Evolving Abstract Transformers for Gradient-Guided, Adaptable Abstract Interpretation

SHAURYA GOMBER, University of Illinois Urbana-Champaign, USA
DEBANGSHU BANERJEE, University of Illinois Urbana-Champaign, USA
GAGANDEEP SINGH, University of Illinois Urbana-Champaign, USA

Current numerical abstract interpretation relies on fixed, hand-crafted, instruction-specific transformers tailored to each domain, giving rise to three significant limitations. First, extensibility is limited because transformers cannot be reused across domains and new transformers need to be designed for each new domain or operator. Second, precise compositional reasoning over instruction sequences is difficult as transformers are defined only at the instruction level. Third, all downstream tasks are forced to use the same fixed transformer, irrespective of their precision, efficiency, or task-specific requirements. To address these limitations, we propose the *Evolving Abstract Transformer*, a general transformer that replaces the fixed single-output design of traditional transformers with an adaptable search over a parametric space of sound outputs. This is achieved through two underlying algorithms we develop. First, the *Universal Parametric Output Space Encoder (UPOSE)* constructs a compact parametric space of sound outputs for any polyhedral numerical domain and any operator in the *Quadratic-Bounded Guarded Operators (QGO)* class which includes both individual instructions and structured sequences. Next, the *Adaptive Gradient Guidance (AGG)* algorithm leverages the differentiable structure of the space generated by UPOSE and uses gradient-based updates to efficiently search it according to downstream analysis objectives and available runtime, continually *evolving* the output as more time is provided. We implement these ideas in the AbsEvolve framework and evaluate their effectiveness across three numerical abstract domains: Zones, Octagons, and Polyhedra. Our results demonstrate that the evolving transformer works across domains and handles diverse instructions and sequences, allowing efficient adaptability in the precision–efficiency tradeoff by adjusting the number of gradient steps in the search, while also reaching the most precise invariants up to 3.2× faster than existing baselines.

CCS Concepts: • **Theory of computation → Program analysis**; • **Software and its engineering → Automated static analysis**.

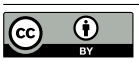
Additional Key Words and Phrases: Efficient and Precise Abstract Interpretation, Adaptable Analysis, Parametric Abstract Outputs, Gradient-Guided Optimization

ACM Reference Format:
Shaurya Gomber, Debangshu Banerjee, and Gagandeep Singh. 2026. Evolving Abstract Transformers for Gradient-Guided, Adaptable Abstract Interpretation. *Proc. ACM Program. Lang.* 10, PLDI, Article 268 (June 2026), 41 pages. https://doi.org/10.1145/3808346

1 Introduction

Abstract interpretation [11] is a widely used framework for static program analysis that soundly approximates a program's *concrete* semantics using an alternate representation called the *abstract domain*. Over the years, a variety of domains have been developed to capture different properties, such as heap structure [33, 45], control-flow behavior [4], and numeric relationships [14, 31]. This

Authors' Contact Information: Shaurya Gomber, University of Illinois Urbana-Champaign, Urbana, USA, sgomber2@illinois.edu; Debangshu Banerjee, University of Illinois Urbana-Champaign, Urbana, USA, db21@illinois.edu; Gagandeep Singh, University of Illinois Urbana-Champaign, Urbana, USA, ggnds@illinois.edu.



ACM 2475-1421/2026/6-ART268
https://doi.org/10.1145/3808346

work focuses on *numerical* domains, which are central to reasoning about variable ranges [48], arithmetic updates [31], and control-flow conditions [4] in real-world programs. Several numerical domains exist, such as Interval [12], Octagons [35] and Polyhedra [14], each varying in expressiveness and efficiency. Selecting the right domain involves the fundamental trade-off between precision and scalability. Once the domain is chosen, the analysis is performed through the domain's *abstract transformers*, which are functions that define how program operations (e.g., assignments) are soundly over-approximated within the domain.

Challenge 1: Per-Domain and Per-Instruction Design. Designing an abstract transformer is complicated, since it requires balancing efficiency and precision while ensuring soundness throughout. This must be carried out for all possible instructions across all abstract domains. Moreover, precision in abstract interpretation is *non-compositional*, i.e., analyzing an instruction sequence per-instruction and composing the results is *generally* less precise than analyzing the sequence jointly. Thus, in principle, achieving precise analysis would require designing separate abstract transformers for all instruction sequences. Designing such a large number of transformers manually is very tedious and has motivated efforts toward automating transformer construction [43]. Prior work such as [27, 28] uses program synthesis to generate transformers within a user-defined DSL for specific domains such as strings. However, such automation is not possible for general numerical domains like Octagons and Polyhedra, which involve reasoning about numerical relations that are more complex to handle. Consequently, current numerical analysis relies on libraries such as ELINA [48], APRON [25], and PPL [1], which support limited standard domains such as Octagons and Polyhedra by providing hand-crafted transformers for them. These libraries face two key limitations. First, many analysis scenarios require domains beyond the standard ones, such as TVPI [44], Octahedron [10], Logahedra [24] etc. Although these domains resemble existing ones like Octagons in being *polyhedral* and representing states via conjunctions of linear constraints, the abstract transformers must be redesigned for each domain, as the existing transformers are tied to their specific domains and cannot be reused. Second, because the transformers are manually designed, it is infeasible to construct separate transformers for all instruction sequences, so they are designed only at the instruction level. As a result, instruction sequences are analyzed by composing per-instruction transformers, which reduces precision.

Challenge 2: Imprecision & Lack of Adaptability. The loss of precision from composing transformers is compounded by the fact that current libraries often do not compute the most precise transformer even for simple operations. For instance, Template Constraint Matrix (TCM) domains [46] such as Octagons are widely used, as they are more efficient than expressive domains like Polyhedra, whose operations have exponential complexity. However, these domains cannot precisely capture even simple affine assignments such as $z = 2x + 3y$ that fall outside their constraint form. Current libraries worsen this imprecision by not even computing the most precise over-approximation within the domain, as it is computationally expensive and involves solving a large number of linear programs per transformer call. Moreover, domains like Polyhedra can represent linear assignments exactly but face similar limitations for non-linear assignments like $z = x * x$, whose precise analysis requires calls to non-linear solvers that are extremely slow and do not even guarantee soundness [22]. To remain efficient, existing libraries use ad hoc handcrafted over-approximations that are sound but often highly imprecise.

Furthermore, current libraries rely on *fixed* transformers that always prioritize efficiency over precision, *irrespective of the analysis task*. Different scenarios, however, demand different trade-offs: lightweight runtime checkers [33] require fast analyses, whereas safety-critical analyzers [13] can afford longer runtimes for higher precision. Prior works [34, 51] have shown the benefits of adapting analyzers to downstream goals by tuning general parameters that control their behavior,

while [9, 23] highlight the need and benefits of adapting analyses to time and memory constraints. Current abstract transformers, however, are not amenable to such tuning, as these transformers are rigid and always follow the same routine, producing a single fixed output (from among potentially infinite sound outputs) that cannot be tuned for precision or compute efficiency.

Key Idea: Evolving Abstract Transformer. To overcome the challenges discussed, we introduce *Evolving Abstract Transformer*, a new paradigm that unifies transformer design across various domains and instructions and replaces the fixed output design with an adaptable, search-based process. The design proceeds in two key phases. For a given input abstract element, our *Universal Parametric Output Space Encoder* (UPOSE) constructs a *symbolic, parametric* space of sound outputs, rather than computing a single fixed output. The *Adaptive Gradient Guidance* (AGG) algorithm then starts from an initial point in this space and efficiently traverses it through gradient-based updates, iteratively refining and *evolving* the outputs toward increasingly precise and always sound results. This novel search-based design enables analyses to adapt precision over time and converge toward more precise outputs within the available time budget. This design offers the following benefits:

(1) **Cross-Domain and Sequence Generality:** Unlike existing transformers, our evolving transformer is not tied to a specific domain or instruction. The UPOSE algorithm works for all Polyhedral domains (Section 2), from standard ones like Octagons, Polyhedra to domains like Octahedron, and handles a broad range of instructions, including affine, quadratic, and guarded assignments captured by the Quadratic-Bounded Guarded Operators (QGO) class (Section 4). The QGO class is highly expressive and also captures instruction sequences whose combined semantics lie in this class (Sec. 6). The UPOSE algorithm enables efficient abstraction of such sequences as a whole, outperforming SMT-based symbolic abstraction approaches such as [32].
(2) **Parametric Space for Adaptable Analysis**: Instead of computing a single sound output, the UPOSE algorithm synthesizes a *sound-by-construction*, *compact parametric* space of outputs, encoded using a small set of parameters. This design makes our evolving transformer flexible, allowing analyses to adaptively search within an always-sound output space to find results that best suit their needs. For instance, the space of sound outputs synthesized by UPOSE includes the precise outputs computed by solver-based transformers [32, 46], enabling analyses to achieve solver-level precision when such outputs are discovered by the search.
(3) **Efficient Gradient-Guided Search:** The effectiveness of our evolving transformer depends on the efficiency of its underlying search. This is addressed by the AGG algorithm, which exploits the fact that the parametric space produced by UPOSE is *differentiable*, and uses gradient information to efficiently guide the search and refine outputs within a given *runtime budget* $\mathcal{R}$ (number of gradient steps). This gradient-guided process enables continual refinement and efficient use of runtime toward achieving higher precision, allowing AGG to converge to the most precise outputs much faster than solver-based transformers [32, 46], as shown in Section 7.1. More generally, AGG can optimize any search objective $\mathcal{J}$ (score function) using its gradient $\nabla\mathcal{J}$, extending beyond precision to other analysis goals.

AbsEvolve framework. We implement these ideas in the AbsEvolve framework (Fig. 1). AbsEvolve performs abstract interpretation using Evolving Abstract Transformers, applying the UPOSE algorithm to encode a parametric space of sound outputs for the supported operators and the AGG algorithm to select suitable outputs during analysis. For instruction sequences whose combined effect lies within the QGO class, AbsEvolve automatically merges them and uses UPOSE to encode a joint parametric space for the entire block (Sec 6), reducing precision loss from composing per-instruction transformers. For unsupported operations, the framework falls back to baseline transformers (from Elina). Users can specify parameters such as the search budget $\mathcal{R}$ (number of gradient steps) to control the search process and adapt precision based on available resources.

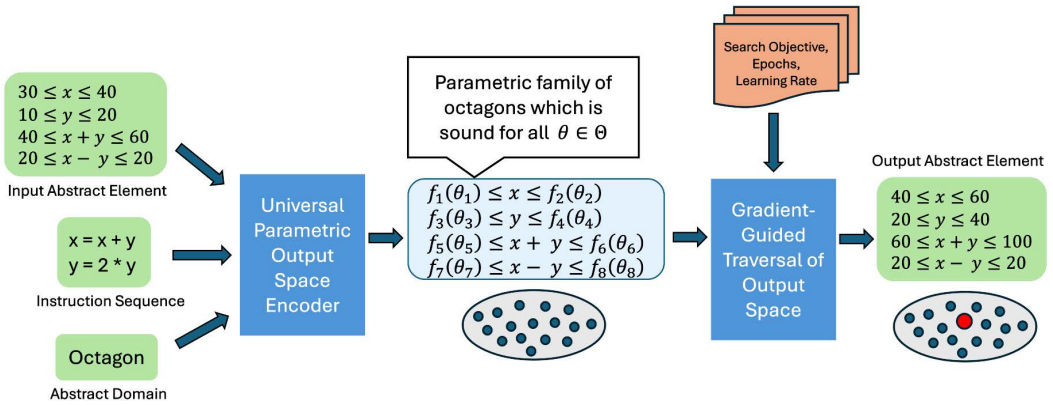


Fig. 1. For a given numerical domain, instruction sequence, and input abstract element, AbsEvolve first generates a parametric space of sound outputs for that sequence. Then, it performs gradient-guided search, driven by user-specified parameters, to efficiently select a suitable output during analysis.

AbsEvolve is the first precision–efficiency trade-off adaptable, sound-by-construction framework for abstract interpretation that combines formal soundness guarantees with the efficiency of gradient-guided optimization.

Main Contributions. This paper makes the following contributions:

- We introduce the *Evolving Abstract Transformer*, a new general transformer that rethinks the traditional design by constructing a parametric space of sound outputs rather than a single output. This is achieved by using our Universal Parametric Output Space Encoder (UPOSE) algorithm, which operates across polyhedral numerical domains and handles a wide range of instructions and sequences from the Quadratic-Bounded Guarded Operators (QGO) class (Section 4).
- To explore the space of outputs efficiently, we propose the novel Adaptive Gradient Guidance (AGG) algorithm, which takes a search objective $\mathcal{J}$ and runtime budget $\mathcal{R}$ as input, and efficiently navigates the output space using gradient signals from $\mathcal{J}$ to guide the search toward outputs that best align with the requirements of the downstream tasks (Section 5).
- We implement these algorithms in the AbsEvolve framework (Section 6) and evaluate them across three numerical domains: Zones, Octagons, and Polyhedra. Our results (Section 7) show that AbsEvolve enables compositional reasoning over instruction sequences and achieves significant, adaptable precision improvements over baselines through gradient-guided evolution, while also reaching the most precise invariants up to 3.2× faster than existing baselines.

2 Background

Abstract Interpretation. Abstract interpretation [11] over-approximates the behavior of a system represented by a *Concrete Domain* using an alternate *Abstract Domain*. The concrete domain $(C, \sqsubseteq_C)$ consists of concrete elements partially ordered by $\sqsubseteq_C$, while the abstract domain $(\mathcal{A}, \sqsubseteq_{\mathcal{A}})$ contains abstract elements used to *represent* elements in C. In program analysis, a *state* is a valuation over program variables $\mathcal{V}$, and the concrete domain C is the powerset of all program states, $\mathcal{P}(States)$, ordered by subset inclusion $\subseteq$. Each element $c \in C$ thus represents a set of possible program states. Abstract domains such as Intervals, Zones, or Polyhedra provide finite representations of such sets, enabling efficient analysis by over-approximating elements of C. For the rest of this paper, we assume the program analysis setting with $C = \mathcal{P}(States)$.

Concretization Function. The *concretization* function $\gamma : \mathcal{A} \to C$ is a monotonic function, which, for a given abstract element $\mathsf{a} \in \mathcal{A}$ computes the concrete element $\gamma(\mathsf{a}) \in C$ represented by a. $\gamma(\mathsf{a})$ captures all program states represented by a.

Definition 2.1 (Precision of Abstract Elements). *An abstract element $\mathsf{a}_1 \in \mathcal{A}$ is* more precise *than another element $\mathsf{a}_2 \in \mathcal{A}$ if $\gamma(\mathsf{a}_1) \subseteq \gamma(\mathsf{a}_2)$.*

Abstract Transformers. The concrete domain includes operations $op : C \to C$ that operate on concrete elements. In program analysis, these operations correspond to statements such as assignments that transform program states. To analyze the concrete domain within the abstract domain, we require, for each concrete operator op, a corresponding abstract operator $\hat{op} : \mathcal{A} \to \mathcal{A}$ that operates on abstract elements in a way that *soundly* reflects the effect of op on concrete elements. This function $\hat{op}$ is called the *abstract transformer* corresponding to the concrete operator op. Now, we define the soundness of abstract transformers, followed by the notion of precision between transformers and the definition of the most-precise transformer.

Definition 2.2 (Soundness of Abstract Transformer). *The abstract transformer $\hat{op} : \mathcal{A} \to \mathcal{A}$ for concrete operator op is sound if it over-approximates the effect of op, i.e. for all abstract elements $\mathsf{a} \in A$, the inclusion relation $op(\gamma(\mathsf{a})) \subseteq \gamma(\hat{op}(\mathsf{a}))$ holds.*

Definition 2.3 (Precision of Abstract Transformers). *Given two sound abstract transformers $\hat{op}_1, \hat{op}_2 : \mathcal{A} \to \mathcal{A}$ for the same concrete operator op, we say that $\hat{op}_1$ is* more precise *than $\hat{op}_2$ if it generates more precise outputs, i,e, $\forall \mathsf{a} \in \mathcal{A},\ \gamma(\hat{op}_1(\mathsf{a})) \subseteq \gamma(\hat{op}_2(\mathsf{a}))$.*

Definition 2.4 (Most-Precise Abstract Transformer). *An abstract transformer $\hat{op}^{\#} : \mathcal{A} \to \mathcal{A}$ is the* most-precise *(or* best*) transformer for a concrete operator $op : C \to C$ in domain $\mathcal{A}$ if it is sound and generates most-precise outputs, i.e, for every other sound abstract transformer $\hat{op}' : \mathcal{A} \to \mathcal{A}$ corresponding to op, $\forall \mathsf{a} \in \mathcal{A},\ \gamma(\hat{op}^{\#}(\mathsf{a})) \subseteq \gamma(\hat{op}'(\mathsf{a}))$ holds.*

Polyhedral Domains. Polyhedral domains are a widely used family of numerical domains that represent sets of program states using systems of linear inequalities. Given a finite set of program variables $\mathcal{V} = \{v_1, \ldots, v_n\}$, an abstract element is $\mathsf{a} = \{\mathbf{A}\mathbf{v} \leq \mathbf{b}\}$, where $\mathbf{v} = [v_1, \ldots, v_n]^\top$, $\mathbf{A} \in \mathbb{R}^{m \times n}$, and $\mathbf{b} \in \mathbb{R}^m$. This formulation subsumes a range of widely used domains: Intervals [12] capture independent bounds on variables, Zones [36] track differences between variables, Octagons [35] handle sums and differences, and general Polyhedra [14] support arbitrary linear relations.

Template Constraint Matrix (TCM) Domains. A common subclass of polyhedral domains is the family of *template-based domains*, which restricts abstract elements to a fixed set of linear directions. These directions are encoded by a *Template Constraint Matrix (TCM)* [46] $\mathcal{T} \in \mathbb{R}^{t \times n}$, where each row $\mathcal{T}_i$ specifies a linear constraint over the finite set of variables $\mathcal{V} = \{v_1, \ldots, v_n\}$. Letting $\mathbf{v} = [v_1, \ldots, v_n]^\top$, abstract elements take the form $\mathsf{a} = \{\mathcal{T}\mathbf{v} \geq \mathbf{c}\}$, where $\mathbf{c} \in (\mathbb{R} \cup \{-\infty\})^t$ gives lower bounds for each constraint. Any constraint with $c_i = -\infty$ is treated as vacuously true and omitted. TCM domains include domains like Interval, Zones, Octagons, etc.

3 Overview

In this section, we highlight the key limitations of the current abstract interpretation design and demonstrate how our *Evolving Abstract Transformer* helps overcome them.

3.1 Limitations of Current Design

Consider the program in Fig. 2a. Since $x = 0$ at start of the loop, $x \leq 0$ holds initially. In each iteration, x increases by y and then decreases by $y + 1$, effectively decreasing by 1. So, $x \leq 0$ holds always and is a valid loop invariant, implying that the assertion holds. However, when analyzed

```
1  int x = 0, y = 0;
2
3  while (y <= 10)
4  {
5     x = x + y;
6     y = y + 1;
7     x = x - y;
8  }
9
10 assert(x <= 0);
```

(a) Imprecision due to imprecise assignment transformers

```
1  int x = 30;
2  int y = 10;
3
4  while (y <= 80)
5  {
6     x = x + y;
7     y = 2*y;
8  }
9
10 assert(x - y <= 20);
```

(b) Imprecision due to statement-wise abstract interpretation

Fig. 2. Examples demonstrating imprecision of current libraries for octagon analysis.

with the state-of-the-art ELINA [48] library using the Octagon domain, the inferred invariant does not include $x \leq 0$ and assertion remains unproven, even though this constraint lies in the octagonal template. This is because the abstract transformers used for assignments in ELINA are *imprecise*. For instance, at Line 7, the following transition arises during the analysis:

$$o_{\text{in}} = \{\, x \leq 10,\ 1 \leq y \leq 11,\ x - y \leq -1,\ x + y \leq 21 \} \quad x := x - y \quad \{\, o_{\text{out}} \,\} \tag{1}$$

In the Octagon domain, each constraint has the form $\pm x \pm y \geq c$, and computing the most precise o_{out} requires solving one linear program per constraint direction. To compute the tightest lower bound for $-x$, we minimize $y - x$ (as new x value is $x - y$) under the current constraints:

$$c^{\#} = \min_{x,y}(y - x) \quad \text{s.t.} \quad x \leq 10,\ 1 \leq y \leq 11,\ x - y \leq -1,\ x + y \leq 21 \tag{2}$$

Limitation 1: Fixed Imprecision. Solving the above LP yields $c^{\#} = 1$, so the most precise bound for $-x$ is $-x \geq 1$. As solving such optimization problems for all template directions (quadratic in the number of variables for Octagons) at each program point is computationally expensive, ELINA uses a sound but imprecise *interval relaxation* heuristic that ignores relational constraints and reasons only over variable intervals. Given $x \in [-\infty, 10]$ and $y \in [1, 11]$, this yields $y - x \in [-9, \infty]$ (and thus $c^{\#} = -9$), giving $-x \geq -9$, which is sound but much less precise than the true bound $-x \geq 1$. This loss of precision ultimately prevents the analyzer from discovering the invariant $x \leq 0$.

