction. $|\mathcal{C}|$: e-classes after saturation; R: assertion roots.

Source	Benchmark	$\|\mathcal{C}\|$	R	Sat.	Std.	D1	Full
Synth.	small	88	15	<1	<1	<1	<1
	medium	211	42	<1	<1	<1	<1
	large	378	80	2	1	1	1
SMT-COMP	VexRiscv-30-nomem	66 193	160	265	416	716	819
	bv-format-hex	42 704	110	182	298	599	839
	picorv32-pcregs	47 128	139	93	425	624	792
	zipcpu-pfcache	236 290	500	89	1 233	2 014	2 735
	arbiter-b30	31 440	208	114	157	308	407

7. Related Work

The extraction problem in equality saturation has grown in importance as e-graphs have been applied to compilers (Joshi, Nelson & Randall, 2002; VanHattum, Nigam, Lee, Bornholt & Sampson, 2021), SMT solvers (Flatt, Coward, Willsey, Tatlock & Panchekha, 2022), hardware verification (Coward, Constantinides & Drane, 2023), and manufacturing (Nandi et al., 2020). The standard extraction algorithm implemented in the `egg` framework (Willsey, Nandi, Wang, Flatt, Tatlock & Panchekha, 2021) performs a single bottom-up traversal that greedily selects the candidate minimizing a decomposable, single-node cost function such as abstract syntax tree size. This approach is efficient and provably optimal when costs decompose additively over individual nodes, but it cannot express objectives that depend on multi-node structural context. Earlier, Joshi, Nelson and Randall (2002) used e-graphs in the Denali superoptimizer to generate optimal code via goal-directed search, demonstrating the potential of e-graphs for program optimization; however, Denali does not formalize an extraction-time objective. To handle more general, non-decomposable cost functions, Tate, Stepp, Tatlock and Lerner (2009) formulated extraction as an integer linear program (ILP), enabling global optimality at the expense of NP-hard worst-case complexity that limits scalability to large e-graphs. Our work is complementary: rather than targeting arbitrary cost functions with a general-purpose solver, we identify a structured class of objectives—bounded-depth weighted pattern coverage—and exploit this structure to obtain an exact, polynomial-time algorithm.

 doi:10.20944/preprints202604.0583.v1

Beyond computational cost, the ILP formulation exhibits a structural limitation for the weighted pattern cover objective. The ILP encoding of Tate, Stepp, Tatlock and Lerner (2009) assigns one binary variable per (e-class, e-node) pair, enforcing a global selection that cannot express position-dependent decisions when a shared e-class appears under parents with differing contextual requirements. The pattern cover objective is therefore *structurally incompatible* with the ILP formulation: the gap lies in the distinction between global selection and position-dependent selection, which DAG-to-tree unfolding (Section 4.1) resolves.

A closely related line of work originates from compiler instruction selection, where the *tree covering* problem seeks to partition a fixed expression tree into non-overlapping tiles matching instruction patterns at minimum total cost. Aho, Ganapathi and Tjiang (1989) formalized code generation as tree matching with dynamic programming, and Pelegri-Llopart and Graham (1988) established the BURS (Bottom-Up Rewrite System) framework with offline automaton compilation (Proebsting, 1995); both achieve optimal tree covering via bottom-up dynamic programming when the input is a fixed tree. However, extending tree covering from trees to DAGs—as required when expression DAGs contain shared subexpressions—renders the problem NP-complete (Blindell, 2016). Practical compilers such as GCC and LLVM therefore resort to unfolding DAGs into trees before applying instruction selection (Koes & Goldstein, 2008). Our algorithm adopts the same unfolding principle but applies it to the AND-OR DAG structure of e-graphs, where each e-class (OR node) introduces candidate selection on top of the standard tile assignment problem. This additional degree of freedom necessitates a three-state DP—the minimal extension of classical BURS that explicitly tracks the tile-leaf role arising from depth-2 mutual exclusivity. When the e-graph degenerates to a single-candidate-per-class tree, the algorithm reduces to standard BURS. To our knowledge, no prior work has migrated tree covering techniques to the e-graph extraction setting or formalized the weighted pattern cover objective on AND-OR DAGs.

8. Conclusions

This paper formalized pattern-aware extraction from e-graphs as a weighted pattern cover problem on AND-OR DAGs and identified three challenges—OR-node annotation ambiguity, context-dependent selection, and DAG sharing conflict—that distinguish this setting from classical tree covering. Exploiting the tree-shaped nature of the extraction output, we designed an algorithm based on three-state tree DP—generalizing BURS tree covering from fixed trees to AND-OR DAGs—that computes an exact optimum for bounded-depth templates in $O(N \cdot K \cdot |\mathcal{P}| \cdot C_{\max})$ time. Experiments on SMT-COMP hardware verification benchmarks demonstrate that pattern-aware extraction improves weighted coverage by up to 31× over standard AST-size extraction; an ablation study on synthetic benchmarks further confirms that the depth-2 tiling mechanism contributes an additional 45–51% improvement when depth-2 template opportunities are present, with extraction overhead remaining within a factor of 2–3× relative to standard extraction.