Limitation 2: Lack of Adaptability. The key limitation is not just imprecision, but also rigidity. Although the most precise LP-based transformer could prove the assertion, ELINA never computes it and always defaults to the interval heuristic for efficiency, *regardless of analysis needs.* This fixed design lacks adaptability; for instance, in safety-critical settings where higher precision is crucial, transformers in the current design would offer no way to trade efficiency for precision.

Limitation 3: Imprecision from Per-instruction design. Libraries like ELINA have hard-coded transformers for all instructions, such as assignments, across every domain they support. During analysis, these transformers are applied instruction-by-instruction. It is well known that precision in abstract interpretation is non-compositional, and that this instruction-wise analysis introduces imprecision, as relational information between variables is lost when dependent instructions are analyzed separately. Consider the program in Fig. 2b. The relation $x - y = 20$ holds initially. Since both x and y are updated using the current value of y ($2y = y + y$), $x - y$ remains constant and $x - y = 20$ is a valid loop invariant, so the assertion holds. However, neither current libraries nor substituting their transformers with LP-based ones can prove it, as the core issue is that both

assignments must be analyzed together. This limitation arises because current transformers are implemented separately for each domain and instruction. In such setup, it is infeasible to design and maintain transformers for all possible instruction sequences that may arise during analysis.

This Work: Evolving Abstract Transformer. To address these limitations, we move beyond fixed, hard-coded transformers and introduce the *Evolving Abstract Transformer*, a novel general transformer that reformulates the traditional approach of computing a single sound output into one that defines a parametric space of sound outputs and searches within it. This enables the transformer to adapt and *evolve* toward higher-precision outputs through guided optimization. This is enabled by two core algorithms we propose, the *Universal Parametric Output Space Encoder* (UPOSE) and the *Adaptive Gradient Guidance* (AGG), which we describe next.

3.2 Universal Parametric Output Space Encoder (UPOSE) Algorithm

While analyzing the program in Fig. 2a above, we discussed two sound lower bounds for $-x$: (1) 1, the *most precise* bound obtained by minimizing $y - x$ under the current constraints (Eq 2), and (2) -9, a much looser bound produced by the interval relaxation heuristic applied by Elina. These are not the only sound lower bounds: any value below 1 is sound, with bounds closer to 1 being more precise but costlier to compute (often requiring LP solving), while faster heuristics like interval relaxation yield less precise results. Since analysis tasks have varying precision and time requirements, our *UPOSE* algorithm does not commit to one output but instead captures the space of sound bounds *symbolically*, enabling the subsequent search procedure to adapt precision based on the available time budget. To capture this space of sound bounds, UPOSE exploits the notion of duality and uses the *Lagrangian dual* function. For a problem of the form $\min_x f(x)$ s.t. $Ax \leq b$, the Lagrangian dual function is $g(\boldsymbol{\lambda}) = \min_x \big(f(x) + \boldsymbol{\lambda}^\top (Ax - b)\big)$. By the weak duality theorem [7], the dual function of the minimization problem in Eq. 2 satisfies $g(\boldsymbol{\lambda}) \leq c^\#$ for every feasible $\boldsymbol{\lambda} \geq 0$, giving us a principled way to parametrize the space of lower bounds $g(\boldsymbol{\lambda})$ in terms of Lagrange multipliers $\boldsymbol{\lambda}$. However, using this form directly has two challenges: (i) the inner minimization defining $g(\boldsymbol{\lambda})$ is not trivial to compute efficiently, and (ii) many choices of multipliers $\boldsymbol{\lambda}$ make this minimization diverge to $-\infty$, yielding trivial meaningless bounds. UPOSE overcomes these challenges and constructs, for each constraint direction, a *Parametric Scalar Map (PSM)* $\mathcal{M} = \{\Theta, L\}$, where each $\theta \in \Theta$ defines a *finite* sound lower bound $L(\theta)$.

First, to efficiently compute and bound $g(\boldsymbol{\lambda})$, UPOSE creates the dual function $g(\boldsymbol{\lambda})$ by introducing Lagrange multipliers *only for the relational* constraints (which capture inter-variable dependencies) and keeping variable bounds intact. The key motivation is that variable bounds define a bounding box over which the inner minimization can be solved cheaply in closed form, as we will see later. Introducing multipliers for all constraints would eliminate this structure, making the inner minimization hard to compute. Instead, we retain the box constraints and only dualize the relational constraints such as $\pm x \pm y \leq c$ in octagons. Most numerical domains already track sound variable bounds, and UPOSE also works even when some or all of these bounds are absent. For our running example (Eq. 2), introducing non-negative multipliers $\boldsymbol{\lambda} = (\lambda_1, \lambda_2)$ for the relational constraints $x - y \leq -1$ and $x + y \leq 21$ yields:

$$o_{dual} = \max_{\lambda_1, \lambda_2 \geq 0} \; \min_{x \in [-\infty, 10],\, y \in [1, 11]} \; y - x + \lambda_1 (x - y + 1) + \lambda_2 (x + y - 21)$$

After simplifying the inner minimization, this construction gives the dual function:

$$g(\lambda_1, \lambda_2) = \min_{x \in [-\infty, 10],\; y \in [1, 11]} (\lambda_1 + \lambda_2 - 1)x + (-\lambda_1 + \lambda_2 + 1)y + \lambda_1 - 21\lambda_2$$

Now, evaluating $g(\lambda_1, \lambda_2)$ over feasible $\lambda_1, \lambda_2 \geq 0$ yields a continuous family of sound bounds parameterized by $\boldsymbol{\lambda}$. But as mentioned above, certain choices of λ_1 and λ_2 make the inner minimization diverge to $-\infty$, which is a sound but trivial lower bound. Hence, while computing g, UPOSE also derives additional constraints on the multipliers to ensure that the resulting parametric space contains only sound and *finite* lower bounds meaningful for downstream analysis. To compute $g(\lambda_1, \lambda_2)$, UPOSE uses the fact that the retained variable bounds make the inner minimization over an interval hyperbox. This structure allows a closed-form minimum (which would not be possible without the bounds) by treating each variable separately and computing its contribution from its bounds. We start with $\mathcal{M} = \{\Theta, L\}$, where Θ initially includes $\lambda_1, \lambda_2 \geq 0$ and $L = 0$, and we iteratively update it with additional constraints while accumulating each variable's contribution to L as follows:

(1) The x term $(\lambda_1 + \lambda_2 - 1)x$ is minimized over $x \in [-\infty, 10]$. Since the lower bound is unbounded, finiteness requires $(\lambda_1 + \lambda_2 - 1) \leq 0$, preventing the minimum from diverging. We add this constraint to Θ and the contribution $L_x = 10(\lambda_1 + \lambda_2 - 1)$ to L.
(2) The y term $(-\lambda_1 + \lambda_2 + 1)y$ is minimized over $y \in [1, 11]$, where both bounds are finite, so the result is finite and Θ remains unchanged. The minimum depends on the sign of the coefficient: if $(-\lambda_1 + \lambda_2 + 1) \geq 0$, it occurs at $y = 1$; otherwise, at $y = 11$. It is easy to verify that these cases combine into the closed form $L_y = -5|\lambda_2 - \lambda_1 + 1| + 6(\lambda_2 - \lambda_1 + 1)$ which we add to L.

Adding the constant term $(\lambda_1 - 21\lambda_2)$ to L yields the final scalar map $\mathcal{M} = \{\Theta, L\}$, where, after simplification, $L = 5\lambda_1 - 5\lambda_2 - 5|\lambda_2 - \lambda_1 + 1| - 4$ and $\Theta = \{\lambda_1 \geq 0,\ \lambda_2 \geq 0,\ \lambda_1 + \lambda_2 - 1 \leq 0\}$. This captures the space of sound lower bounds for our problem (Eq. 2). Collecting such maps for all template directions yields the *parametric space of sound outputs* (details in Sec. 4). As discussed in Sec. 4, the constructed output space has the property that setting all parameters (multipliers) to zero recovers the interval relaxation result. In this example, $\boldsymbol{\lambda} = (0, 0)$ satisfies Θ and gives the finite lower bound $L = -9$, matching the interval bound discussed earlier. The parametric space also contains the most precise lower bound for linear operators (like the linear assignment in our example). Choosing $\boldsymbol{\lambda}^* = (1, 0)$ yields $L = 1$, matching the most precise bound discussed above.

Construction for quadratic case. The example discussed above demonstrates the construction for a linear operator. However, the UPOSE algorithm also supports quadratic operators. For instance, if the operator at Line 7 is replaced by the quadratic update $x := -x^2$, while keeping the same input bounds and relational constraints, the corresponding inner minimization becomes:

$$\min_{x \in [-\infty, 10],\ y \in [1, 11]} x^2 + (\lambda_1 + \lambda_2)\, x + (-\lambda_1 + \lambda_2)\, y + \lambda_1 - 21\lambda_2, \quad \lambda_1, \lambda_2 \geq 0.$$

This minimization is harder than in the linear case because the objective mixes a quadratic term x^2 with a linear term in x, making it unclear how to separate their contributions as before. To address this, UPOSE introduces a split parameter s that decomposes the x-coefficient as $(\lambda_1 + \lambda_2) = s + (\lambda_1 + \lambda_2 - s)$, allowing the objective to be separated into quadratic and linear parts, which yields:

$$\hat{g}(\lambda_1, \lambda_2, s) = \underbrace{\min_x (x^2 + sx)}_{\text{quadratic part}} + \underbrace{\min_{x,y} [(\lambda_1 + \lambda_2 - s)x + (-\lambda_1 + \lambda_2)y + \lambda_1 - 21\lambda_2]}_{\text{linear part}},$$

We have $\hat{g}(\lambda_1, \lambda_2, s) \leq g(\lambda_1, \lambda_2) \leq c^{\#}$, where the first inequality holds because the sum of independent minima is always less than or equal to the joint minimum, and the second follows from weak duality. Hence, $\hat{g}(\lambda_1, \lambda_2, s)$ defines a sound parametric space of lower bounds for the quadratic operator. Linear part is minimized as discussed above. For the quadratic part, similar conditions for finiteness and minimized value can be deduced by checking the interval boundaries and the critical point (details in Appendix A.3). Although strong duality does not hold for nonlinear operators and

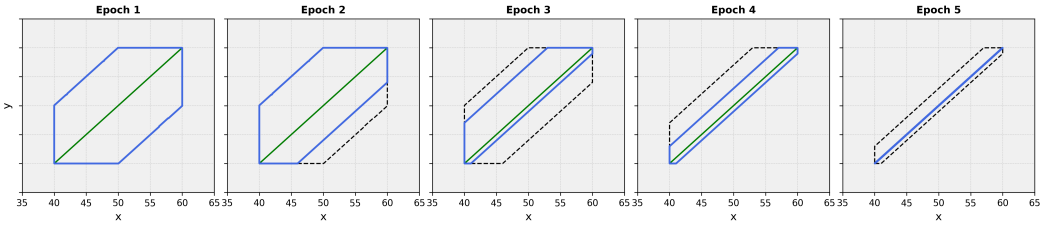


Fig. 3. Transformer output (blue) converging to most precise output (green) over successive gradient steps.

the space may not include the exact optimum $c^{\#}$, this formulation enables AbsEvolve to achieve more precise invariants for non-linear assignments in practice (Sec. 7.2).

Supports Multiple Domains and Operator Sequences. The UPOSE algorithm is general and works uniformly across all Polyhedral numerical domains (Sec. 2), and supports the broad range of *Quadratic-Bounded Guarded Operators (QGOs)* (Sec. 4). QGOs combine linear or quadratic updates with guards expressed as conjunctions of linear inequalities, covering affine assignments, quadratic updates, guarded assignments, and conditionals like `assume`. This formulation extends naturally beyond single assignments, as many common instruction sequences have combined semantics that lie within the QGO class and can therefore be analyzed jointly. For instance, the sequence $[x := a + b;\ y := x \cdot c]$ yields the final updates $x = a + b$ and $y = (a + b) \cdot c$, both of which are quadratic-bounded, and the sequence lies in the QGO class. The AbsEvolve framework (Sec. 6) merges such sequences into blocks and uses the UPOSE algorithm to analyze them jointly and yield more precise invariants. This enables AbsEvolve to infer the invariant $x - y = 20$ and prove the assertion in Fig. 2b, by analyzing the loop body $[x := x - y;\ y := 2 \cdot y]$ as a single unit.

3.3 Adaptive Gradient Guidance (AGG) Algorithm

The parametric family of outputs computed by UPOSE is defined by t Parametric Scalar Maps (PSMs), indexed by $i \in \{1, \ldots, t\}$. Each PSM $\mathcal{M}_i = \{\Theta_i, L_i\}$ characterizes the space of sound, finite lower bounds admissible for the i-th constraint. The next step is to traverse this space and pick concrete bounds which can then be used for further analysis. We only need to consider non-empty Θ_i as if Θ_i is empty or infeasible, it indicates that no finite lower bound can be inferred for that direction, and the only sound bound under our construction is $-\infty$. For efficient traversal of the space, we propose the *Adaptive Gradient Guidance (AGG)* (Sec. 5) algorithm, which is a gradient-guided search procedure to select $\theta_i \in \Theta_i$ suitable for analysis tasks. It takes as input a differentiable score function $\mathcal{J}_i$ that evaluates the quality of each bound $L_i(\theta)$ for the analysis task via $\mathcal{J}_i(L_i(\theta))$, as well as a number of gradient steps $\mathcal{R}$ that controls the runtime of the search. Exploiting the piecewise differentiability of the constructed space L (discussed in Sec 4), AGG uses the gradient $\nabla\mathcal{J}_i(L_i(\theta))$ to efficiently navigate it, steering toward high-scoring bounds while keeping θ within the feasible region Θ_i. When optimizing for precision, where higher lower bounds indicate greater precision, the score function $\mathcal{J}_i$ can simply be set to L_i, and AGG performs gradient ascent to obtain tighter sound bounds within the feasible region. For instance, the PSM for our example above has $L(\lambda_1, \lambda_2) = 5\lambda_1 - 5\lambda_2 - 5|\lambda_2 - \lambda_1 + 1| - 4$ with $\Theta = \{\lambda_1 \geq 0,\ \lambda_2 \geq 0,\ \lambda_1 + \lambda_2 \leq 1\}$. To obtain more precise bounds (i.e., higher L), the Adaptive Gradient Guidance (AGG) algorithm starts from the interval relaxation baseline $(\lambda_1, \lambda_2) = (0, 0)$ and performs gradient-guided updates that increase L within Θ:

$$((\lambda_1, \lambda_2), L):\ ((0,0), -9)\ \Rightarrow\ ((0.3,0), -6)\ \Rightarrow\ ((0.6,0), -3)\ \Rightarrow\ ((1,0), 1)\ \text{(most precise)}.$$

The AGG procedure converges to the most precise bound in just three iterations. Figure 3 illustrates the per-step progression of AGG as it converges to the most precise invariant $x - y = 20$ during the analysis of program in Fig. 2b. The evaluation results in Section 7.1 also demonstrate that AGG enables finding the most precise outputs significantly faster than solver-based baselines, highlighting the efficacy of gradient-guided search in efficiently exploring the parametric space.

4 Construction of Parametric Space of Sound Outputs

In this section, we formalize the notion of a parametric space of sound outputs and describe how the *Universal Parametric Output Space Encoder (UPOSE)* algorithm systematically constructs such a space. The algorithm works for any operator whose overall effect can be expressed as a *Quadratic-Bounded Guarded Operator (QGO)*. We begin by defining the notion of an *Effective Update Map (EUM)*, which compactly represents the symbolic effect of an operator on program variables.

Definition 4.1 (Effective Update Map). *Let $\mathcal{V}$ be the finite set of program variables and O a concrete operator (atomic or composite). The* Effective Update Map (EUM) *of O is a total function $\sigma : \mathcal{V} \to \mathbb{E}(\mathcal{V})$, where $\mathbb{E}(\mathcal{V})$ denotes symbolic expressions over $\mathcal{V}$. Let $\mathcal{U} \subseteq \mathcal{V}$ denote the set of updated variables. For each $v \in \mathcal{U}$, $\sigma(v)$ denotes its final value after applying O; for each unchanged variable $v \notin \mathcal{U}$, we have $\sigma(v) = v$. The map is flat, meaning every expression $\sigma(v)$ is written directly in terms of the original variables in $\mathcal{V}$ without referencing intermediate assignments.*

Quadratic-Bounded Guarded Operators. A *Quadratic-Bounded Guarded Operator (QGO)* is any concrete operator O whose semantics can be represented by a pair $(\sigma_{\leq 2}, \mathcal{G})$, where $\sigma_{\leq 2}$ is a *quadratic-bounded effective update map* and $\mathcal{G}$ is a *conjunctive linear guard*. The EUM $\sigma_{\leq 2} : \mathcal{V} \to \mathcal{P}_{\leq 2}(\mathcal{V})$ assigns each updated variable a polynomial of total degree at most two, i.e., $\sigma_{\leq 2}(v_j) = \sum_{i \leq k} a_{ik} v_i v_k + \sum_i b_i v_i + c$, with real coefficients a_{ik}, b_i, c, while the guard $\mathcal{G} = \{Pv \leq d\}$ is a conjunction of linear inequalities which restricts the input space. Either component may be absent, allowing only updates when $\mathcal{G}$ is empty and only guards when $\sigma_{\leq 2}$ is the identity. This formulation covers all commonly occurring instructions handled by current libraries like Elina [47], including affine assignments, quadratic updates, guarded assignments, and conditional statements like `assume`. Moreover, a key benefit of this formulation is its expressiveness: it extends beyond single instructions to sequences whose combined effect can be flattened into a quadratic-bounded map, thereby enabling joint block-level analysis. For example, the sequence $[x := a + b;\ y := x \cdot c]$ corresponds to $\sigma_{\leq 2}(x) = a + b$ and $\sigma_{\leq 2}(y) = (a + b) \cdot c$, respectively, which lies within the QGO class and can be analyzed jointly by the UPOSE algorithm.

Template-Constrained Outputs. The UPOSE algorithm constructs a *template-constrained* space of outputs, i.e. each output in the space is expressed over a fixed Template Constraint Matrix (TCM) $\mathcal{T} \in \mathbb{R}^{t \times n}$ (Sec. 2). For template-based domains such as Zones or Octagons, $\mathcal{T}$ directly corresponds to the domain's intrinsic constraint structure. In contrast, for expressive domains such as general Polyhedra, where templates are not fixed, any user-specified $\mathcal{T}$ can be provided; by default, the algorithm uses the Zone template. If the underlying abstract domain $\mathcal{D}$ supports abstract elements of arbitrary shapes, as in the Polyhedra domain, the generated outputs can be used directly. Otherwise, the domain must provide a *sound conditional operator*, which soundly intersects abstract elements with arbitrary linear constraints. In such cases, the generated outputs can be used in two ways: (i) to refine the result of a baseline analyzer by intersecting its output with the generated output constraints, or (ii) independently by intersecting the constraints with $\top$, yielding an abstract element $\mathsf{a}_{\mathcal{D}} \in \mathcal{D}$ that over-approximates a_{out}, i.e., $\gamma(\mathsf{a}_{out}) \subseteq \gamma(\mathsf{a}_{\mathcal{D}})$.

Parametric Space of Outputs. In template domains, an abstract element is of the form $\mathsf{a} = \{\mathcal{T}\mathbf{v} \geq \mathbf{c}\}$, characterized by scalar lower bounds $\mathbf{c} = [c_1, \ldots, c_t]^T$, where each c_i corresponds to

constraint $\mathcal{T}_i$. Instead of fixing these bounds to a single value, we allow them to vary as functions of parameters, thereby representing a space of possible abstract elements in a unified form. This is captured by *Parametric Scalar Maps* (PSMs):

Definition 4.2 (Parametric Scalar Map). *A Parametric Scalar Map (PSM) is a pair $\mathcal{M} = (\Theta, L)$ where $\Theta \subseteq \mathbb{R}^d$ is a parameter space, and $L : \Theta \to \mathbb{R}$ maps each $\theta \in \Theta$ to a finite scalar value.*

Each template constraint is assigned a PSM $\mathcal{M}_i$, which encodes a space of admissible values for the corresponding bound c_i. A collection of such PSMs, one for each template direction, is used to define our parametric space of outputs as follows.

Definition 4.3 (Parametric Space of Abstract Elements). *Let $\mathcal{T} \in \mathbb{R}^{t \times n}$ be a fixed template matrix. Consider the ordered collection of t PSMs $\overline{\mathcal{M}} = [\mathcal{M}_1, \ldots, \mathcal{M}_t]$ where $\mathcal{M}_i = (\Theta_i, L_i)$. Each $\mathcal{M}_i$ is extended to a total map $L'_i : \mathbb{R}^{d_i} \to \mathbb{R} \cup \{-\infty\}$ defined as $L'_i(\theta_i) = L_i(\theta_i)$ if $\theta_i \in \Theta_i$, and $L'_i(\theta_i) = -\infty$ otherwise. The collection $\overline{\mathcal{M}}$ induces the following parametric space of elements:*

$$\mathcal{S}_{\mathcal{T}}(\overline{\mathcal{M}}) = \lambda\boldsymbol{\theta}.\ \{\mathcal{T}_i\mathbf{v} \geq L'_i(\theta_i) \mid i = 1, \ldots, t\}$$

Fixing choices of parameter θ, such as, $\boldsymbol{\theta} = (\hat{\theta}_1, \ldots, \hat{\theta}_t)$ generates abstract elements $\mathcal{S}_{\mathcal{T}}(\overline{\mathcal{M}}, \boldsymbol{\theta}) = \{\mathcal{T}_i\mathbf{v} \geq L'_i(\hat{\theta}_i) \mid i = 1, \ldots, t\}$.

Intuitively, each PSM $\mathcal{M}_i$ specifies the finite values that the bound c_i can take. When $\theta_i \notin \Theta_i$, c_i defaults to $-\infty$, effectively removing the constraint since it always holds. If Θ_i is empty, c_i has no finite value, no output can have finite c_i and the entire space omits the corresponding constraint.

Example 4.1. *Suppose the template $\mathcal{T}$ includes two rows corresponding to the constraints $x + y \geq c_1$ and $x - y \geq c_2$. Consider the two Parametric Scalar Maps:*

$$\mathcal{M}_1 = (\Theta_1 = \{\lambda_1 \geq 1,\ \lambda_2 \geq 1,\ \lambda_1 + \lambda_2 \geq 1\},\ L_1(\lambda_1, \lambda_2) = 2\lambda_1 + \lambda_2),$$
$$\mathcal{M}_2 = (\Theta_2 = \{\lambda_3 \geq 1,\ \lambda_4 \geq 1,\ \lambda_3 + \lambda_4 \leq 1\},\ L_2(\lambda_3, \lambda_4) = 3\lambda_3 + 2\lambda_4)$$

$\mathcal{S}_{\mathcal{T}}([\mathcal{M}_1, \mathcal{M}_2])$ defines a parametric space of elements with each parameter tuple $(\lambda_1, \lambda_2, \lambda_3, \lambda_4)$ defining one element in this space. For $\mathcal{M}_1$, feasible parameters yield finite bounds; for instance, $(\lambda_1, \lambda_2) = (1, 1)$ gives $x + y \geq 3$ and $(2, 1)$ gives $x + y \geq 5$. For $\mathcal{M}_2$, no feasible parameters exist since $\lambda_3, \lambda_4 \geq 1$ and $\lambda_3 + \lambda_4 \leq 1$ cannot hold together, so every element in the space has $x - y \geq -\infty$.

A parametric space is *sound space of outputs* for a given operator f and input element a_{in} if all possible instantiations (all abstract outputs in the space) over-approximate the effect of f on a_{in}.

Definition 4.4 (Soundness of a Parametric Space). *A parametric space $\mathcal{S}_{\mathcal{T}}(\overline{\mathcal{M}})$ for a template $\mathcal{T}$ is a sound space of outputs for an input abstract element a_{in} and concrete operator f if, for all parameter tuples $\boldsymbol{\theta}$, $f(\gamma(\mathsf{a}_{in}) \subseteq \gamma(\mathcal{S}_{\mathcal{T}}(\overline{\mathcal{M}}, \boldsymbol{\theta}))$.*

The UPOSE algorithm (Alg. 1) constructs a sound parametric space of outputs in template $\mathcal{T}$ for an input abstract element $\mathsf{a}_{in} = \{A\mathbf{v} \leq b\}$ from polyhedral domain $\mathcal{D}$ and a QGO operator $\mathcal{O} = \{\sigma_{\leq 2},\ \mathcal{G} = P\mathbf{v} \leq d\}$. It works in two steps: first, it computes *effective objectives and constraints* that capture how each template constraint transforms under $\mathcal{O}$; second, it derives the PSM set $\overline{\mathcal{M}} = [\mathcal{M}_1, \ldots, \mathcal{M}_t]$, where each PSM $\mathcal{M}_i$ *soundly lower bounds* the i^{th} template constraint. The resulting set $\overline{\mathcal{M}}$ defines the parametric output space $\mathcal{S}_{\mathcal{T}}(\overline{\mathcal{M}})$. We detail these steps below:

Step 1: Computing Effective Objectives and Constraints. First, UPOSE collects the *effective constraints* $\hat{G}$ by combining the constraints in the input $\{A\mathbf{v} \leq b\}$ with those introduced by the operator $\{P\mathbf{v} \leq d\}$, i.e. $\hat{G} = \{\mathbf{v} \mid \bar{A}\mathbf{v} \leq \bar{b}\}$, where $\bar{A} = \begin{bmatrix} A \\ P \end{bmatrix}$ and $\bar{b} = \begin{bmatrix} b \\ d \end{bmatrix}$. Intuitively, $\hat{G}$ captures the joint feasible region over which subsequent bounds are computed. Next, for each

Algorithm 1 Universal Parametric Output Space Encoder (UPOSE) Algorithm

Input: Input $\mathsf{a}_{in} = \{A\mathbf{v} \le b\}$, QGO operator $O = \{\sigma_{\le 2}, \mathcal{G} = P\mathbf{v} \le d\}$, Template $\mathcal{T} \in \mathbb{R}^{t \times n}$
$\overline{\mathcal{M}} \leftarrow []$
$\hat{G} \leftarrow \{A\mathbf{v} \le b\} \cup \{P\mathbf{v} \le d\}$ ▷ Effective Output Constraints (Input and Guard)
for $i = 1$ to t **do**
  $\hat{F} \leftarrow \textsc{GetEffectiveObjective}(\mathcal{T}_i, \sigma_{\le 2})$ ▷ Updated value of constraint $\mathcal{T}_i$
  $\mathcal{M}_i \leftarrow \textsc{ParametricLowerBound}(\hat{F}, \hat{G})$ ▷ Duality-based parametric bounding of $\hat{F}_i$
  $\overline{\mathcal{M}}.\mathsf{append}(\mathcal{M}_i)$
end for
return $\mathcal{S}_{\mathcal{T}}(\overline{\mathcal{M}})$ ▷ Construct Parametric Space with the bounding PSMs

template constraint direction $\mathcal{T}_i$, UPOSE constructs an *effective objective* $\hat{F}_i$ that captures how the corresponding constraint evolves under the QGO O. Each $\hat{F}_i$ is obtained by applying the QGO's update map $\sigma_{\le 2}$ to the linear form $\mathcal{T}_i\mathbf{v}$, yielding $\hat{F}_i = \sum_j \mathcal{T}_{i,j} \cdot \sigma_{\le 2}(v_j)$. Since $\mathcal{T}_i$ is linear and $\sigma_{\le 2}$ produces expressions of degree at most two, each $\hat{F}_i$ remains quadratic-bounded. For example, let the input element be $\mathsf{a}_{in} = \{\, 0 \le d \le 2,\ 0 \le e \le 1 \,\}$, the operator O have updates $\sigma_{\le 2}(x) = d^2$, $\sigma_{\le 2}(y) = 3d+e$, and guard $\mathcal{G} = \{\, d+e \le 5 \,\}$, and the template row be $\mathcal{T}_i = (1, 1)$ (i.e., $x+y$). Then the effective objective is $\hat{F}_1 = d^2+3d+e$, and the feasible region is $\hat{G} = \{\, 0 \le d \le 2,\ 0 \le e \le 1,\ d+e \le 5 \,\}$.

Step 2: Deriving Parametric Lower Bounds. The most precise lower bound for template constraint $\mathcal{T}_i$ can be computed by solving the minimization problem: $c_i^{\#} = \min_{\mathbf{v} \in \hat{G}} \hat{F}_i(\mathbf{v})$. But this is computationally expensive and returns only a fixed value. To obtain a space of sound bounds, we apply our *Parametric Bounding Procedure* (App. A), which takes $(\hat{F}_i, \hat{G})$ as input and returns a *sound lower-bound PSM* $\mathcal{M}_i = (\Theta_i, L_i)$ satisfying $L_i(\theta_i) \le c_i^{\#}$ for all $\theta_i \in \Theta_i$. This means that each value produced by L_i forms a valid lower bound on $c_i^{\#}$, compactly representing a space of sound bounds for the i^{th} template constraint. The procedure derives this space symbolically in three key steps:

(1) **Dual Construction with Domain-Aware Simplification:** The first step is to construct the Lagrangian dual of the problem: $\min_{\mathbf{v} \in \hat{G}} \hat{F}_i(\mathbf{v})$. Since abstract domains already provide tight variable bounds, we treat these as box constraints and introduce Lagrange multipliers only for the remaining relational ones, yielding: $\max_{\boldsymbol{\lambda} \ge 0}\ \min_{\mathbf{v} \in [\mathbf{l}, \mathbf{u}]} \left(\hat{F}_i(\mathbf{v}) + \boldsymbol{\lambda}^\top (\bar{A}\mathbf{v} - \bar{b})\right)$. This also works when some or all bounds are missing by using $\pm\infty$ as limits. Here, the result of inner minimization is known as the dual function $g(\boldsymbol{\lambda})$. Considering the dual function has a key advantage: by the Weak Duality Theorem [7], its value for any feasible $\boldsymbol{\lambda}$ serves as a certified lower bound on the primal optimum $c_i^{\#}$, providing a principled way to obtain a space of sound lower bounds, parametrized by the dual variables $\boldsymbol{\lambda}$. Furthermore, the box-constrained structure allows $g(\boldsymbol{\lambda})$ to be computed symbolically and enables simplifications not possible in general quadratic programs (details of dual construction in Appendix A.1).

(2) **Symbolic Decomposition via Coefficient Splitting:** Another insight is that although solving the inner minimization defining $g(\boldsymbol{\lambda})$ is difficult due to bilinear and indefinite quadratic terms, we can introduce symbolic parameters $(\mathbf{S}, \mathbf{D})$ to *redistribute linear coefficients* across the objective:

$$\underbrace{\min_{\mathbf{v}} \left(\mathbf{v}^\top H\mathbf{v} + \mathbf{q}^\top\mathbf{v} + \boldsymbol{\lambda}^\top A\mathbf{v}\right)}_{g(\boldsymbol{\lambda})} \rightsquigarrow \underbrace{\min_{\mathbf{v}} \sum_{i<j} (H_{ij}v_iv_j + D_{ij}^i v_i + D_{ij}^j v_j) + \min_{\mathbf{v}} \sum_i (Q_i v_i^2 + S_i v_i) + \cdots}_{\hat{g}(\boldsymbol{\lambda}, \mathbf{S}, \mathbf{D})}$$

Here, H_{ij} and Q_i are fixed coefficients from the original quadratic form, while D_{ij}^i, D_{ij}^j, and S_i are symbolic split parameters that control the redistribution of linear terms. This coefficient

splitting decomposes the objective into independent 1D and 2D subproblems that can be minimized efficiently in isolation (Appendix A.2). The decomposition is sound: by minimizing each subproblem independently, we obtain a value less than or equal to the true joint minimum, thus yielding a valid lower bound on the original objective: $\forall\, \boldsymbol{\lambda} \geq 0,\ \mathbf{S}, \mathbf{D}.\ \hat{g}(\boldsymbol{\lambda}, \mathbf{S}, \mathbf{D}) \leq g(\boldsymbol{\lambda}) \leq c_i^{\#}$.

(3) **Tractable, Modular Evaluation of Subproblems:** Each resulting subproblem is now either a linear, bilinear, or quadratic expression over a box (1D or 2D), which can be evaluated using closed-form expressions or cheap bounded minimization routines. Some subproblems may have unbounded minima over all $\boldsymbol{\lambda} \geq 0$, but we aim to capture the space of non-trivial (finite) bounds and thus derive additional conditions on $\boldsymbol{\lambda}$ under which they minimum remains bounded, and add these to the feasible region Θ_i. The resulting symbolic expression for minimum is captured in L_i, thus yielding the lower bounding PSM $\mathcal{M}_i = \{\Theta_i, L_i\}$ (Appendix A.3).

Having defined the construction of the PSMs $\overline{\mathcal{M}}$ that collectively define the parametric output space $\mathcal{S}_{\mathcal{T}}(\overline{\mathcal{M}})$, we now formally establish the soundness and precision guarantees.

Theorem 4.1 (Soundness of the Constructed Parametric Space). *Let $\mathcal{S}_{\mathcal{T}}(\overline{\mathcal{M}})$ be the parametric space of elements constructed by the UPOSE algorithm for an input element a_{in}, QGO operator O and template $\mathcal{T}$, where $\overline{\mathcal{M}} = [\mathcal{M}_1, \ldots, \mathcal{M}_t]$ is the collection of PSMs $\mathcal{M}_i$ obtained by lower bounding effective objectives $\hat{F}_i$. Then, $\mathcal{S}_{\mathcal{T}}(\overline{\mathcal{M}})$ is a sound space of outputs (Def. 4.4) for a_{in} and O.*

Proof. Assume that $\mathcal{S}_{\mathcal{T}}(\overline{\mathcal{M}})$ is not a sound space of outputs for a_{in} and O. Then, by Definition 4.4, there exists $\hat{\boldsymbol{\theta}}$ such that $f(\gamma(\mathsf{a}_{in})) \not\subseteq \gamma(\mathcal{S}_{\mathcal{T}}(\overline{\mathcal{M}}, \hat{\boldsymbol{\theta}}))$. For the QGO $O = \{\sigma_{\leq 2}, \mathcal{G}\}$, this means there is $\mathbf{v} \in \gamma(\mathsf{a}_{in})$ with $\mathcal{G}(\mathbf{v})$ and $\mathbf{v}' = \sigma_{\leq 2}(\mathbf{v})$ such that $\mathbf{v}' \notin \gamma(\mathcal{S}_{\mathcal{T}}(\overline{\mathcal{M}}, \hat{\boldsymbol{\theta}}))$. This implies that some template constraint j is violated, i.e., $\mathcal{T}_j\mathbf{v}' > L_j(\hat{\theta}_j)$. By Theorem C.1 in Appendix, we have $L_j(\hat{\theta}_j) \leq c_j^{\#}$ (each PSM encodes lower bounds on the minimum value of its template constraint), and since $c_j^{\#}$ is the minimum of $\mathcal{T}_j\mathbf{v}'$ over all reachable states, it follows that $c_j^{\#} \leq \mathcal{T}_j\mathbf{v}'$, leading to a contradiction. Thus, such a violation cannot occur, and $\mathcal{S}_{\mathcal{T}}(\overline{\mathcal{M}})$ is sound space of outputs for a_{in} and O. □

Theorem 4.2 (Most Precise Output for Linear Case). *For QGO operators O with linear $\sigma_{\leq 2}$, the space $\mathcal{S}_{\mathcal{T}}(\overline{\mathcal{M}})$ contains the most precise abstract output. That is, for each PSM $\mathcal{M}_i$, there exists a parameter instantiation that yields the most precise bound $c_i^{\#}$.*

Proof. This follows by Theorem C.2 in Appendix, which uses Strong Duality [7] to show that, for linear cases, the induced space captures the optimum, i.e., there exists $\theta \in \Theta$ such that $L(\theta) = c_i^{\#}$. □

Other key properties of UPOSE. Beyond soundness and precision, the parametric space $\mathcal{S}_{\mathcal{T}}(\overline{\mathcal{M}})$ constructed by our UPOSE algorithm has the following key features:

(1) **Interval Relaxation at $\vec{0}$:** The abstract element $\mathcal{S}_{\mathcal{T}}(\overline{\mathcal{M}}, \vec{0})$, obtained by setting $\boldsymbol{\theta} = \vec{0}$, corresponds to the output of the *interval relaxation* heuristic used commonly in existing libraries, which computes bounds by simply discarding inter-variable constraints and only using the interval bounds of the variables (By Theorem C.2 proved in Appendix).

(2) **Differentiable, Polyhedral Parameterization:** All PSMs $\mathcal{M}_i = (\Theta_i, L_i)$ are such that the parameter space Θ_i is polyhedral, i.e., of the form $\{\mathbf{A}_i\theta_i \leq \mathbf{b}_i\}$ (Theorem C.4 in Appendix), and each map $L_i(\theta_i)$ is piecewise differentiable (Theorem C.5 in Appendix). This structure makes the space easy to traverse and enables the use of gradient-based optimization techniques.

(3) **Extensibility to Joins and Disjunctions:** The construction extends naturally to joins and disjunctions by constructing per-branch parametric spaces and combining them via a min over the joint parameter space, which remains polyhedral and piecewise differentiable, allowing AGG to be applied directly (details in Appendix E).

5 Gradient-Guided Exploration of Parametric Output Space

Once we have a space of sound outputs $\mathcal{S}_{\mathcal{T}}(\overline{\mathcal{M}})$, the next step is to traverse this space and find a parameter vector $\boldsymbol{\theta}$, which yields the abstract element $\mathcal{S}_{\mathcal{T}}(\overline{\mathcal{M}}, \boldsymbol{\theta})$ suitable for analysis. Each PSM $\mathcal{M}_i = \{\Theta_i, L_i\}$ defines the parameter space Θ_i to search, where each $\theta_i \in \Theta_i$ yields a finite bound for the i^{th} template direction. Template directions with empty Θ_i always have the trivial $-\infty$ bound and don't require search. Any efficient search procedure should be guided by the analysis requirements of the downstream task, for which it must account for the following components:

(1) **Search Objective ($\mathcal{J}$):** The search procedure should allow downstream tasks to specify a score function $\mathcal{J}$ that measures the *quality* of candidate outputs, where higher scores correspond to outputs that better suit the analysis goal. For each constraint i, the score function assigns a quality score $\mathcal{J}(L_i(\theta_i))$ to every possible lower bound in the set $\mathcal{L}_i = \{L_i(\theta_i) \mid \theta_i \in \Theta_i\}$. This score can help guide the search process by steering it toward parameter values θ_i that yield more desirable outputs, i.e., pick lower bounds $L_i(\theta_i)$ with higher $\mathcal{J}(L_i(\theta_i))$. Some examples of such score functions include:
   (a) *Precision Objective $\mathcal{J}_{prec}$:* In general program analysis settings, the goal is to infer the most precise invariants. For example, consider a constraint of the form $x - y \geq L(\theta)$ for $\theta \in \Theta$. The most precise invariant is obtained by choosing $\theta^* \in \Theta$ that maximizes $L(\theta)$, since $x - y \geq c_1$ is strictly more precise than $x - y \geq c_2$ whenever $c_1 > c_2$. This reflects the general principle that larger lower bounds yield tighter (i.e., more precise) abstract outputs. Accordingly, one can define the score function as $\mathcal{J}_{prec}(\theta_i) = L_i(\theta_i)$, so that maximizing $\mathcal{J}_{prec}$ directly corresponds to improving precision.
   (b) *Target Objective $\mathcal{J}_{tgt}$:* Beyond computing the most precise invariants, program analysis tasks may require selecting outputs that satisfy a given specification. For example, consider a constraint of the form $x + y \geq L(\theta)$ for $\theta \in \Theta$, and suppose the goal is to prove $x + y \geq e$. This reduces to finding $\theta^* \in \Theta$ such that $L(\theta^*) \geq e$. Accordingly, one can define a score function that measures satisfaction of this condition, such as the violation-based score $\mathcal{J}_{tgt}(\theta_i) = -\max(0,\ e_i - L_i(\theta_i))$, so that higher scores correspond to smaller violations and the maximum is attained when $L_i(\theta_i) \geq e_i$. This example also highlights how the search objective $\mathcal{J}$ enables adaptive search: rather than always computing the most precise output, it can guide the search toward outputs *sufficient* to establish the desired property or meet the task requirements. Techniques proposed by works like DL2 [17] can further be used to encode more complex specifications as differentiable objectives.

(2) **Runtime Control ($\mathcal{R}$):** The search procedure should also expose mechanisms that allow tasks to regulate the computational effort during search by defining some form of runtime budget $\mathcal{R}$. This enables tasks with different runtime constraints to choose appropriate configurations. Examples of such budgets can include search parameters, such as the number of samples, gradient steps, or epochs, or more coarse-grained options like wall-clock timeouts. Varying this budget can allow the tasks to find the desired tradeoff between output quality and runtime.