Several directions remain for future investigation. Extending the framework to templates of depth three or beyond would require additional DP states and a careful analysis of the resulting inter-position coupling. Integrating pattern-aware extraction with the saturation phase itself—for instance, by guiding rewrite rule application toward regions where high-value templates are likely to match—could further improve overall optimization quality. Finally, evaluation on a broader range of application domains, including compiler middle-end optimization (VanHattum, Nigam, Lee, Bornholt & Sampson, 2021) and hardware synthesis (Coward, Constantinides & Drane, 2023), would help characterize the generality of bounded-depth pattern coverage as an extraction objective.

Author Contributions: Conceptualization, Z.C. and M.Y.; methodology, Z.C.; software, Z.C.; formal analysis, Z.C.; writing—original draft preparation, Z.C.; writing—review and editing, M.Y. and L.Z.; supervision, M.Y. and L.Z. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

 doi:10.20944/preprints202604.0583.v1

Data Availability Statement: The source code and benchmark data will be made available upon publication.

Acknowledgments: The authors thank the anonymous reviewers for their constructive feedback.

Conflicts of Interest: The authors declare no conflicts of interest.

Abbreviations

The following abbreviations are used in this manuscript:

BURS	Bottom-Up Rewrite System
DAG	Directed Acyclic Graph
DP	Dynamic Programming
ILP	Integer Linear Programming
SMT	Satisfiability Modulo Theories

References

Willsey, M., Nandi, C., Wang, Y. R., Flatt, O., Tatlock, Z., & Panchekha, P. (2021). Egg: Fast and extensible equality saturation. *Proceedings of the ACM on Programming Languages*, *5*(POPL), 1–29.

Tate, R., Stepp, M., Tatlock, Z., & Lerner, S. (2009, January). Equality saturation: A new approach to optimization. In *Proceedings of the 36th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages* (pp. 264–276).

Blindell, H. (2016). *Instruction selection*. Springer International Publishing.

Pelegri-Llopart, E., & Graham, S. L. (1988, January). Optimal code generation for expression trees: An application of BURS theory. In *Proceedings of the 15th ACM SIGPLAN-SIGACT symposium on Principles of programming languages* (pp. 294–308).

Koes, D. R., & Goldstein, S. C. (2008, April). Near-optimal instruction selection on DAGs. In *Proceedings of the 6th annual IEEE/ACM international symposium on Code generation and optimization* (pp. 45–54).

De Moura, L., & Bjørner, N. (2008, March). Z3: An efficient SMT solver. In *International conference on Tools and Algorithms for the Construction and Analysis of Systems* (pp. 337–340). Berlin, Heidelberg: Springer Berlin Heidelberg.

Nelson, G., & Oppen, D. C. (1980). Fast decision procedures based on congruence closure. *Journal of the ACM (JACM)*, *27*(2), 356–364.

Aho, A. V., Ganapathi, M., & Tjiang, S. W. (1989). Code generation using tree matching and dynamic programming. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, *11*(4), 491–516.

Proebsting, T. A. (1995). BURS automata generation. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, *17*(3), 461–486.

Joshi, R., Nelson, G., & Randall, K. (2002). Denali: A goal-directed superoptimizer. *ACM SIGPLAN Notices*, *37*(5), 304–314.

Nandi, C., Willsey, M., Anderson, A., Wilcox, J. R., Darulova, E., Grossman, D., & Tatlock, Z. (2020, June). Synthesizing structured CAD models with equality saturation and inverse transformations. In *Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation* (pp. 31–44).

VanHattum, A., Nigam, R., Lee, V. T., Bornholt, J., & Sampson, A. (2021, April). Vectorization for digital signal processors via equality saturation. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems* (pp. 874–886).

Flatt, O., Coward, S., Willsey, M., Tatlock, Z., & Panchekha, P. (2022, October). Small proofs from congruence closure. In *2022 Formal Methods in Computer-Aided Design (FMCAD)* (pp. 75–83). IEEE.

Coward, S., Constantinides, G. A., & Drane, T. (2023, July). Automating constraint-aware datapath optimization using e-graphs. In *2023 60th ACM/IEEE Design Automation Conference (DAC)* (pp. 1–6). IEEE.

Barrett, C., Fontaine, P., & Tinelli, C. (2017). *The SMT-LIB Standard: Version 2.6* (Tech. Rep.). Department of Computer Science, The University of Iowa. Available at `www.SMT-LIB.org`.