Limitations of Existing Search Strategies. A naive search strategy is to generate outputs by randomly sampling parameters from the feasible space Θ. Runtime $\mathcal{R}$ can be controlled by limiting the number of samples, and the best output is selected by measuring the score function $\mathcal{J}$. However, this search is unguided and not *objective-aware*, often failing to find high-scoring outputs. Moreover, efficient sampling is difficult when Θ is high-dimensional or defined by complex constraints. On the other hand, a highly objective-aware strategy is to use solvers to directly maximize $\mathcal{J}_i(L_i(\theta_i))$ over each Θ_i. This generates high-quality outputs but is expensive in practice, as it requires solving one optimization problem for each constraint direction i, resulting in many solver invocations. Although solvers provide timeout mechanisms, these apply per call, while downstream tasks usually

Algorithm 2 Adaptive Gradient Guidance (AGG) Search

Input: score $\mathcal{J}(L(\theta))$, feasible region $\Theta = \{\hat{\mathbf{A}}\theta \leq \hat{\mathbf{b}}\}$, step η, penalty β, gradient steps R
Init: $\theta \leftarrow 0$, $\theta^{\text{best}} \leftarrow 0$ ▷ Start from interval relaxation
for $r = 1$ to R **do**
  if $\theta \in \Theta$ **then**
    $\theta \leftarrow \theta + \eta \nabla \mathcal{J}(L(\theta))$ ▷ Ascend along objective inside feasible region
  else
    $\theta \leftarrow \theta - \eta\beta\nabla \| \max(\hat{\mathbf{A}}\theta - \hat{\mathbf{b}}, 0)\|_p$ ▷ Reduce violations outside feasible region
  end if
  if $\theta \in \Theta$ and $\mathcal{J}(L(\theta)) > \mathcal{J}(L(\theta^{\text{best}}))$ **then**
    $\theta^{\text{best}} \leftarrow \theta$ ▷ Record best feasible parameters found so far
  end if
end for
return θ^{best}

impose a global runtime budget $\mathcal{R}$. When $\mathcal{R}$ is split across many calls, as in big programs with many template constraints, each call gets little time, often yielding no feasible or low-quality outputs. In summary, random sampling provides controllable runtime but lacks objective guidance, whereas solver-based optimization offers strong objective guidance but limited runtime control.

Gradient-Guided Exploration. To balance objective awareness with runtime control, an ideal search procedure should progressively move toward parameters that yield higher scores under the objective $\mathcal{J}_i$. Gradient-guided search naturally fits this goal: it enables incremental improvement, avoids costly solver calls or exhaustive enumeration, and improves output quality as more runtime becomes available. This is especially well-suited for our task, because as discussed in the last section, the output $L_i(\theta_i)$ is piecewise differentiable. Assuming the score function $\mathcal{J}_i$ can be defined differentiably, as is the case for many tasks including precision objective, we can follow the gradient $\nabla \mathcal{J}_i(L_i(\theta_i))$ to optimize it. This results in an iterative procedure where the number of gradient steps serves as a natural and fine-grained control for the runtime $\mathcal{R}$. Unlike solver-based methods that require separate invocations per constraint, gradient descent enables joint optimization across all template directions, progressively improving output quality within the available budget.

However, applying standard descent methods in our setting presents several challenges. We have a constrained maximization problem: maximize score $\mathcal{J}(L(\theta))$ over the parameter space Θ, which is polyhedral (as discussed in the previous section) and so defined as $\hat{A}\theta \leq \hat{b}$. A naive gradient step $\theta \leftarrow \theta + \eta \nabla \mathcal{J}(L(\theta))$ may leave the feasible region Θ. Projected Gradient Descent (PGD) corrects this by projecting back onto Θ, but for polyhedral regions, each projection requires solving a quadratic or linear program, making repeated updates computationally expensive. Penalty-based methods avoid projection by adding a penalty term such as $-\beta\|\max(\hat{A}\theta - \hat{b}, 0)\|_p$ to the objective. However, this assumes $\mathcal{J}(L(\theta))$ is defined for all $\theta \in \mathbb{R}^d$, whereas $L(\theta)$ is only valid within Θ, making the objective ill-posed outside it. Barrier methods, such as log-barrier or interior-point techniques, maintain feasibility by penalizing boundary violations. But they require strict feasible initialization and solve large linear problems at each iteration, making them unscalable. To overcome these challenges, we propose a lightweight, projection-free algorithm called *Adaptive Gradient Guidance (AGG)* (Alg. 2). AGG avoids solver calls and does not require a strictly feasible start. It adaptively selects the update direction based on whether the current point θ lies inside the feasible region Θ. When feasible, it ascends along the gradient of the score to improve output quality; when infeasible, it follows a penalty gradient that reduces constraint violations to restore feasibility.

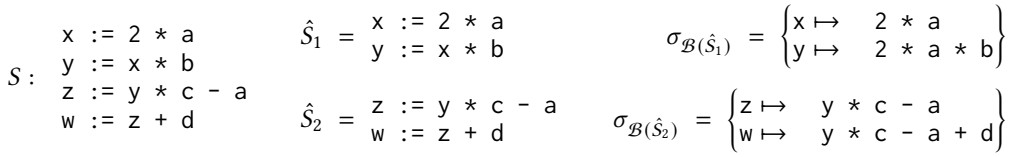


Fig. 4. While merging assignment sequences, AbsEvolve partitions the sequence S into two sub-sequences and merges them into blocks, each of whose Effective Update Map remains quadratic-bounded. Without this partitioning, the combined update would exceed quadratic degree, violating the QGO restriction.

Monotonic Progress Over Baseline. The AGG algorithm enables objective-aware progress within Θ while steering infeasible iterates back without projections or solver calls. As discussed in the previous section, $\theta = 0$ corresponds to the interval-relaxation output used by existing libraries, so we start the search from this point as a natural baseline. The algorithm tracks the best feasible parameters after each gradient step, making the resulting progress *monotonic*: the returned output is never weaker than the baseline, and increasing the search budget can only preserve or improve the result. After the allotted number of gradient steps, the procedure returns the best-scoring parameters found. For linear cases, since the parametric space contains the most precise output (Theorem 4.2), AGG can, in principle, recover this output as well. This is demonstrated in Section 7.1, where our gradient-guided search efficiently finds the most precise outputs in practice.

6 AbsEvolve Framework

We implement the ideas discussed above in AbsEvolve[1] framework. It is built upon the Clam[2] analyzer (part of SeaHorn [20] framework) and the numerical abstract domain library Elina [48]. As discussed above, the UPOSE algorithm works for a wide range of QGO operators. For our implementation, we focus on on assignment operators, which are common in real-world programs. This includes assignment instructions that fall in QGO class such as (1) Affine assignments: $v_j := \sum_i c_i v_i + b$, with $c_i, b \in \mathbb{R}$ and (2) Quadratic assignments: $v_j := \sum_{i \le k} a_{ik} v_i v_k + \sum_i b_i v_i + c$, with $a_{ik}, b_i, c \in \mathbb{R}$. Moreover, there can also be assignment sequences whose combined effect lies within the QGO class. The AbsEvolve framework therefore (1) merges such sequences into *blocks* for joint and more precise analysis, and (2) performs abstract interpretation on the transformed program.

Merging Assignment Sequences into Blocks. Frameworks like Clam analyze programs instruction-by-instruction, as implementing abstract transformers for entire sequences is challenging. In contrast, AbsEvolve uses a merging algorithm that first identifies assignment sequences $\hat{S}$ whose combined updates remain within the QGO class; that is, sequences whose Effective Update Map (Definition 4.1) is quadratic-bounded. Each such sequence is then merged into a single block $\mathcal{B}(\hat{S})$ for joint analysis. For example, as illustrated in Figure 4, AbsEvolve identifies two quadratic-bounded sub-sequences $\hat{S}_1$ and $\hat{S}_2$ within the larger sequence S, and merges them into blocks. Note that if all the assignments were merged together, the resulting EUM would have $\sigma(z) = (2 * a * b * c - a)$ which is cubic and violates the quadratic-bounded restriction. Hence, the merging algorithm splits the sequence. The algorithm also provides options such as only merging quadratic assignments or setting a maximum block length (detailed algorithm in Appendix B).

Abstract Interpretation on the Transformed Program. The merging of assignment sequences $\hat{S}$ into blocks $\mathcal{B}(\hat{S})$ in a program $\mathcal{P}$ yields a transformed program $\mathcal{P}'$. AbsEvolve performs

[1]https://github.com/uiuc-focal-lab/AbsEvolve
[2]https://github.com/seahorn/clam

abstract interpretation on this new program $\mathcal{P}'$. For affine assignments, quadratic assignments and blocks $\mathcal{B}(\hat{S})$, it uses our evolving transformer by first applying the UPOSE algorithm to construct a space of sound outputs and then using the AGG algorithm to search this space via gradient-descent and select a output. This search can be controlled by user-configurable parameters such as step size η and number of gradient steps R used by the AGG algorithm. Standard transformers from Elina are used for the rest of the instructions.

AbsEvolve analyzes the transformed program $\mathcal{P}'$ but the goal is to obtain invariants for the original program $\mathcal{P}$. We argue this is sound in two steps. Let $[\![\mathcal{P}]\!]_\ell$ denote the set of concrete states reachable at program point (label) ℓ in $\mathcal{P}$. First, for every concrete operator in $\mathcal{P}'$, the analysis uses a sound abstract transformer: our evolving transformer for assignments and blocks (sound by Thm. 4.1) and standard Elina transformers otherwise. So, at each program point ℓ, if the analysis computes abstract output a_l, then $[\![\mathcal{P}']\!]_\ell \subseteq \gamma(\mathsf{a}_l)$ holds. Next, by Theorem 6.1, we have for each program point ℓ that $[\![\mathcal{P}]\!]_\ell \subseteq [\![\mathcal{P}']\!]_\ell$. This, combined with the first step (per-step abstract soundness on $\mathcal{P}'$, i.e., $[\![\mathcal{P}']\!]_\ell \subseteq \gamma(\mathsf{a}_\ell)$), yields $[\![\mathcal{P}]\!]_\ell \subseteq \gamma(\mathsf{a}_\ell)$, thus proving the soundness of our analysis.

Theorem 6.1 (Soundness of Block Merging (per point)). *Let $\mathcal{P}'$ be obtained from $\mathcal{P}$ by replacing each assignment sequence $\hat{S}$ with a block $\mathcal{B}(\hat{S})$ whose concrete semantics is its Effective Update Map $\sigma_{\mathcal{B}(\hat{S})}$. Then, for every program point (label) ℓ, $[\![\mathcal{P}]\!]_\ell \subseteq [\![\mathcal{P}']\!]_\ell$.*

Proof Sketch. Using Theorem C.6 in Appendix, which proves that jointly analyzing a sequence via its EUM over-approximates step-wise semantics, we have $[\![\hat{S}]\!] \subseteq [\![\mathcal{B}(\hat{S})]\!]$ for all replaced sequences $\hat{S}$. As composition of semantics is monotonic, replacing an instruction by its over-approximation can only add new reachable states, and so ℓ, $[\![\mathcal{P}]\!]_\ell \subseteq [\![\mathcal{P}']\!]_\ell$. □

7 Evaluation

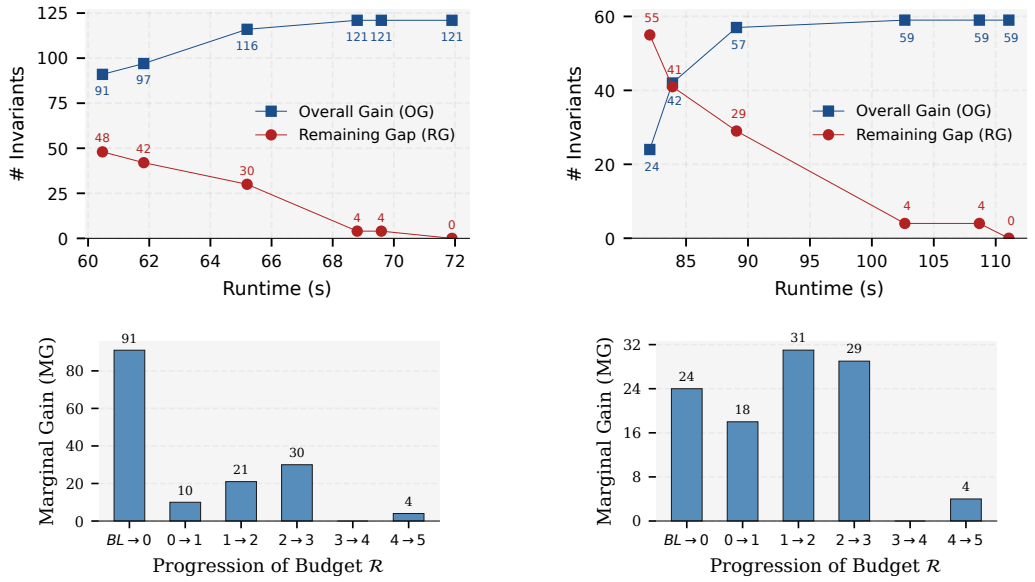
Research Questions. Our evaluation addresses two research questions that assess the key benefits of our evolving transformer: its ability to flexibly trade off precision and efficiency, achieve high precision efficiently, and generalize across domains and instructions.

- **RQ1:** Can AbsEvolve provide controllable precision–efficiency tradeoffs while also achieving the most precise invariants for linear operators efficiently?
- **RQ2:** Does AbsEvolve maintain this adaptability and precision for more complex cases such as quadratic operators and the Polyhedra domain?

Experimental Setup & Benchmarks. We use our AbsEvolve framework (Sec. 6) for experiments and compare the invariants computed by it with those computed by the state-of-the-art library Elina [48]. All experiments were run on a machine with a 16-core Intel(R) Core(TM) Ultra 7 255H CPU @ 1.10,GHz, 31 GiB RAM, and Ubuntu 24.04.3 LTS. As benchmarks, we use the 57 programs from the NLA-Digbench [37] suite of the SV-COMP benchmarks [5]. This is one of the largest suites of programs involving nonlinear numerical invariants and is widely used [15, 29, 38] to evaluate approaches that compute numerical invariants. These benchmarks include a wide range of instructions, such as affine and quadratic assignments and their sequences, which makes them challenging for standard analyses and also allows us to test how well AbsEvolve handles complex linear and nonlinear operators. We analyze these programs across three popular numerical domains, Zones, Octagons, and Polyhedra, and keep a timeout of 200s for each program.

7.1 Efficient and Adaptable Analysis for Linear Operators

As discussed in Section 6, our evolving transformer can handle both affine and quadratic operators and their sequences. However, obtaining ground-truth most-precise outputs for quadratic operators is infeasible and so for this section, we restrict AbsEvolve to only use our evolving transformer



(a) Zones (b) Octagons

Fig. 5. Evolution of invariant strengthening for Zones and Octagons (linear case) as the gradient-step budget $\mathcal{R}$ varies, showing how the analysis can be adapted across settings. Analysis with smaller $\mathcal{R}$ runs faster but is less precise, while larger $\mathcal{R}$ yields stronger invariants, with RG reaching 0 (most-precise invariants) at $\mathcal{R} = 5$.

for affine assignments and their sequences. This allows direct comparison with the most-precise outputs for affine operators. Also, since the Polyhedra domain already captures affine assignments exactly, we focus only on the Zones and Octagon domains in this section. For each domain, we first analyze all benchmark programs using Elina to compute the set of baseline invariants $\mathcal{I}_{bl}$, collected at the entry of each basic block across all programs (237 in total). This analysis takes around 20s for both domains. Next, we compute the *ground-truth* invariants $\mathcal{I}_{gt}$ obtained using the most-precise transformer for affine assignment sequences. For linear operators, this involves solving t linear programs (LPs) per transformer call (Sec. 4), where t is the number of template directions, which grows quadratically with the number of variables for Zones and Octagon domains. We use the LP-solving-based transformer [46] by directly using Gurobi [21] solver to solve these LPs efficiently. As discussed in Sec. 5, AbsEvolve jointly searches for bounds over all template directions, while the LP solver optimizes each direction independently. For fairness, we parallelize these LP calls across available threads. This method computes the set of most-precise *ground-truth* invariants $\mathcal{I}_{gt}$, taking 230s for Zones and 350s for Octagons. We also experimented with Symba [32], an SMT-based optimizer for Linear Real Arithmetic problems, but its reliance on SMT solving made it significantly slower, taking 43 minutes for the Zones domain (timing out on 11 programs) and 56 minutes for the Octagon domain (timing out on 12).

Monotonic Strengthening of Invariants. Next, we analyze the benchmark programs using AbsEvolve by varying the search budget $\mathcal{R}$ from 0 to 5 (where $\mathcal{R}$ is the number of gradient steps per transformer call) with $\mathcal{J}$ set to the precision objective $\mathcal{J}_{prec}$ (Section 5). Let $\mathcal{I}_r = \{a_r^1, a_r^2, \dots\}$ denote the invariants produced at budget r, where a_r^i is the i-th corresponding invariant in the set. These sets exhibit a monotonic strengthening structure: for all $r \geq 1$ and all i, we have

$\gamma(a_r^i) \subseteq \gamma(a_{r-1}^i)$, so invariants never become weaker as the budget increases; this property is enforced by the monotonicity of AGG discussed in Section 5. Since $r = 0$ performs no gradient steps, AGG returns the interval–relaxation baseline output used by standard libraries such as Elina, so one might expect $\mathcal{I}_0 = \mathcal{I}_{bl}$. However, because AbsEvolve performs sequence-level reasoning (rather than instruction-by-instruction as in the baseline), $\mathcal{I}_0$ often already strengthens several baseline invariants, i.e., $\gamma(a_0^i) \subseteq \gamma(a_{bl}^i)$. More generally, for all r and i, we have $\gamma(a_r^i) \subseteq \gamma(a_0^i) \subseteq \gamma(a_{bl}^i)$. To more precisely capture this strengthening behavior across budgets, we track the following metrics:

(1) **Overall Gain (OG)**: This counts how many invariants in $\mathcal{I}_r$ are stronger than their counterparts in $\mathcal{I}_{bl}$, i.e. $OG = \sum_{i=1}^{|\mathcal{I}_r|} \mathbf{I}(\gamma(\mathsf{a}_r^i) \subset \gamma(\mathsf{a}_{bl}^i))$, where $\mathbf{I}(\cdot)$ denotes the indicator function.
(2) **Remaining Gap (RG)**: This measures how many invariants in $\mathcal{I}_r$ are weaker than those in ground-truth set $\mathcal{I}_{gt}$ and gives a sense of gap from optimality: $RG = \sum_{i=1}^{|\mathcal{I}_r|} \mathbf{I}(\gamma(\mathsf{a}_{gt}^i) \subset \gamma(\mathsf{a}_r^i))$.
(3) **Marginal Gain (MG)**: OG provides a global view of total improvement over $\mathcal{I}_{bl}$ but can hide progress across budget settings, because once an invariant strengthens over the baseline, it contributes to OG only once, even if it is refined further at higher budgets. MG provides a local view by measuring how many invariants are strengthened at budget r relative to $r-1$, i.e., $MG = \sum_{i=1}^{|\mathcal{I}_r|} \mathbf{I}(\gamma(a_r^i) \subset \gamma(a_{r-1}^i))$. $\mathcal{I}_{r-1}$ is taken as $\mathcal{I}_{bl}$ when $r = 0$. MG shows how much additional refinement a one-step increase in search budget provides over the previous setting.

Adaptable Precision–Efficiency Analysis. Figure 5 shows the metrics defined above over different gradient-step budgets. The line charts show the runtimes for budgets $\mathcal{R} = 0$ to 5 (dots left to right) along with how the overall gains from baseline (OG) and the gap to the most-precise invariants (RG) evolve as the budget increases. The accompanying bar plots report the marginal gains obtained when increasing the budget from one setting to the next. As expected, larger budgets lead to higher runtimes since more gradient steps are performed within each transformer call. However, they also yield increasingly precise invariants, as reflected by rising OG and decreasing RG trends. Although OG flattens after $\mathcal{R} = 3$, the continued decrease in RG and non-zero marginal gains from $\mathcal{R} = 4$ to $\mathcal{R} = 5$ show that further strengthening still occurs on invariants already strengthened. Overall, these results show that the number of gradient-steps R provides an effective way to adapt the analysis by tuning the precision–efficiency tradeoff: larger budgets improve precision at higher cost, while smaller budgets offer faster but less precise analyses.

Computing the Most-Precise Invariants Efficiently. As demonstrated by Figure 5, at $\mathcal{R} = 5$ AbsEvolve achieves $RG = 0$ for both Zones and Octagons, meaning that the invariant set $\mathcal{I}_5$ matches the ground-truth most-precise set $\mathcal{I}_{gt}$.

Notably, AbsEvolve reaches the most-precise invariants in only *72s* for Zones and *111s* for Octagons, compared to *230s* and *350s* for the LP-solver-based transformer, making it about **3.2× faster**. As discussed in Sec. 4, the parametric output space contains the most-precise output for linear cases, and these results show that the AGG algorithm not only finds this output but does so efficiently. This efficiency arises because AGG searches for bounds jointly across all template directions using gradients, while the LP-solver-based transformer solves a separate optimization problem per direction. Figure 6 further confirms this, with the LP-based transformer's runtime growing steeply with the number of directions, while AbsEvolve 's increases much more gradually.

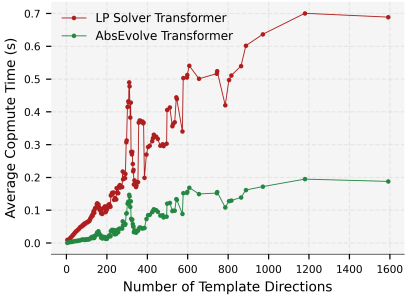
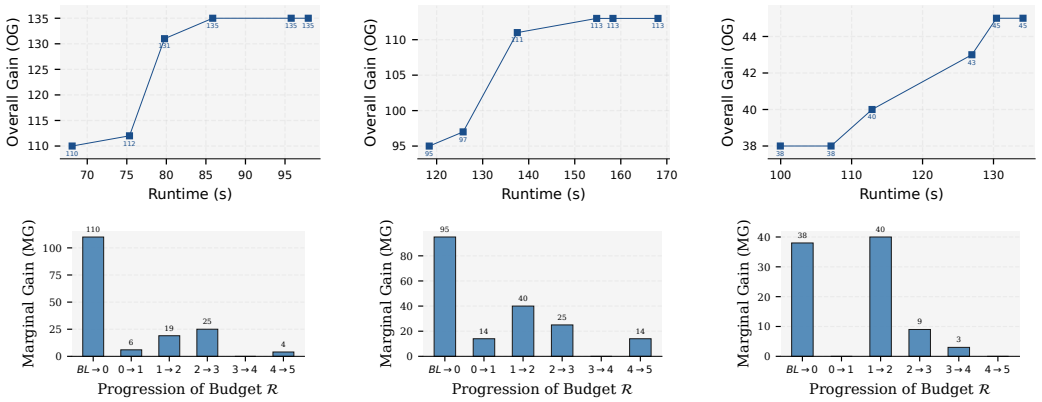


Fig. 6. Transformer Computer Time vs. Number of Template Directions



(a) Zones (b) Octagons (c) Polyhedra

Fig. 7. Evolution of invariant strengthening while handling quadratic assignments across domains.

7.2 Extending Analysis to Nonlinear Operators and Expressive Domains

Quadratic Assignment Sequences. Unlike in Section 7.1, where AbsEvolve was restricted to affine assignments for ground-truth comparison, we now use it in its full setting, allowing it to apply our evolving transformer to quadratic assignments and sequences as well, as discussed in Sec. 6. Since solvers like Gurobi are unsound for nonlinear operators [22] and Symba [32] supports only linear arithmetic, obtaining ground-truth outputs is infeasible in this setting. As before, we run AbsEvolve with gradient-step budgets varying from 0 to 5, but track only OG and MG metrics, as RG cannot be measured without ground truth. The increasing OG values in Figs. 7a and 7b, and the non-zero MG values ($4 \rightarrow 5$) even when OG plateaus, show that AbsEvolve continues to strengthen invariants as the budget increases, enabling adaptable analysis even for quadratic operators. Overall gain (OG) reaches 135 for Zones and 113 for Octagons, compared to 121 and 59 in the linear case, indicating that handling quadratic operators further strengthens invariants and yields more precise analyses.

Polyhedra Domain. Since Polyhedra is not a template-constraint (TCM) domain and supports arbitrary linear constraints, we configure AbsEvolve to use the Zones [36] template that tracks variable differences (AbsEvolve always generates outputs within a chosen template, Sec. 4). As Polyhedra can already represent linear updates exactly, AbsEvolve applies the evolving transformer only to quadratic assignments and assignment sequences with overall quadratic effects. Due to numerical overflows in Elina specific to the Polyhedra domain, we run experiments on a subset of 49 benchmarks (details in Appendix D.2), yielding a baseline invariant set $\mathcal{I}_{bl}$ of size 195. The results in Fig. 7c show that AbsEvolve enables precise and adaptable analysis even for expressive domains like Polyhedra, strengthening 45 (about 25%) of the baseline invariants by $\mathcal{R} = 5$. demonstrating the benefits of precise (and sequence-level) reasoning for quadratic operators.

Runtime Trends Across Domains. As seen in Fig. 7, the analysis time does not grow linearly with the gradient-step budget $\mathcal{R}$. This is expected, since $\mathcal{R}$ controls only the transformer outputs, while the total runtime depends on the full sequence of all transformer calls during the analysis. Moreover, as the output space is complex, the search may get stuck in local minima and produce identical outputs across consecutive budget settings. Nevertheless, our gradient-guided traversal is generally effective at escaping such plateaus. For instance, in Zones and Octagons (Figs. 7a and 7b),

budgets $\mathcal{R}$=3 and $\mathcal{R}$=4 yield identical results, but precision improves at $\mathcal{R}$=5, as indicated by the non-zero MG from $\mathcal{R}$=4 to $\mathcal{R}$=5. A similar pattern appears for Polyhedra (Fig. 7c), where progress resumes after stagnation between $\mathcal{R}$=0 and $\mathcal{R}$=1.

Efficiency Comparison with Elina. Analyzing the benchmarks with Elina takes approximately 20 seconds (18.53s for Zones, 20.11s for Octagons, and 23.2s for Polyhedra). In contrast, as shown above, AbsEvolve incurs higher runtime, ranging from 70 to 170 seconds as the number of gradient steps increases from 0 to 5. This overhead is expected, as AbsEvolve consistently computes more precise invariants than Elina. Importantly, AbsEvolve enables users to navigate the precision–runtime tradeoff by selecting configurations tailored to their needs, and as discussed in the last section, it even outperforms the industrial-grade, heavily optimized LP solver Gurobi based transformer in computing the most precise invariants. Moreover, for a fair comparison on standard benchmarks, we implement AbsEvolve within the Clam/Elina framework, incurring integration overhead due to conversions between ELINA's internal abstract states and the tensor-based representation used for gradient-based optimization (≈50% of runtime overhead). This cost is not inherent and can be reduced with additional engineering effort by implementing a native abstract domain tailored to AbsEvolve-style parametric transformers.

In terms of memory, AbsEvolve introduces only modest overhead, as it does not explicitly compute all abstract outputs, but instead efficiently represents the space of sound abstract outputs *symbolically* via parameters. In our evaluation, the median memory usage increases from 0.08 GB to 0.14 GB (max 0.23 GB), corresponding to well under 1% of memory on our 32 GB machine (≈0.7% at peak), making the footprint negligible. This increase is expected, as our PyTorch [40]-based gradient descent relies on tensor computations that consume some additional memory.

Ineffectiveness of Random Sampling. The results in Fig. 5 and Fig. 7 show the efficacy of our AGG search in discovering more precise invariants as the budget $\mathcal{R}$ increases. As discussed in Sec. 5, a naive search strategy can be to randomly sample parameters from Θ. We evaluate this using rejection sampling, where we aim to obtain r feasible points for budget $\mathcal{R} = r$, ideally drawing samples for each point until a feasible one is found. However, since Θ is a high-dimensional polyhedron defined by complex linear constraints, most randomly drawn points are either infeasible or far from the high-scoring regions where precise outputs lie, and satisfying these constraints by chance could require arbitrarily many attempts. To match AGG's runtime, we cap this at 5 attempts per point, accepting the first feasible sample found and falling back to the interval relaxation ($\lambda = 0$) otherwise. This approach yields *zero improvement* over $\mathcal{R} = 0$, confirming that unguided search fails regardless of budget. This highlights the importance of our adaptive gradient-guided search, which jointly manages feasibility and objective-awareness by adaptively switching between optimizing $\mathcal{J}$ when feasible and restoring feasibility otherwise, efficiently navigating Θ toward precise outputs even with small budgets.

Precision Benefits from Sequence-Level Reasoning. The results in Fig. 7 use our standard sequence-level analysis, where the instruction sequences supported by AbsEvolve are analyzed jointly. This joint handling avoids the precision loss incurred when composing per-instruction transformers and is a key contributor to the gains we observe. As shown in Appendix D.3, disabling sequence-level handling leads to substantially fewer strengthened invariants.

8 Related Works

Manually designing efficient abstract transformers is both challenging and tedious and has motivated substantial work on automating their construction [6, 19, 27, 28, 30, 41, 43]. Synthesis-based approaches such as [27, 28] use program synthesis over a user-provided DSL to generate transformers for specific domains such as strings. Data-driven techniques like [30, 41] train neural networks to

serve as abstract transformers for specific non-linear operations, but these methods remain limited to bounded abstract elements. Neither of these directions applies to expressive relational domains like Octagons [35] and Polyhedra [14], where abstract elements may be unbounded and involve many numeric constraints, making properties like soundness difficult to enforce. This is evident in [19], which attempts to learn transformers for such domains, but the trained models have no soundness guarantees, making them unsuitable for static analysis. Another line of work [26, 32, 49, 53] uses symbolic abstraction [42] to compute most-precise transformers for these domains, not only for individual instructions but also for entire sequences, thus enabling more precise analysis than per-instruction reasoning. However, these methods typically rely on expensive solver calls, which limits their scalability. Our evolving abstract transformer addresses these limitations, as it is general and not tied to any particular domain or instruction. The same transformer works for all supported domains and operators, while also enabling efficient sequence-level reasoning.

While symbolic optimization yields precise transformers and works such as [53] improve its efficiency, it remains solver-dependent and not general across domains, causing libraries to default to faster but less precise transformers regardless of the analysis task, which is limiting. Prior work [8, 34, 51] has demonstrated the benefits of tuning general analyzer parameters for specific downstream goals, and [9, 23] show the value of adapting analyses to time and memory budgets of the task. However, these approaches act only at the analyzer level and do not provide fine-grained control over the precision–efficiency trade-offs of individual transformers for varied scenarios. Our evolving abstract transformer addresses this by moving from computing a single fixed output to searching within a parametric space of sound outputs using an efficient gradient descent-based method that adapts to the task. For abstract interpretation, to the best of our knowledge, gradient-based methods have been used only in neural network verification settings [2, 16, 50, 52], where they are applied to hand-crafted parametric transformers for specific operators such as ReLU over bounded interval domains. These settings are simpler, and extending gradient-based methods to complex unbounded domains is substantially more difficult. Our formulation achieves this by transforming efficient yet unreliable first-order optimization methods such as gradient descent into a reliable mechanism with sound-by-construction guarantees, by using them within a parametric space of sound outputs to identify more precise outputs for instructions and instruction sequences.

9 Conclusion

We introduce the AbsEvolve framework, built on our Evolving Abstract Transformer, addressing the fundamental limitation of existing transformers that have fixed imprecision, are non-adaptable, and must be manually designed per domain and operator. AbsEvolve uses our UPOSE algorithm to generate a parametric space of sound outputs for a broad class of domains and operators, and our AGG algorithm to efficiently search this space for outputs best suited to the analysis objectives. This reformulation into an efficient search problem, combined with gradient-based optimization while preserving soundness, enables precise and adaptable analysis where the number of gradient steps can be adjusted to flexibly trade precision for efficiency. Across domains, AbsEvolve achieves up to 3.2× faster convergence to the most precise invariants compared to existing baselines. This work opens up promising directions for accelerator-driven program analysis on modern hardware such as GPUs, as well as learning-based approaches to abstract interpretation.

ACKNOWLEDGEMENTS

We thank the anonymous reviewers for their insightful comments, which greatly helped us improve the clarity and presentation of this work. This work was supported in part by a grant from the Amazon Illinois Center on AI for Interactive Conversational Experiences (AICE), NSF Grants No. CCF-2238079, CCF-2316233, CNS-2148583, and NAIRR240476, an Open Philanthropy research grant, and compute resources from Indiana Jetstream2.

DATA AVAILABILITY STATEMENT

The source code and evaluation data used for the experiments in this paper are publicly available on Zenodo [18] (DOI: 10.5281/zenodo.19079444), with instructions to reproduce the results. The latest version of the framework is maintained on GitHub at https://github.com/uiuc-focal-lab/AbsEvolve.

References

[1] Roberto Bagnara, Patricia M. Hill, and Enea Zaffanella. 2008. The Parma Polyhedra Library: Toward a complete set of numerical abstractions for the analysis and verification of hardware and software systems. *Sci. Comput. Program.* 72, 1–2 (June 2008), 3–21. doi:10.1016/j.scico.2007.08.001

[2] Debangshu Banerjee and Gagandeep Singh. 2024. Relational DNN Verification With Cross Executional Bound Refinement. In *Proceedings of the 41st International Conference on Machine Learning (ICML)*. 2779–2807. https://arxiv.org/abs/2405.10143

[3] Dimitri P. Bertsekas. 1999. *Nonlinear Programming* (2nd ed.). Athena Scientific.

[4] Dirk Beyer. 2016. Reliable and Reproducible Competition Results with BenchExec and Witnesses Report on SV-COMP 2016. In *Proceedings of the 22nd International Conference on Tools and Algorithms for the Construction and Analysis of Systems - Volume 9636*. Springer-Verlag, Berlin, Heidelberg, 887–904. doi:10.1007/978-3-662-49674-9_55

[5] Dirk Beyer and Jan Strejček. 2025. Improvements in Software Verification and Witness Validation: SV-COMP 2025. In *Proceedings of the 31st International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS) (Lecture Notes in Computer Science, Vol. 15698)*. Springer, 151–186. doi:10.1007/978-3-031-90660-2_9

[6] Pavol Bielik, Veselin Raychev, and Martin Vechev. 2017. Learning a Static Analyzer from Data. In *Proceedings of the 29th International Conference on Computer Aided Verification (CAV) (Lecture Notes in Computer Science, Vol. 10426)*. Springer, 233–253. doi:10.1007/978-3-319-63387-9_12

[7] Stephen Boyd and Lieven Vandenberghe. 2004. *Convex Optimization*. Cambridge University Press. https://web.stanford.edu/~boyd/cvxbook/

[8] Sooyoung Cha, Myungho Lee, Seokhyun Lee, and Hakjoo Oh. 2022. SYMTUNER: Maximizing the Power of Symbolic Execution by Adaptively Tuning External Parameters. In *2022 IEEE/ACM 44th International Conference on Software Engineering (ICSE)*. 2068–2079. doi:10.1145/3510003.3510185

[9] Maria Christakis and Valentin Wüstholz. 2016. Bounded Abstract Interpretation. In *Proceedings of the 23rd International Static Analysis Symposium (SAS) (Lecture Notes in Computer Science, Vol. 9837)*. Springer, 105–125. doi:10.1007/978-3-662-53413-7_6

[10] Robert Clarisó and Jordi Cortadella. 2007. The Octahedron Abstract Domain. *Science of Computer Programming* 64, 1 (2007), 115–139. doi:10.1016/j.scico.2006.03.009

[11] Patrick Cousot and Radhia Cousot. 1977. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Proceedings of the 4th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages* (Los Angeles, California) *(POPL '77)*. Association for Computing Machinery, New York, NY, USA, 238–252. doi:10.1145/512950.512973

[12] Patrick Cousot and Radhia Cousot. 1977. Static determination of dynamic properties of generalized type unions. In *Proceedings of an ACM Conference on Language Design for Reliable Software* (Raleigh, North Carolina). Association for Computing Machinery, New York, NY, USA, 77–94. doi:10.1145/800022.808314

[13] Patrick Cousot, Radhia Cousot, Jerôme Feret, Laurent Mauborgne, Antoine Miné, David Monniaux, and Xavier Rival. 2005. The ASTREÉ analyzer. In *Proceedings of the 14th European Conference on Programming Languages and Systems* (Edinburgh, UK) *(ESOP'05)*. Springer-Verlag, Berlin, Heidelberg, 21–30. doi:10.1007/978-3-540-31987-0_3

[14] Patrick Cousot and Nicolas Halbwachs. 1978. Automatic discovery of linear restraints among variables of a program. In *Proceedings of the 5th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages* (Tucson, Arizona) *(POPL '78)*. Association for Computing Machinery, New York, NY, USA, 84–96. doi:10.1145/512760.512770

[15] John Cyphert and Zachary Kincaid. 2024. Solvable Polynomial Ideals: The Ideal Reflection for Program Analysis. *Proc. ACM Program. Lang.* 8, POPL, Article 25 (Jan. 2024), 29 pages. doi:10.1145/3632867

[16] Claudio Ferrari, Mark Niklas Mueller, Nikola Jovanović, and Martin Vechev. 2022. Complete Verification via Multi-Neuron Relaxation Guided Branch-and-Bound. In *International Conference on Learning Representations*. https://openreview.net/forum?id=l_amHf1oaK

[17] Marc Fischer, Mislav Balunovic, Dana Drachsler-Cohen, Timon Gehr, Ce Zhang, and Martin Vechev. 2019. DL2: Training and Querying Neural Networks with Logic. In *Proceedings of the 36th International Conference on Machine Learning (ICML) (Proceedings of Machine Learning Research, Vol. 97)*. PMLR, 1931–1941. https://proceedings.mlr.press/v97/fischer19a.html

[18] Shaurya Gomber, Debangshu Banerjee, and Gagandeep Singh. 2026. AbsEvolve: Artifact for "Evolving Abstract Transformers for Gradient-Guided, Adaptable Abstract Interpretation". doi:10.5281/zenodo.19079444

[19] Shaurya Gomber and Gagandeep Singh. 2025. Neural Abstract Interpretation. In *ICLR 2025 Workshop: VerifAI: AI Verification in the Wild*. https://openreview.net/forum?id=WTyyhWhp4m

[20] Arie Gurfinkel, Temesghen Kahsai, Anvesh Komuravelli, and Jorge A. Navas. 2015. The SeaHorn Verification Framework. In *Proceedings of the 27th International Conference on Computer Aided Verification (CAV) (Lecture Notes in Computer Science, Vol. 9206)*. Springer, 343–361. doi:10.1007/978-3-319-21690-4_20

[21] Gurobi Optimization, LLC. 2024. Gurobi Optimizer Reference Manual. https://www.gurobi.com

[22] Gurobi Support. 2024. Non-Convex MIQP with Bilinear Terms. https://support.gurobi.com/hc/en-us/community/posts/29050048200593/comments/29055912559889. Accessed April 2025.

[23] Kihong Heo, Hakjoo Oh, and Hongseok Yang. 2019. Resource-aware program analysis via online abstraction coarsening. In *Proceedings of the 41st International Conference on Software Engineering* (Montreal, Quebec, Canada) *(ICSE '19)*. IEEE Press, 94–104. doi:10.1109/ICSE.2019.00027

[24] Jacob M. Howe and Andy King. 2009. Logahedra: A New Weakly Relational Domain. In *Proceedings of the 7th International Symposium on Automated Technology for Verification and Analysis (ATVA) (Lecture Notes in Computer Science, Vol. 5799)*. Springer, Berlin, Heidelberg, 306–320. doi:10.1007/978-3-642-04761-9_23

[25] Bertrand Jeannet and Antoine Miné. 2009. Apron: A Library of Numerical Abstract Domains for Static Analysis. In *Proceedings of the 21st International Conference on Computer Aided Verification (CAV) (Lecture Notes in Computer Science, Vol. 5643)*. Springer, Berlin, Heidelberg, 661–667. doi:10.1007/978-3-642-02658-4_52

[26] Jiahong Jiang, Liqian Chen, Xueguang Wu, and Ji Wang. 2017. Block-Wise Abstract Interpretation by Combining Abstract Domains with SMT. In *Proceedings of the 18th International Conference on Verification, Model Checking, and Abstract Interpretation (VMCAI) (Lecture Notes in Computer Science, Vol. 10145)*. Springer, Cham, 310–329. doi:10.1007/978-3-319-52234-0_17

[27] Pankaj Kumar Kalita, Sujit Kumar Muduli, Loris D'Antoni, Thomas Reps, and Subhajit Roy. 2022. Synthesizing abstract transformers. *Proc. ACM Program. Lang.* 6, OOPSLA2, Article 171 (Oct. 2022), 29 pages. doi:10.1145/3563334

[28] Pankaj Kumar Kalita, Thomas Reps, and Subhajit Roy. 2025. Synthesizing Abstract Transformers for Reduced-Product Domains. In *Proceedings of the 31st International Static Analysis Symposium (SAS) (Lecture Notes in Computer Science, Vol. 14995)*. Springer, Cham, 147–172. doi:10.1007/978-3-031-74776-2_6

[29] Zachary Kincaid, Nicolas Koh, and Shaowei Zhu. 2023. When Less Is More: Consequence-Finding in a Weak Theory of Arithmetic. *Proceedings of the ACM on Programming Languages* 7, POPL, Article 44 (2023), 1275–1307 pages. doi:10.1145/3571237

[30] Jacob Laurel, Siyuan Qian, Gagandeep Singh, and Sasa Misailovic. 2023. Synthesizing Precise Static Analyzers for Automatic Differentiation. *Proceedings of the ACM on Programming Languages* 7 (10 2023), 1964–1992. doi:10.1145/3622867

[31] Vincent Laviron and Francesco Logozzo. 2009. SubPolyhedra: A (More) Scalable Approach to Infer Linear Inequalities. In *Proceedings of the 10th International Conference on Verification, Model Checking, and Abstract Interpretation (VMCAI) (Lecture Notes in Computer Science, Vol. 5403)*. Springer, Berlin, Heidelberg, 229–244. doi:10.1007/978-3-540-93900-9_20

[32] Yi Li, Aws Albarghouthi, Zachary Kincaid, Arie Gurfinkel, and Marsha Chechik. 2014. Symbolic optimization with SMT solvers. In *Proceedings of the 41st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages* (San Diego, California, USA) *(POPL '14)*. Association for Computing Machinery, New York, NY, USA, 607–618. doi:10.1145/2535838.2535857

[33] Francesco Logozzo and Manuel Fähndrich. 2008. Pentagons: A Weakly Relational Abstract Domain for the Efficient Validation of Array Accesses. In *Proceedings of the 23rd ACM Symposium on Applied Computing (SAC)*. ACM, New York, NY, USA, 184–188. doi:10.1145/1363686.1363736

[34] Muhammad Numair Mansur, Benjamin Mariano, Maria Christakis, Jorge A. Navas, and Valentin Wüstholz. 2021. Automatically Tailoring Abstract Interpretation to Custom Usage Scenarios. In *Computer Aided Verification: 33rd International Conference, CAV 2021, Virtual Event, July 20–23, 2021, Proceedings, Part II*. Springer-Verlag, Berlin, Heidelberg, 777–800. doi:10.1007/978-3-030-81688-9_36

[35] A. Mine. 2001. The octagon abstract domain. In *Proceedings Eighth Working Conference on Reverse Engineering*. 310–319. doi:10.1109/WCRE.2001.957836

[36] Antoine Miné. 2002. A Few Graph-Based Relational Numerical Abstract Domains. In *Proceedings of the 9th International Static Analysis Symposium (SAS) (Lecture Notes in Computer Science, Vol. 2477)*. Springer, Berlin, Heidelberg, 117–132. doi:10.1007/3-540-45789-5_11

[37] Thanhvu Nguyen, Deepak Kapur, Westley Weimer, and Stephanie Forrest. 2014. DIG: A Dynamic Invariant Generator for Polynomial and Array Invariants. *ACM Trans. Softw. Eng. Methodol.* 23, 4, Article 30 (Sept. 2014), 30 pages. doi:10.1145/2556782

[38] ThanhVu Nguyen, KimHao Nguyen, and Hai Duong. 2022. SymInfer: inferring numerical invariants using symbolic states. In *Proceedings of the ACM/IEEE 44th International Conference on Software Engineering: Companion Proceedings* (Pittsburgh, Pennsylvania) *(ICSE '22)*. Association for Computing Machinery, New York, NY, USA, 197–201. doi:10.1145/3510454.3516833

[39] Jorge Nocedal and Stephen J. Wright. 2006. *Numerical Optimization* (2nd ed.). Springer, New York, NY. doi:10.1007/978-0-387-40065-5

[40] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *Advances in Neural Information Processing Systems 32 (NeurIPS)*. Curran Associates Inc., 8024–8035. https://proceedings.neurips.cc/paper/2019/hash/bdbca288fee7f92f2bfa9f7012727740-Abstract.html

[41] Brandon Paulsen and Chao Wang. 2022. LinSyn: Synthesizing Tight Linear Bounds for Arbitrary Neural Network Activation Functions. In *Proceedings of the 28th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS) (Lecture Notes in Computer Science, Vol. 13243)*. Springer, Cham, 357–376. doi:10.1007/978-3-030-99524-9_19

[42] Thomas Reps, Mooly Sagiv, and Greta Yorsh. 2004. Symbolic Implementation of the Best Transformer. In *Proceedings of the 5th International Conference on Verification, Model Checking, and Abstract Interpretation (VMCAI) (Lecture Notes in Computer Science, Vol. 2937)*. Springer, Berlin, Heidelberg, 252–266. doi:10.1007/978-3-540-24622-0_21

[43] Thomas Reps and Aditya Thakur. 2016. Automating Abstract Interpretation. In *Proceedings of the 17th International Conference on Verification, Model Checking, and Abstract Interpretation - Volume 9583* (St. Petersburg, FL, USA) *(VMCAI 2016)*. Springer-Verlag, Berlin, Heidelberg, 3–40. doi:10.1007/978-3-662-49122-5_1

[44] Joao Rivera, Franz Franchetti, and Markus Püschel. 2024. Floating-Point TVPI Abstract Domain. *Proc. ACM Program. Lang.* 8, PLDI, Article 165 (June 2024), 25 pages. doi:10.1145/3656395

[45] Mooly Sagiv, Thomas Reps, and Reinhard Wilhelm. 1999. Parametric shape analysis via 3-valued logic. In *Proceedings of the 26th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages* (San Antonio, Texas, USA) *(POPL '99)*. Association for Computing Machinery, New York, NY, USA, 105–118. doi:10.1145/292540.292552

[46] Sriram Sankaranarayanan, Henny B. Sipma, and Zohar Manna. 2005. Scalable Analysis of Linear Systems Using Mathematical Programming. In *Proceedings of the 6th International Conference on Verification, Model Checking, and Abstract Interpretation (VMCAI) (Lecture Notes in Computer Science, Vol. 3385)*. Springer, Berlin, Heidelberg, 25–41. doi:10.1007/978-3-540-30579-8_2

[47] Gagandeep Singh, Markus Püschel, and Martin T. Vechev. 2015. Making Numerical Program Analysis Fast. In *Proceedings of the 36th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*. ACM, 303–313. doi:10.1145/2737924.2738000

[48] Gagandeep Singh, Markus Püschel, and Martin T. Vechev. 2017. Fast Polyhedra Abstract Domain. In *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages (POPL)*. ACM, 46–59. doi:10.1145/3009837.3009885

[49] A. Thakur, A. Lal, J. Lim, and T. Reps. 2015. PostHat and All That. *Electron. Notes Theor. Comput. Sci.* 311, C (Feb. 2015), 15–32. doi:10.1016/j.entcs.2015.02.003

[50] Shiqi Wang, Huan Zhang, Kaidi Xu, Xue Lin, Suman Jana, Cho-Jui Hsieh, and J Zico Kolter. 2021. Beta-CROWN: Efficient Bound Propagation with Per-neuron Split Constraints for Neural Network Robustness Verification. In *ICML 2021 Workshop on Adversarial Machine Learning*. https://openreview.net/forum?id=Mm3gxxTfT7A

[51] Zhongyi Wang, Linyu Yang, Mingshuai Chen, Yixuan Bu, Zhiyang Li, Qiuye Wang, Shengchao Qin, Xiao Yi, and Jianwei Yin. 2024. Parf: Adaptive Parameter Refining for Abstract Interpretation. In *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. ACM, 1082–1093. doi:10.1145/3691620.3695487

[52] Kaidi Xu, Huan Zhang, Shiqi Wang, Yihan Wang, Suman Jana, Xue Lin, and Cho-Jui Hsieh. 2021. Fast and Complete: Enabling Complete Neural Network Verification with Rapid and Massively Parallel Incomplete Verifiers. In *International Conference on Learning Representations*. https://openreview.net/forum?id=nVZtXBI6LNn

[53] Peisen Yao, Qingkai Shi, Heqing Huang, and Charles Zhang. 2021. Program analysis via efficient symbolic abstraction. *Proc. ACM Program. Lang.* 5, OOPSLA, Article 118 (Oct. 2021), 32 pages. doi:10.1145/3485495

A Duality-based construction of Sound Parametric Scalar Map

Our algorithm to generate a parametric family of sound transformers relies on finding a sound Parametric Scalar Map (PSM) (Def 4.2) for the optimal value $o_{\min}$ of optimization problems of the form:

$$o_{\min} = \min_{\mathbf{v}\in\mathbb{R}^n} f(\mathbf{v}) \quad \text{s.t.} \quad A\,\mathbf{v} \le \mathbf{b} \tag{3}$$

where the objective function $f(\mathbf{v})$ is of the form:

$$f(\mathbf{v}) = \sum_{1\le i<k\le n} H_{ik}\, v_i v_k + \sum_{i=1}^{n} Q_i\, v_i^2 + \mathbf{c}^\top \mathbf{v} + d \tag{4}$$

Here, $H_{ik}\, v_i v_k$ denotes the bilinear terms (products of distinct variables), $Q_i\, v_i^2$ denotes the quadratic terms, $c_i\, v_i$ denotes the linear terms, and $d \in \mathbb{R}$ is a constant.

Soundness criterion. A scalar ℓ is a *sound lower bound* on the optimal value $o_{\min}$ whenever $\ell \le o_{\min}$. We obtain a sound Parametric Scalar Map for Eq 3 through the following steps.

A.1 Step 1: Efficient Dual Construction

As our goal is to obtain certified *lower bounds* on the primal optimum $o_{\min}$ for the problem in Eq 3, we turn to its Lagrangian dual, because weak duality guarantees that any feasible value of the dual function is a lower bound. Introducing non-negative multipliers $\boldsymbol{\lambda} \ge 0$ for the constraints $A\mathbf{v} \le \mathbf{b}$ in the problem in Eq 3 yields the following re-formulation of the same problem:

$$o_{\min} = \min_{\mathbf{v}\in\mathbb{R}^n} \max_{\boldsymbol{\lambda}\ge 0} \big(f(\mathbf{v}) + \boldsymbol{\lambda}^\top (A\mathbf{v} - \mathbf{b})\big)$$

Swapping the order of optimization produces the dual problem

$$o_{\text{dual}} = \max_{\boldsymbol{\lambda}\ge 0} \underbrace{\min_{\mathbf{v}\in\mathbb{R}^n} \big(f(\mathbf{v}) + \boldsymbol{\lambda}^\top (A\mathbf{v} - \mathbf{b})\big)}_{g(\boldsymbol{\lambda})}$$

where $g(\boldsymbol{\lambda})$ is the *dual function*. By weak duality

$$g(\boldsymbol{\lambda}) \le o_{\min} \quad \text{for every } \boldsymbol{\lambda} \ge 0$$

Consequently, if we can compute the inner minimization $g(\boldsymbol{\lambda})$ for a given $\boldsymbol{\lambda} \ge 0$, then inserting *any* such multiplier values produces a sound lower bound, thus giving us the desired parametric family of bounds.

Inefficiency of the naive dual construction. Before evaluating the dual function $g(\boldsymbol{\lambda})$, note that constructing the dual from the entire constraint set $A\mathbf{v} \le \mathbf{b}$ is unnecessarily costly. The abstract input element—like most numerical abstract domains—already contains tight *box constraints* $l_i \le v_i \le u_i$. Including these bounds in the dual construction introduces two Lagrange multipliers per variable, inflating the dual vector and increasing the cost of every evaluation of g. A standard remedy is to *peel off* the box constraints first: one rewrites the feasible region as $\mathbf{v} \in [l, u]$ together with the remaining coupled inequalities, and constructs the dual only for the latter. The inner minimisation then becomes a box-constrained problem, which typically admits a closed-form solution or a very cheap projection step [3, Ch. 1]; [39, Ch. 17]. This strategy keeps the dual compact, reduces the cost of computing $g(\boldsymbol{\lambda})$, and accelerates the subsequent maximisation over $\boldsymbol{\lambda} \ge 0$.

So, we isolate tight lower and upper bounds for each variable from the constraint system $A\mathbf{v} \le \mathbf{b}$. This yields a bounding box $\mathbf{v} \in [\mathbf{l}, \mathbf{u}]$, where $l_i \le v_i \le u_i$ for each i. (Some bounds may be unbounded, i.e. $l_i = -\infty$ or $u_i = +\infty$, depending on the original constraints.) Removing these trivial bounds from

the coupled inequalities leaves a reduced system $A'\mathbf{v} \leq \mathbf{b}'$. The optimization problem is therefore equivalent to

$$o_{\min} \;=\; \min_{\mathbf{v}\in[\mathbf{l},\mathbf{u}]} \; f(\mathbf{v}) \quad \text{s.t.} \quad A'\mathbf{v} \leq \mathbf{b}' \tag{5}$$

where the box constraint is handled explicitly and only the coupled inequalities are retained in $A'\mathbf{v} \leq \mathbf{b}'$. Introducing non-negative multipliers $\boldsymbol{\lambda} \geq 0$ for the constraints $A'\mathbf{v} \leq \mathbf{b}'$ in the problem in Eq 5 yields the following re-formulation:

$$o_{\min} \;=\; \min_{\mathbf{v}\in[\mathbf{l},\mathbf{u}]} \; \max_{\boldsymbol{\lambda}\geq 0} \big(f(\mathbf{v}) + \boldsymbol{\lambda}^\top (A'\mathbf{v} - \mathbf{b}')\big) \tag{6}$$

Swapping the order of optimization then produces the dual problem

$$o_{\text{dual}} \;=\; \max_{\boldsymbol{\lambda}\geq 0} \underbrace{\min_{\mathbf{v}\in[\mathbf{l},\mathbf{u}]} \big(f(\mathbf{v}) + \boldsymbol{\lambda}^\top (A'\mathbf{v} - \mathbf{b}')\big)}_{g(\boldsymbol{\lambda})} \tag{7}$$

where $g(\boldsymbol{\lambda})$ is the *dual function*. Now, as discussed above, by weak duality, every value $g(\boldsymbol{\lambda})$ with $\boldsymbol{\lambda} \geq 0$ is a certified lower bound on $o_{\min}$. Henceforth, we focus on evaluating this dual function efficiently: once we can compute $g(\boldsymbol{\lambda})$ for any chosen multiplier, we immediately obtain an entire parametric family of sound bounds.

Substituting $f(\mathbf{v})$ from Eq. (4):

$$g(\boldsymbol{\lambda}) \;=\; \min_{\mathbf{v}\in[\mathbf{l},\mathbf{u}]} \left(\sum_{1\leq i<k\leq n} H_{ik}\, v_i v_k \;+\; \sum_{i=1}^{n} Q_i\, v_i^2 \;+\; (\mathbf{c} + A'^T\lambda)^T\mathbf{v} \;+\; d - \lambda^T\mathbf{b}' \right)$$

Note, however, that this inner problem is still far from trivial: even with the simple box constraint, the presence of arbitrary bilinear terms $H_{ik}v_iv_k$ and potentially indefinite quadratic coefficients Q_i turns the minimisation into a general (and possibly non-convex) quadratic programme, so an exact closed-form solution for $g(\boldsymbol{\lambda})$ is not available in general.

A.2 Step 2: Sound inner minimisation via linear coefficient splitting

We now discuss how to make the inner minimisation problem *tractable*. The idea is to redistribute each linear coefficient between its associated quadratic and bilinear terms so that the objective can be broken into one– and two–variable "boxes" that can be optimised independently.

Step 1: Introduce splitting parameters. Pick auxiliary vectors $\mathbf{S} \in \mathbb{R}^n$ and $\mathbf{D} \in \mathbb{R}^{n\times n}$ and rewrite $g(\boldsymbol{\lambda})$ as

$$\begin{aligned} g(\boldsymbol{\lambda}) = g(\boldsymbol{\lambda}, \mathbf{S}, \mathbf{D}) = \min_{\mathbf{v}\in[\mathbf{l},\mathbf{u}]} \Bigg(& \sum_{1\leq i<k\leq n} \big(H_{ik}v_iv_k + D^i_{ik}v_i + D^k_{ik}v_k\big) \\ & + \sum_{i=1}^{n} \big(Q_i v_i^2 + S_i v_i\big) \\ & + \big(\mathbf{c} + A^\top\lambda - \mathbf{S} - \boldsymbol{\Delta}\big)^\top\mathbf{v} + d - \lambda^\top\mathbf{b}\Bigg), \end{aligned}$$

where $\Delta_i := \sum_{k\neq i} D^i_{ik}$ and $\boldsymbol{\Delta} := (\Delta_1, \ldots, \Delta_n)^\top$. For any fixed $(\mathbf{S}, \mathbf{D})$ this is algebraically identical to the original $g(\boldsymbol{\lambda})$; only the grouping of the linear terms has changed.

Step 2: Minimise each box separately. Splitting the objective term-wise yields the relaxed value

$$\begin{aligned}\hat{g}(\boldsymbol{\lambda}, \mathbf{S}, \mathbf{D}) = & \sum_{1 \le i < k \le n} \min_{(v_i, v_k) \in [l_i, u_i] \times [l_k, u_k]} \left(H_{ik} v_i v_k + D^i_{ik} v_i + D^k_{ik} v_k \right) \\ & + \sum_{i=1}^{n} \min_{v_i \in [l_i, u_i]} \left(Q_i v_i^2 + S_i v_i \right) \\ & + \min_{\mathbf{v} \in [\mathbf{l}, \mathbf{u}]} \left(\mathbf{c} + A^\top \lambda - \mathbf{S} - \Delta \right)^\top \mathbf{v} + d - \lambda^\top \mathbf{b}. \end{aligned} \tag{8}$$

Step 3: Soundness of the relaxation. For any family $\{h_j\}$ on a common domain $\mathcal{X}$,

$$\min_{x \in \mathcal{X}} \sum_j h_j(x) \;\ge\; \sum_j \min_{x \in \mathcal{X}} h_j(x), \tag{9}$$

so applying (9) to (8) gives

$$\hat{g}(\boldsymbol{\lambda}, \mathbf{S}, \mathbf{D}) \;\le\; g(\boldsymbol{\lambda}) \;\le\; o_{\min}, \qquad \text{for all } \boldsymbol{\lambda} \ge 0,\ \mathbf{S},\ \mathbf{D}. \tag{10}$$

Intuition. The split yields $n(n-1)/2$ two-variable boxes, n single-variable boxes, and one residual linear box. Each box now contains only a single quadratic, bilinear, or linear term, so every subproblem is straightforward to minimise. Summing their minima gives the relaxed value $\hat{g}(\boldsymbol{\lambda}, \mathbf{S}, \mathbf{D})$, which remains a sound lower bound as discussed above.

A.3 Step 3: Efficient Evaluation of Bilinear, Quadratic, and Linear Terms

We have established that $\hat{g}(\boldsymbol{\lambda}, \mathbf{S}, \mathbf{D})$ is a sound lower bound for every $(\boldsymbol{\lambda}, \mathbf{S}, \mathbf{D})$ with $\boldsymbol{\lambda} \ge 0$, regardless of the particular choices of $\mathbf{S}$ and $\mathbf{D}$. Equivalently, we can view $(\boldsymbol{\lambda}, \mathbf{S}, \mathbf{D})$ as a parameter vector and Θ as the feasible parameter space

$$\Theta \;=\; \{\ (\boldsymbol{\lambda}, \mathbf{S}, \mathbf{D}) \in \mathbb{R}^m \times \mathbb{R}^n \times \mathbb{R}^{n \times n} \;\mid\; \boldsymbol{\lambda} \ge 0\ \},$$

so that for every parameter in Θ we obtain a finite, certified bound $\hat{g}(\boldsymbol{\lambda}, \mathbf{S}, \mathbf{D}) \le o_{\min}$.

Next, we describe how to *compute* $\hat{g}(\boldsymbol{\lambda}, \mathbf{S}, \mathbf{D})$ efficiently. For fixed $(\boldsymbol{\lambda}, \mathbf{S}, \mathbf{D})$, the relaxed objective decomposes into independent one- and two-dimensional subproblems. By solving each subproblem separately and summing their optimal values, we obtain the value of $\hat{g}(\boldsymbol{\lambda}, \mathbf{S}, \mathbf{D})$ and, at the same time, we update the feasible parameter space Θ to reflect any additional constraints discovered during this process. More concretely, the value of $\hat{g}(\boldsymbol{\lambda}, \mathbf{S}, \mathbf{D})$ is obtained by solving the following one- and two-dimensional subproblems for each class of terms:

(1) **Bilinear terms:** Consider a general bilinear subproblem (assume $A \neq 0$ as only such terms occur when we split the objective):

$$\min_{(v_i, v_k) \in [l_i, u_i] \times [l_k, u_k]} \left(A\, v_i v_k + B\, v_i + C\, v_k \right).$$

If we look at $\hat{g}(\boldsymbol{\lambda}, \mathbf{S}, \mathbf{D})$ in Eq (8), we see that B and C depend on the parameters $\mathbf{D}$ introduced in our splitting (and A is a fixed constant), so the conditions derived below give rise to corresponding constraints that we add to the parameter set Θ:

$$\Theta \;\leftarrow\; \Theta \,\cap\, \{\text{new sign or bound constraint}\}\,.$$

First, we rewrite:

$$A\, v_i v_k + B\, v_i + C\, v_k \;=\; (A\, v_i + C)\left(v_k + \frac{B}{A}\right) - \frac{BC}{A},$$

so that the product is between two affine terms with intervals:

$$A\,v_i + C \in \begin{cases} [A\,l_i + C,\ A\,u_i + C]\,, & A \geq 0\,, \\ [A\,u_i + C,\ A\,l_i + C]\,, & A < 0\,, \end{cases} \qquad v_k + \frac{B}{A} \in [\,l_k + B/A,\ u_k + B/A\,]\,.$$

Setting

$$l_1 := \min(A\,l_i + C,\ A\,u_i + C), \qquad u_1 := \max(A\,l_i + C,\ A\,u_i + C),$$
$$l_2 := l_k + \frac{B}{A}, \qquad u_2 := u_k + \frac{B}{A}\,,$$

the value extreme products are $\min(\{\, l_1 l_2,\ l_1 u_2,\ u_1 l_2,\ u_1 u_2 \,\}) - BC/A$.

There are four configurations in which one of these corner-products may diverge to $-\infty$:

(a) $l_1 = -\infty$ and $u_2 > 0$,
(b) $l_2 = -\infty$ and $u_1 > 0$,
(c) $u_1 = +\infty$ and $l_2 < 0$,
(d) $u_2 = +\infty$ and $l_1 < 0$.

If in any of these cases the corresponding paired bound is also unbounded (e.g. $u_2 = +\infty$, $u_1 = +\infty$, $l_2 = -\infty$, or $l_1 = -\infty$, respectively), then the objective is unbounded and cannot yield a finite lower bound. In this situation we add the contradictory constraint

$$\Theta \ \leftarrow\ \Theta \ \cap\ \{\, 0 \leq -1 \,\}\,,$$

rendering the parameter set infeasible so that the resulting subproblem value is $-\infty$.

If, instead, the paired bound is finite, we add the appropriate sign constraint ($u_2 \leq 0$, $u_1 \leq 0$, $l_2 \geq 0$, or $l_1 \geq 0$) to Θ:

$$\Theta \ \leftarrow\ \Theta \ \cap\ \{\text{the corresponding bound constraint}\}\,,$$

so that the problematic product cannot become negatively unbounded.

Once all such constraints have been added to Θ, the bilinear subproblem is well-posed. Its optimal value is then

$$l_{\text{bilin}} \ :=\ \min\{\, l_1 l_2,\ l_1 u_2,\ u_1 l_2,\ u_1 u_2 \,\} \ -\ \frac{BC}{A}\,,$$

which is finite if the constraints are satisfied. Otherwise, if the constraints cannot be satisfied, this subproblem value is $-\infty$.

(2) **Quadratic terms:** Consider a general quadratic subproblem (assume $A \neq 0$ as only such terms occur after splitting the objective):

$$\min_{v_i \in [l_i, u_i]} \ \left(A\,v_i^2 + B\,v_i\right),$$

where B depends on the splitting parameters $\mathbf{S}$, so the conditions derived below give rise to corresponding constraints that we add to Θ:

$$\Theta \ \leftarrow\ \Theta \ \cap\ \{\text{new constraint}\}\,.$$

- If $A > 0$ (convex), we first compute the unconstrained minimizer

$$v_i^* := -\frac{B}{2A}\,.$$

v_i^* is the global minimizer. If $v_i^* \in [l_i, u_i]$, the objective value is

$$l_{\text{quad}} := A\,(v_i^*)^2 + B\,v_i^*\,,$$

which is finite. If $v_i^* \notin [l_i, u_i]$, then the minimizer lies at one of the endpoints. Moreover, the unconstrained minimizer cannot lie outside the box if both bounds were truly unbounded

(i.e. $l_i = -\infty$ and $u_i = +\infty$), so at least one of the endpoints is finite. Thus, we can always evaluate the objective at the finite endpoint(s) and take the smaller value.

$$l_{\text{quad}} := \min\{\, A\, l_i^2 + B\, l_i,\ A\, u_i^2 + B\, u_i \,\},$$

which is finite.

- If $A < 0$ (concave), then the objective can decrease without bound unless both l_i and u_i are finite. More precisely:
  - If $l_i = -\infty$, then as $v_i \to -\infty$ we have $A\, v_i^2 \to -\infty$.
  - If $u_i = +\infty$, then as $v_i \to +\infty$ we have $A\, v_i^2 \to -\infty$.

  In either of these cases the subproblem is unbounded below, so we add the contradictory constraint

$$\Theta \ \leftarrow\ \Theta \ \cap\ \{\, 0 \le -1 \,\},$$

  rendering the parameter set infeasible and the subproblem value $-\infty$. If both l_i and u_i are finite, we evaluate at the two endpoints and take

$$l_{\text{quad}} := \min\{\, A\, l_i^2 + B\, l_i,\ A\, u_i^2 + B\, u_i \,\},$$

  which is finite.

(3) **Linear terms:** Consider the subproblem

$$\min_{\mathbf{v} \in [\mathbf{l}, \mathbf{u}]} \ \mathbf{c}^\top \mathbf{v} + d$$

We compute the value of this objective variable-wise. That is, for each term $c_i \cdot v_i$ where $v_i \in [l_i, u_i]$, we compute its minimum contribution L_i and determine a corresponding set of constraints Θ_i on the parameters (i.e., on the coefficients c_i) that ensure this minimum is finite. We consider four exhaustive cases:

(a) *Both bounds finite:* The minimum depends on the sign of c_i: if $c_i \ge 0$, the minimum is $c_i \cdot l_i$; if $c_i < 0$, it is $c_i \cdot u_i$. This can be expressed uniformly as:

$$L_i = -\frac{|c_i|}{2}(u_i - l_i) + \frac{c_i}{2}(u_i + l_i)$$

No constraint is needed on c_i in this case.

(b) *Lower bound is $-\infty$, upper bound is finite:* If $c_i > 0$, the minimum is unbounded ($-\infty$). To ensure a finite value, we require $c_i \le 0$, in which case $L_i = c_i \cdot u_i$.

(c) *Lower bound is finite, upper bound is $+\infty$:* If $c_i < 0$, the minimum is unbounded. To ensure a finite value, we require $c_i \ge 0$, in which case $L_i = c_i \cdot l_i$.

(d) *Both bounds infinite:* In this case, the minimum is finite only if $c_i = 0$, so we add this constraint and take $L_i = 0$.

We apply this procedure to all coordinates $i = 1, \ldots, n$, summing the individual contributions to compute the overall minimized value:

$$L = \sum_{i=1}^{n} L_i + d$$

The complete constraint set Θ is formed by collecting the individual Θ_i across all variables. If the parameters $(\boldsymbol{\lambda}, \mathbf{S}, \mathbf{D})$ satisfy Θ, then the dual value is well-defined and finite. Otherwise, the objective is unbounded and evaluates to $-\infty$.

Conclusion. In Step A.2, we constructed the relaxed objective $\hat{g}(\boldsymbol{\lambda}, \mathbf{S}, \mathbf{D})$ by decomposing the inner minimization into independent subproblems and applying the min-split inequality. This yielded a symbolic under-approximation of $o_{\min}$. In Step A.3, we showed how to efficiently evaluate $\hat{g}(\boldsymbol{\lambda}, \mathbf{S}, \mathbf{D})$

by solving each bilinear, quadratic, and linear subproblem separately and aggregating the resulting sign and bound constraints into a parameter set Θ.

Together, these steps define a *Parametric Scalar Map* (Θ, L) for the optimization problem (3), where

$$L : \Theta \to \mathbb{R}, \qquad L(\boldsymbol{\lambda}, \mathbf{S}, \mathbf{D}) := \hat{g}(\boldsymbol{\lambda}, \mathbf{S}, \mathbf{D}) .$$

By construction, each $(\boldsymbol{\lambda}, \mathbf{S}, \mathbf{D}) \in \Theta$ yields a finite, sound lower bound on $o_{\min}$—that is,

$$L(\boldsymbol{\lambda}, \mathbf{S}, \mathbf{D}) \leq o_{\min} \quad \text{and} \quad L(\boldsymbol{\lambda}, \mathbf{S}, \mathbf{D}) \in \mathbb{R} .$$

If Θ is empty or infeasible, then (Θ, L) yields no informative bound beyond the trivial $-\infty$. This may occur if the true optimum is unbounded or if the relaxation is too coarse to derive a finite, sound lower bound. Thus, Steps A.1–A.3 compute a sound PSM for the original optimization problem.

Special case: linear objectives. Our construction also applies when the objective function $f(\mathbf{v})$ is purely linear, i.e., when $H = 0$ and $Q = 0$. In this setting, the primal optimization problem becomes a standard linear program:

$$f(\mathbf{v}) = \mathbf{c}^\top \mathbf{v} + d, \qquad o_{\min} = \min_{\mathbf{v} \in [\mathbf{l}, \mathbf{u}]} \mathbf{c}^\top \mathbf{v} + d \quad \text{s.t. } A'\mathbf{v} \leq \mathbf{b}'.$$

Here, Step 2 (linear coefficient splitting) is no longer needed, as there are no bilinear or quadratic terms to redistribute. The Parametric Scalar Map simplifies to the pair (Θ, L), where:

$$\Theta = \{\ \boldsymbol{\lambda} \in \mathbb{R}^m_{\geq 0} \ \mid \ \text{linear subproblem yields finite bound}\ \} \quad \text{and} \quad L(\boldsymbol{\lambda}) = \hat{g}(\boldsymbol{\lambda}) .$$

However, *feasibility constraints on $\boldsymbol{\lambda}$ still arise* from the symbolic analysis of the linear evaluation step (see Section A.3). Specifically, to ensure that the inner minimum over $[\mathbf{l}, \mathbf{u}]$ is finite, the affine objective coefficients $\mathbf{c} + A'^\top \boldsymbol{\lambda}$ must obey sign conditions based on unbounded directions of the box domain. For instance, if $l_i = -\infty$ for some coordinate i, then to avoid $-\infty$ in the objective, we must ensure that the i-th coefficient of $\mathbf{c} + A'^\top \boldsymbol{\lambda}$ is non-negative; similar conditions apply for $u_i = +\infty$. These constraints prune the parameter space Θ accordingly. Thus, even in the linear case, our construction produces a valid Parametric Scalar Map that yields sound bounds $L(\boldsymbol{\lambda}) \leq o_{\min}$, with Θ encoding the feasibility region induced by symbolic soundness conditions.

B Algorithms

Algorithm 3 Compute Effective Update Map (EUM)

Input: Instruction block $\mathcal{B} = [I_1, I_2, \ldots, I_k]$; set of input variables V
Output: Map σ from output variables to expressions over V
procedure ComputeEUM($\mathcal{B}$, V)
    Initialize $\sigma \leftarrow \{\}$
    for each instruction $x := e$ in B **do**
        $\hat{e} \leftarrow e[v \mapsto \sigma(v) \mid v \in \text{vars}(e) \cap \sigma]$ ▷ with substitution and algebraic simplification
        $\sigma(x) \leftarrow \hat{e}$
    end for
    return σ
end procedure

Concrete Semantics of a Block. Let $\hat{S} = [ins_1; \ldots; ins_n]$ be an admissible sequence of assignments, and let $\mathcal{B}(\hat{S})$ be the corresponding block with Effective Update Map $\sigma_{\mathcal{B}} : \mathcal{V} \to \mathbb{P}(\mathcal{V})$. Let $\mathcal{A} =$

Algorithm 4 Merging Assignment Sequences into QGO Blocks

Input: CFG $\mathcal{F}$ with basic blocks $\{N_1, \ldots, N_k\}$
Output: Updated CFG $\widehat{\mathcal{F}}$ with merged instruction blocks; set of merged blocks $\mathbb{B}$
procedure MergeAssignmentSequences($C, \mathcal{V}$)
    Initialize $\mathbb{B} \leftarrow \{\}$
    for each basic block N in C **do**
        Initialize $\mathcal{B}_N \leftarrow [\,]$, curr $\leftarrow [\,]$
        for each instruction I in N **do**
            if I is affine or quadratic assignment **then**
                tmp $\leftarrow$ curr $\cup [I]$
                $\sigma \leftarrow$ ComputeEUM(tmp, $\mathcal{V}$)
                if IsQuadraticBounded(σ) **then**
                    curr $\leftarrow$ tmp
                else
                    Append curr to $\mathcal{B}_N$; curr $\leftarrow [I]$ ▷ Degree exceeds 2 -> Instruction in new block
                end if
            else
                if curr $\neq [\,]$ **then**
                    Append curr to $\mathcal{B}_N$; curr $\leftarrow [\,]$
                end if ▷ Skip non-assignment instruction I
            end if
        end for
        if curr $\neq [\,]$ **then**
            Append curr to $\mathcal{B}_N$
        end if
        Replace instructions of N with $\mathcal{B}_N$; $\mathbb{B} \leftarrow \mathbb{B} \cup \mathcal{B}_N$
    end for
    return $(\widehat{C}, \mathbb{B})$
end procedure

$\{x_1, \ldots, x_m\}$ be the set of variables updated in the block. Then, for any input state s, the concrete big-step semantics of the block is defined as:

$$[\![\mathcal{B}(\hat{S})]\!](s) \triangleq s\big[x_1 \leftarrow \sigma_{\mathcal{B}}(x_1)(s),\ \ldots,\ x_m \leftarrow \sigma_{\mathcal{B}}(x_m)(s)\big],$$

where $\sigma_{\mathcal{B}}(x_i)(s)$ denotes the evaluation of the polynomial $\sigma_{\mathcal{B}}(x_i)$ in the input state s.

C Theorems and Proofs

Theorem C.1 (Soundness of PSM Computing Procedure). *The procedure outlined in Appendix A computes a Parametric Scalar Map $\mathcal{M} = (\Theta, L)$ for optimization problems of the form:*

$$o_{\min} = \min_{\mathbf{v} \in \mathbb{R}^n} f(\mathbf{v}) \quad s.t. \quad A\,\mathbf{v} \leq \mathbf{b},$$

where the objective function $f(\mathbf{v})$ is a quadratic polynomial given by:

$$f(\mathbf{v}) = \sum_{1 \leq i < k \leq n} H_{ik}\, v_i v_k + \sum_{i=1}^{n} Q_i\, v_i^2 + \mathbf{c}^\top \mathbf{v} + d.$$

The procedure is sound, meaning that for all $\theta \in \Theta$, the bound $L(\theta)$ satisfies $L(\theta) \leq o_{\min}$.

Proof. As described in Appendix A, the procedure constructs the PSM $\mathcal{M} = (\Theta, L)$ using the following parameters:

- Dual multipliers $\boldsymbol{\lambda} \in \mathbb{R}^m$ for the constraints $A\,\mathbf{v} \leq \mathbf{b}$,
- Splitting parameters $\mathbf{S} \in \mathbb{R}^n$ and $\mathbf{D} \in \mathbb{R}^{n \times n}$ used to redistribute linear coefficients during the decomposition of the inner minimization.

The parameter space Θ consists of all tuples $(\boldsymbol{\lambda}, \mathbf{S}, \mathbf{D})$ that satisfy the feasibility constraints collected during the minimization of the decomposed objective components in Section A.3, together with the constraint $\boldsymbol{\lambda} \geq 0$. The map L is defined as:

$$L(\boldsymbol{\lambda}, \mathbf{S}, \mathbf{D}) := \hat{g}(\boldsymbol{\lambda}, \mathbf{S}, \mathbf{D}),$$

where $\hat{g}$ denotes the relaxed objective obtained by decomposing the dual objective $g(\boldsymbol{\lambda})$ into independent subproblems using the splitting parameters $\mathbf{S}$ and $\mathbf{D}$.

By the weak duality theorem, we know that:

$$g(\boldsymbol{\lambda}) \;\leq\; o_{\min} \qquad \text{for all } \boldsymbol{\lambda} \geq 0.$$

Furthermore, by construction of the relaxation (Appendix A.2), the decomposed bound satisfies:

$$\hat{g}(\boldsymbol{\lambda}, \mathbf{S}, \mathbf{D}) \;\leq\; g(\boldsymbol{\lambda}) \qquad \text{for all } \mathbf{S}, \mathbf{D}.$$

Combining the two, we obtain:

$$L(\boldsymbol{\lambda}, \mathbf{S}, \mathbf{D}) = \hat{g}(\boldsymbol{\lambda}, \mathbf{S}, \mathbf{D}) \;\leq\; g(\boldsymbol{\lambda}) \;\leq\; o_{\min},$$

for all $\boldsymbol{\lambda} \geq 0$ and for all choices of $\mathbf{S}, \mathbf{D}$.

Since Θ is defined to include only those tuples $(\boldsymbol{\lambda}, \mathbf{S}, \mathbf{D})$ satisfying $\boldsymbol{\lambda} \geq 0$ along with the additional feasibility constraints, it follows that every $(\boldsymbol{\lambda}, \mathbf{S}, \mathbf{D}) \in \Theta$ satisfies the above inequality. Thus, L yields a finite, sound lower bound on $o_{\min}$ for each valid parameter, establishing that $\mathcal{M} = (\Theta, L)$ is a sound Parametric Scalar Map for the given optimization problem. Note that the correctness of the proof relies on the correct computation of the relaxed bound $\hat{g}(\boldsymbol{\lambda}, \mathbf{S}, \mathbf{D})$, which relies on the correctness of the algebraic decompositions used to evaluate the independent bilinear, quadratic, and linear subproblems (Appendix A.3). These decompositions involve standard algebraic manipulations and closed-form expressions, whose correctness we assume.

□

Theorem C.2 (Tightness of PSM Procedure for Linear Programs). *For the linear program*

$$\text{(P)} \qquad o_{\min} \;=\; \min_{\mathbf{v} \in \mathbb{R}^n} \mathbf{c}^\top \mathbf{v} + d \quad \textit{s.t.} \quad A\mathbf{v} \leq \mathbf{b},$$

let $\mathcal{M} = (\Theta, L)$ be the Parametric Scalar Map produced by the procedure of Appendix A. Then the optimal bound $o_{\min}$ is realised by $\mathcal{M}$; that is,

$$\exists\, \theta \in \Theta :\; L(\theta) = o_{\min}.$$

Proof. Assume the primal LP (P) is feasible and has a finite optimum $o_{\min}$; otherwise the claim is vacuous.

Dual form. After separating variable bounds as in Appendix A, the dual objective is

$$g(\boldsymbol{\lambda}) = \min_{\mathbf{v} \in [\mathbf{l}, \mathbf{u}]} (\mathbf{c} + A'^\top \boldsymbol{\lambda})^\top \mathbf{v} - \boldsymbol{\lambda}^\top \mathbf{b}' + d, \qquad \boldsymbol{\lambda} \geq 0,$$

and the PSM construction keeps precisely those multipliers giving finite values:

$$\Theta = \{\boldsymbol{\lambda} \geq 0 \mid g(\boldsymbol{\lambda}) > -\infty\}, \quad L(\boldsymbol{\lambda}) = g(\boldsymbol{\lambda}).$$

Strong duality. Feasibility and boundedness of (P) imply strong duality:

$$o_{\min} = \max_{\boldsymbol{\lambda} \geq 0} g(\boldsymbol{\lambda}).$$

Hence there exists a dual optimal $\boldsymbol{\lambda}^* \geq 0$ with $g(\boldsymbol{\lambda}^*) = o_{\min}$.

Membership in Θ. Since $g(\boldsymbol{\lambda}^*) = o_{\min} > -\infty$, the multiplier $\boldsymbol{\lambda}^*$ satisfies the finiteness condition defining Θ, so $\boldsymbol{\lambda}^* \in \Theta$.

Conclusion. Setting $\theta = \boldsymbol{\lambda}^*$ gives $L(\theta) = g(\boldsymbol{\lambda}^*) = o_{\min}$, so the optimal bound is realised by $\mathcal{M}$.

Remark. This tightness result hinges on strong duality, which holds for linear programs but need not hold for general quadratic objectives, especially non-convex ones where a positive duality gap may exist. In the linear case, strong duality guarantees that the dual optimum exactly matches the primal optimum $o_{\min}$, so $\mathcal{M}$ always achieves the tightest possible sound bound. For general quadratic objectives, a positive duality gap may cause the dual optimum to fall strictly below $o_{\min}$, and even this tightness guarantee fails: $\mathcal{M}$ may not be able to produce $o_{\min}$ as a bound at all. Note also that $\mathcal{M}$ is currently not complete, in the sense that not every finite $\ell \leq o_{\min}$ need be realizable: the box-peeled construction trades completeness over all lower bounds for tractability of optimization over Θ. Completeness could in principle be recovered by dualising the box constraints as well, at the cost of introducing additional multipliers and making optimization over Θ significantly harder. □

Theorem C.3 (Interval Relaxation Behavior). *The symbolic procedure described in Appendix A constructs a Parametric Scalar Map* $\mathcal{M} = (\Theta, L)$ *for optimization problems of the form:*

$$o_{\min} \;=\; \min_{\mathbf{v}\in\mathbb{R}^n} f(\mathbf{v}) \quad s.t. \quad A\,\mathbf{v} \leq \mathbf{b},$$

where the objective $f(\mathbf{v})$ *is a quadratic polynomial given by:*

$$f(\mathbf{v}) = \sum_{1\leq i<k\leq n} H_{ik}\, v_i v_k + \sum_{i=1}^{n} Q_i\, v_i^2 + \mathbf{c}^\top \mathbf{v} + d.$$

When the parameters are set to zero, i.e., $\theta = (\boldsymbol{\lambda}, \mathbf{S}, \mathbf{D}) = (\vec{0}, \vec{0}, \vec{0})$*, the behavior of the map* $\mathcal{M}$ *coincides with that of interval relaxation. If interval relaxation yields a finite lower bound, then* $\vec{0} \in \Theta$ *and* $L(\vec{0})$ *matches the interval relaxation value. Otherwise, if the interval relaxation is unbounded below, then* $\vec{0} \notin \Theta$.

Proof. As described in Appendix A, the Parametric Scalar Map $\mathcal{M} = (\Theta, L)$ is constructed using:

- Dual multipliers $\boldsymbol{\lambda} \in \mathbb{R}^m$ for the constraints $A\,\mathbf{v} \leq \mathbf{b}$,
- Splitting parameters $\mathbf{S} \in \mathbb{R}^n$ and $\mathbf{D} \in \mathbb{R}^{n\times n}$ for distributing linear terms.

The lower bound is computed by the relaxed expression:

$$\begin{aligned}\hat{g}(\boldsymbol{\lambda}, \mathbf{S}, \mathbf{D}) = &\sum_{1\leq i<k\leq n} \min_{(v_i,v_k)\in[l_i,u_i]\times[l_k,u_k]} \left(H_{ik} v_i v_k + D^i_{ik} v_i + D^k_{ik} v_k\right) \\ &+ \sum_{i=1}^{n} \min_{v_i\in[l_i,u_i]} \left(Q_i v_i^2 + S_i v_i\right) \\ &+ \min_{\mathbf{v}\in[\mathbf{l},\mathbf{u}]} \left(\left(\mathbf{c} + A^\top\boldsymbol{\lambda} - \mathbf{S} - \boldsymbol{\Delta}\right)^\top \mathbf{v}\right) + d - \boldsymbol{\lambda}^\top \mathbf{b}\end{aligned}$$

When $(\boldsymbol{\lambda}, \mathbf{S}, \mathbf{D}) = (\vec{0}, \vec{0}, \vec{0})$, this simplifies to:

$$\begin{aligned}\hat{g}(\vec{0}, \vec{0}, \vec{0}) = & \sum_{1 \le i < k \le n} \min_{(v_i, v_k) \in [l_i, u_i] \times [l_k, u_k]} H_{ik} v_i v_k \\ & + \sum_{i=1}^{n} \min_{v_i \in [l_i, u_i]} Q_i v_i^2 \\ & + \min_{\mathbf{v} \in [\mathbf{l}, \mathbf{u}]} \mathbf{c}^\top \mathbf{v} + d\end{aligned}$$

This is precisely the interval relaxation of $f(\mathbf{v})$ over the input box $[l_i, u_i]$, ignoring the linear constraints $A\,\mathbf{v} \le \mathbf{b}$. During evaluation, Section A.3 adds symbolic feasibility constraints to ensure that any parameter tuple leading to a bound of $-\infty$ is excluded from the parameter space Θ. In particular, if the interval relaxation yields $-\infty$, then the zero-point configuration $(\boldsymbol{\lambda}, \mathbf{S}, \mathbf{D}) = (\vec{0}, \vec{0}, \vec{0})$ is ruled out by these constraints, and $\vec{0} \notin \Theta$. Otherwise, if the interval relaxation yields a finite value, then $\vec{0} \in \Theta$, and the corresponding bound $L(\vec{0})$ exactly matches the interval relaxation value. Thus, the Parametric Scalar Map recovers interval relaxation as the zero-parameter instantiation, when valid.

□

Theorem C.4 (Polyhedrality of the feasible-parameter set). *Let $\theta = (\boldsymbol{\lambda}, \mathbf{S}, \mathbf{D}) \in \mathbb{R}^{m+n+n^2}$ be the parameter vector constructed by the PSM computing procedure in Appendix A. The set of admissible parameters*

$$\Theta \;=\; \big\{\theta \mid \textit{all feasibility checks introduced in Steps 2–4 are satisfied}\big\}$$

is a (possibly empty) polyhedron*; equivalently, there exist a matrix M and vector $\mathbf{h}$ such that $\Theta = \{\theta \mid M\theta \le \mathbf{h}\}$.*

Proof. We enumerate the constraints added during the construction and show that each is linear in θ.

(i) Dual non-negativity. Step 2 fixes $\boldsymbol{\lambda} \ge \mathbf{0}$, that is, $\lambda_i \ge 0$ for $i = 1, \ldots, m$; these are linear half-spaces.

(ii) Bilinear sub-problems. For every pair (i, k) the rule in Section A.3 may add

$$u_2 \le 0,\ u_1 \le 0,\ l_2 \ge 0,\ \text{or } l_1 \ge 0 \quad \text{or the contradictory } 0 \le -1.$$

Here l_1, u_1, l_2, u_2 are affine expressions in the splitting parameters D^i_{ik}, D^k_{ik}; hence each inequality is linear in θ.

(iii) Quadratic sub-problems. When $A < 0$ and a bound is missing, the procedure inserts the contradictory half–space $0 \le -1$; otherwise no new restriction is added. Again, $0 \le -1$ is linear.

(iv) Linear sub-problem. For every coordinate i the guard may require $c_i \le 0,\ c_i \ge 0$, or $c_i = 0$. Because each coefficient takes the affine form $c_i = \hat{c}_i(\boldsymbol{\lambda}, \mathbf{S}, \mathbf{D})$, these relations are linear in θ.

(v) Closure under intersection. The parameter set Θ is the finite intersection of the half-spaces listed in (i)–(iv). A finite intersection of linear half-spaces is a polyhedron by definition.

Hence Θ is polyhedral. □

Theorem C.5 (Differentiability of L function in PSM). *Let $\theta = (\boldsymbol{\lambda}, \mathbf{S}, \mathbf{D}) \in \mathbb{R}^{m+n+n^2}$ be the parameter vector constructed by the PSM computing procedure in Appendix A. This procedure outputs a parametric sound map (PSM) $\mathcal{M} = (\Theta, L)$, where $\Theta \subseteq \mathbb{R}^{m+n+n^2}$ is the polyhedral set of admissible parameters (as established in Theorem C.4) and $L : \Theta \to \mathbb{R}$ is the function that maps each*

admissible $\theta \in \Theta$ to the minimum value of the relaxed dual objective $\hat{g}(\theta)$. Then the function $L(\theta)$ is piecewise-defined and differentiable on the interior of Θ.

More precisely:

- *On any region of Θ where the minimizing terms and endpoints in all subproblems (bilinear, quadratic, and linear) remain fixed, the expression for $L(\theta)$ is a differentiable function of θ.*
- *The function $L(\theta)$ is continuous on all of Θ, and differentiable almost everywhere in Θ, except possibly at boundaries where the active minimizer (e.g., which endpoint of an interval achieves the minimum) changes.*

Proof. The function $L(\theta)$ is computed as described in the derivation procedure in Appendix A, by summing the minimum values of several subproblems, one for each bilinear, quadratic, and linear term in the relaxed objective. We analyze the differentiability of each class of subproblem:

(i) Linear terms. The minimized value of each $c_i v_i$ term is computed based on whether c_i is positive, negative, or zero. The result is a piecewise-linear function of c_i, and since $c_i = \hat{c}_i(\boldsymbol{\lambda}, \mathbf{S}, \mathbf{D})$ is affine in θ, the contribution L_i is piecewise-affine in θ, hence differentiable except at switching points (where the sign of c_i changes).

(ii) Quadratic terms. Consider subproblems of the form $\min_{v_i \in [l_i, u_i]} Av_i^2 + Bv_i$ with fixed $A \neq 0$ and parameter-dependent $B = \hat{B}(\theta)$. The analysis splits into two cases:

- If $A > 0$ (convex), the objective is minimized at the unconstrained critical point $v_i^* = -B/(2A)$. If $v_i^* \in [l_i, u_i]$, the minimum is attained there and equals $A(v_i^*)^2 + Bv_i^*$, which is a smooth function of B, hence of θ. If $v_i^* \notin [l_i, u_i]$, the minimum occurs at one of the endpoints. Since B is affine in θ, both $Al_i^2 + Bl_i$ and $Au_i^2 + Bu_i$ are smooth in θ, and their minimum is piecewise-smooth depending on which endpoint is active. Thus, in both cases, the quadratic contribution is differentiable on regions where the active minimizer remains fixed.
- If $A < 0$ (concave), the objective can decrease without bound unless both l_i and u_i are finite. In particular, the term Av_i^2 dominates and diverges to $-\infty$ if $v_i \to \pm\infty$. To ensure boundedness, the feasibility check adds the constraint that both bounds must be finite. If so, the minimum is computed as $\min\{Al_i^2 + Bl_i,\ Au_i^2 + Bu_i\}$, which is continuous and piecewise-differentiable in θ.

(iii) Bilinear terms. Consider the subproblem

$$\min_{(v_i, v_k) \in [l_i, u_i] \times [l_k, u_k]} \left(A\, v_i v_k + B\, v_i + C\, v_k \right),$$

with $A \neq 0$ fixed and B, C affine in θ. Rewriting:

$$Av_i v_k + Bv_i + Cv_k = (Av_i + C)(v_k + B/A) - \frac{BC}{A},$$

we reduce to minimizing a product of two affine expressions over intervals $[l_1, u_1]$ and $[l_2, u_2]$, where:

$$l_1 := \min(Al_i + C,\ Au_i + C), \quad l_2 := l_k + B/A, \quad \text{etc.}$$

The minimal value is:

$$l_{\text{bilin}} := \min\{l_1 l_2,\ l_1 u_2,\ u_1 l_2,\ u_1 u_2\} - \frac{BC}{A}.$$

Unboundedness arises in four cases (e.g., $l_1 = -\infty$ and $u_2 > 0$). If the paired bound is also infinite, we add the contradictory constraint $0 \leq -1$ to Θ; otherwise, we insert the appropriate bound constraint (e.g., $u_2 \leq 0$). This ensures the subproblem remains well-posed and the resulting l_{bilin} is finite and piecewise-smooth in θ.

(iv) Summation. The total value $L(\theta)$ is the sum of these subproblem contributions. Since each term is continuous and piecewise-differentiable, their sum is also continuous and piecewise-differentiable. Differentiability holds on any region where the active minimizers in all subproblems are fixed. □

Theorem C.6 (EUM over-approximates a sequence). *Let $\tilde{S}$ be a sequence of assignments, and let $\mathcal{B}(\tilde{S})$ have concrete semantics given by its EUM. Then*

$$[\![\tilde{S}]\!] \subseteq [\![\mathcal{B}(\tilde{S})]\!].$$

Proof. Let (S, T) be program states (valuations) with $(S, T) \in [\![\tilde{S}]\!]$, and write $\tilde{S}$ as $x_1 := e_1;\ x_2 := e_2;\ \ldots;\ x_k := e_k$. Unfolding the run gives intermediate states $U_0 = S, U_1, \ldots, U_k = T$, where step i sets x_i to the value of e_i in U_{i-1} and leaves other variables unchanged. Thus $T(x_1)$ depends on S, $T(x_2)$ depends on S and the new value of x_1, and in general $T(x_i)$ depends on S and the updated values $x_1, \ldots, x_{i-1}$. By a straightforward induction on i, there exist flattened expressions F_v such that

$$T(v) = \text{value of } F_v \text{ in } S \quad \text{for all } v,$$

i.e., all intermediate states $U_1, \ldots, U_{k-1}$ can be eliminated by substitution so that each final variable is expressed purely in terms of the initial state. The EUM for the block captures exactly this flattened relation: it is obtained from these equations by symbolic rewriting and algebraic simplification, in a way that it exactly computes the function F, and so $(S, T) \in [\![\mathcal{B}(\tilde{S})]\!]$. Since (S, T) was arbitrary, we conclude $[\![\tilde{S}]\!] \subseteq [\![\mathcal{B}(\tilde{S})]\!]$.

Why inclusion can be strict. Symbolic simplification in the EUM can hide intermediate errors and thereby admit extra states. *Example.* Consider

$$a := a_1; \quad b := a_1; \quad c := a/d - b/d.$$

Flattening gives $a' = a_1,\ b' = a_1,\ c' = (a_1/d) - (a_1/d) = 0$. If the EUM keeps only these simplified postconditions and omits the condition on the intermediate divisions, then states with $d = 0$ and $c' = 0$ satisfy the EUM even though the stepwise execution would fail on the divisions. Thus new states are added and $[\![\tilde{S}]\!] \subsetneq [\![\mathcal{B}(\tilde{S})]\!]$. □

Remark. The sequences $\hat{S}$ considered by AbsEvolve are those whose combined effect lies within the QGO class. Such sequences consist solely of affine and quadratic assignments, which are total operations defined for all input states. For such sequences, no intermediate errors can arise during symbolic flattening, and the EUM captures the exact semantics without introducing additional states, i.e., $[\![\hat{S}]\!] = [\![\mathcal{B}(\hat{S})]\!]$.

D Evaluation Data

D.1 Linear Case

Table 1. Full stats for Zones Experiments (Affine)

Analysis Method	Runtime (s)	Invariants Strengthened over ELINA (OG)	Invariants Strengthened from Prev. Step (MG)	Remaining Gap (RG) from most-precise
ELINA	18.53	0	-	121
AbsEvolve-R-0	60.47	91	91	48
AbsEvolve-R-1	61.82	97	10	42
AbsEvolve-R-2	65.2	116	21	30
AbsEvolve-R-3	68.8	121	30	4
AbsEvolve-R-4	69.59	121	0	4
AbsEvolve-R-5	71.9	121	4	0
LP Solver Based	230.74	121	-	0

Table 2. Full stats for Octagon Experiments (Affine)

Analysis Method	Runtime (s)	Invariants Strengthened over ELINA (OG)	Invariants Strengthened from Prev. Step (MG)	Remaining Gap (RG) from most-precise
ELINA	20.11	0	-	59
AbsEvolve-R-0	82.07	24	24	55
AbsEvolve-R-1	83.9	42	18	41
AbsEvolve-R-2	89.07	57	31	29
AbsEvolve-R-3	102.66	59	29	4
AbsEvolve-R-4	108.68	59	0	4
AbsEvolve-R-5	111.06	59	4	0
LP Solver Based	344.01	59	-	0

D.2 Nonlinear Operators and Expressive Domains

The following programs give numerical overflow errors while running polyhedra analysis with ELINA: geo1-u.c, geo1-u2.c, fermat1.c, fermat2.c, fermat1-ll.c, fermat2-ll.c, dijkstra.c and hard.c

Table 3. Full stats for Zones Experiments (Affine + Quadratic)

Analysis Method	Runtime (s)	Invariants Strengthened over ELINA (OG)	Invariants Strengthened from Prev. Step (MG)
ELINA	15.48	0	-
AbsEvolve-R-0	68.09	110	110
AbsEvolve-R-1	75.34	112	6
AbsEvolve-R-2	79.77	131	19
AbsEvolve-R-3	85.86	135	25
AbsEvolve-R-4	95.78	135	0
AbsEvolve-R-5	97.94	135	4

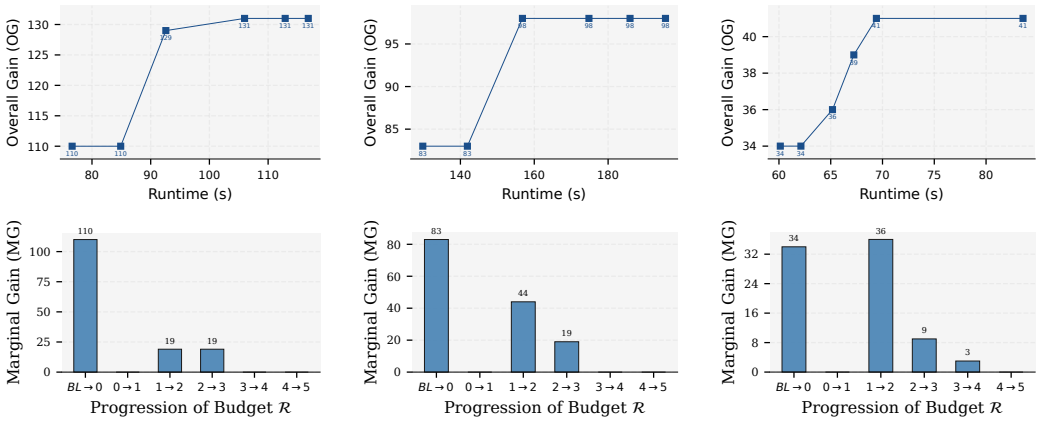
Table 4. Full stats for Octagon Experiments (Affine + Quadratic)

Analysis Method	Runtime (s)	Invariants Strengthened over ELINA (OG)	Invariants Strengthened from Prev. Step (MG)
ELINA	16.44	0	-
AbsEvolve-R-0	118.39	95	95
AbsEvolve-R-1	125.75	97	14
AbsEvolve-R-2	137.52	111	40
AbsEvolve-R-3	154.76	113	25
AbsEvolve-R-4	158.3	113	0
AbsEvolve-R-5	168.08	113	14

Table 5. Full stats for Polyhedra Experiments (Quadratic)

Analysis Method	Runtime (s)	Invariants Strengthened over ELINA (OG)	Invariants Strengthened from Prev. Step (MG)
ELINA	23.2	0	-
AbsEvolve-R-0	99.95	38	38
AbsEvolve-R-1	107.1	38	0
AbsEvolve-R-2	112.88	40	40
AbsEvolve-R-3	126.95	43	9
AbsEvolve-R-4	130.42	45	3
AbsEvolve-R-5	134.16	45	0

D.3 Results without analyzing sequences together



(a) Zones (b) Octagons (c) Polyhedra

Fig. 8. Evolution of invariant strengthening while handling quadratic assignments across domains.

Table 6. Stats for Zones Experiments without sequence merging (Affine + Quadratic)

Analysis Method	Runtime (s)	Invariants Strengthened over ELINA (OG)	Invariants Strengthened from Prev. Step (MG)
ELINA	13.53	0	-
AbsEvolve-R-0	76.62	110	110
AbsEvolve-R-1	84.89	110	0
AbsEvolve-R-2	92.58	129	19
AbsEvolve-R-3	105.99	131	19
AbsEvolve-R-4	112.91	131	0
AbsEvolve-R-5	116.84	131	0

Table 7. Stats for Octagon Experiments without sequence merging (Affine + Quadratic)

Analysis Method	Runtime (s)	Invariants Strengthened over ELINA (OG)	Invariants Strengthened from Prev. Step (MG)
ELINA	15.08	0	-
AbsEvolve-R-0	129.81	83	83
AbsEvolve-R-1	141.89	83	0
AbsEvolve-R-2	156.8	98	44
AbsEvolve-R-3	174.81	98	19
AbsEvolve-R-4	185.94	98	0
AbsEvolve-R-5	195.54	98	0

Table 8. Stats for Polyhedra Experiments without sequence merging (Quadratic)

Analysis Method	Runtime (s)	Invariants Strengthened over ELINA (OG)	Invariants Strengthened from Prev. Step (MG)
ELINA	26.93	0	-
AbsEvolve-R-0	60.13	34	34
AbsEvolve-R-1	62.12	34	0
AbsEvolve-R-2	65.17	36	36
AbsEvolve-R-3	67.23	39	9
AbsEvolve-R-4	69.41	41	3
AbsEvolve-R-5	83.57	41	0

E Extension to Joins and Disjunctions

The UPOSE algorithm extends naturally to joins and disjunctions. Consider a disjunctive feasible set $S_1 \cup S_2$, arising for instance from a disjunctive guard such as $x \leq y \vee z \leq a$. For template direction i, the bound to compute is:

$$m_i := \min_{x \in S_1 \cup S_2} f_i(x) = \min(m_{i,1}, m_{i,2}), \quad \text{where} \quad m_{i,j} := \min_{x \in S_j} f_i(x).$$

For each branch $j \in \{1, 2\}$, UPOSE constructs a parametric lower bound $L_{i,j}(\lambda_j)$ over a polyhedral parameter space $P_{i,j}$, such that $\forall \lambda_j \in P_{i,j},\ L_{i,j}(\lambda_j) \leq m_{i,j}$. The two branches are then combined as:

$$L_i(\lambda_1, \lambda_2) := \min(L_{i,1}(\lambda_1), L_{i,2}(\lambda_2)),$$

over the joint polyhedral parameter space $P_i = \{(\lambda_1, \lambda_2) \mid \lambda_1 \in P_{i,1}, \lambda_2 \in P_{i,2}\}$. This yields a polyhedral parameter space with a piecewise differentiable objective, on which AGG can be applied directly. More broadly, this demonstrates that our approach is general and can naturally accommodate joins and disjunctions, and once supported, they immediately work across all domains handled by the framework, including new user-defined TCM domains.

Received 2025-11-14; accepted 2026-04-03