Compositional Program Synthesis via Two Abstractions A and A^+: A Short Empirical Note

Abstract

We present a concrete instantiation of a research plan on compositional reasoning via abstraction–refinement. We define two abstraction layers over program synthesis on paired integers: a cross-free factorization A and an interface-augmented A^+ that captures where cross-operations occur. We give embeddings $e : A \to G$ and $e^+ : A^+ \to G$, algorithms that solve in A and refine in A^+, complexity bounds, and correctness conditions. Experiments show large efficiency gains over global search: 7–60× fewer nodes and 8–180× faster in coupled tasks, with identical accuracy. We conclude that A is a direct instantiation of the original plan; A^+ is a problem-specific refinement of the program search space that fits the spirit of abstraction–refinement even if it is not the exact state-space abstraction emphasized in the original note.

---

1 Introduction (Intuition First)

Global program synthesis often explores a huge search space. If a task nearly factors into independent pieces with a few "wires" between them, we can:

1. Solve the easy factors, ignoring the wires.
2. Refine by putting back a small interface that reconnects the factors.

We demonstrate this idea in a tiny DSL on integer pairs (x, y). In separable tasks, solving each coordinate independently is optimal. In coupled tasks (e.g., y must read the current x), a global solver works but is wasteful. Our Compositional+ solver first synthesizes the x-only program, then searches a small space of cross-op placements that wire y to x at just the right moments.

---

2 Concrete Setting

2.1 Domains, DSL, Semantics

- Concrete state space: $G = \mathbb{Z} \times \mathbb{Z}$.

- Primitives are partitioned

$$\Sigma = \Sigma_X \;\cup\; \Sigma_Y \;\cup\; \Sigma_\times, \tag{1}$$

  where Σ_X edits only x, Σ_Y edits only y, and $\Sigma_\times$ are cross-ops (e.g., `add_first_to_second`: $(x, y) \mapsto (x,\, y + x)$).

  A program is a word $p \in \Sigma^*$ with standard functional semantics $[\![p]\!] : G \to G$.

- Supervision: dataset $D = \{(s_i, t_i)\} \subseteq G \times G$. Goal: find p with $\forall i,\ [\![p]\!](s_i) = t_i$.

---

3 Abstraction A: Cross-Free Factorization

3.1 Definition

Let

$$A \;=\; \Sigma_X \times \Sigma_Y. \tag{2}$$

An element $a = (p_X, p_Y)$ means "do p_X on x and p_Y on y, with no cross-ops."

3.2 Embedding and Agreement

Because Σ_X commutes with Σ_Y,

$$e : A \to \Sigma^*, \quad e(p_X, p_Y) = \text{any interleaving of } p_X, p_Y \tag{3}$$

is well-defined up to semantic equivalence: all such interleavings produce the same $[\![e(p_X, p_Y)]\!]$ on G.

3.3 Solve in A

Project the dataset onto coordinates and synthesize independently:

$$\forall i:\ \pi_X([\![p_X]\!](s_i)) = \pi_X(t_i), \tag{4}$$

$$\forall i:\ \pi_Y([\![p_Y]\!](s_i)) = \pi_Y(t_i). \tag{5}$$

If both succeed, return $e(p_X, p_Y)$.

Intuition. A "turns off" the wires between coordinates, producing two small searches.

---

4 Abstraction A^+: Factorization with a Finite Interface

4.1 Slots and Interfaces

Fix a bound K (number of cross-ops). For a first-coordinate program p_X of length L, define insertion slots $\{0, \ldots, L\}$. Let

$$\Pi_{L,K} \;=\; \big\{\, \alpha = (\alpha_1 \le \cdots \le \alpha_K) \;\big|\; \alpha_j \in \{0, \ldots, L\} \,\big\}. \tag{6}$$

Then

$$A^+ \;=\; \bigcup_{L \ge 0} \big(\Sigma_X^L \times \Pi_{L,K}\big). \tag{7}$$

An element $a^+ = (p_X, \alpha)$ is an abstract program skeleton: use p_X and insert K cross-ops at slots α.

4.2 Embedding to Concrete Programs

Choose $\kappa \in \Sigma_\times$ (e.g., `add_first_to_second`). Define

$$e^+(p_X, \alpha) \in \Sigma^* \tag{8}$$

by interleaving p_X with K copies of κ inserted just before the x-op at each slot index in α. (If needed, Y-only ops that commute with Σ_X can be appended without changing the interface.)

4.3 Solve-then-Refine (Compositional+)

1. Solve in A: find p_X satisfying the x-projection of D.
2. Refine in A^+: enumerate $\alpha \in \Pi_{|p_X|,K}$ and test $e^+(p_X, \alpha)$ on full D. Return the first that fits.

4.4 Completeness (Triangular Coupling)

Assume each $\kappa \in \Sigma_\times$ is triangular: it updates y by a function of the current x and leaves x unchanged, i.e.,

$$\kappa : \;(x, y) \mapsto (x,\; y \oplus h(x)). \tag{9}$$

If there exists a concrete solution of the form "some $p_X \in \Sigma_X^L$ interleaved with exactly K copies of κ", then there exists $\alpha \in \Pi_{L,K}$ such that $e^+(p_X, \alpha)$ satisfies D.

Sketch. Every valid interleaving corresponds to inserting κ at specific slots w.r.t. p_X; these slots are exactly α. Hence enumerating $\Pi_{L,K}$ is complete for this family.

---

5 Algorithms and Complexity

- **Global BFS (baseline).** Branching b over Σ; minimal solution length $L_X + K$. Cost: $O(b^{L_X+K})$ (modulo semantic memoization).
- **Compositional+.**
  (i) Synthesize p_X with branching b_X over Σ_X.
  (ii) Enumerate $\binom{L_X+K}{K}$ interfaces (combinations with repetition).

  Cost:

  $$O(b_X^{L_X}) \;+\; O\!\left(\binom{L_X+K}{K}\right). \tag{10}$$

  For small K and moderate L_X, the second term is tiny relative to the global exponential.

---

6 Examples and Results (All Executed)

6.1 Separable Task (fits A)

Target: $(x, y) \mapsto \big(2(x+3),\; y^2+1\big)$.

Minimal program: `inc1_first`×3 → `double_first` and `square_second` → `inc1_second`.

- Global BFS: found length 6, 343 nodes, 0.0119 s.
- Compositional (A): found length 6, 23 nodes, 0.00029 s.
- Both perfectly match held-out tests.

Intuition. Perfect factorization; solving coordinates independently is optimal.

6.2 Coupled Task (needs A^+)

Target: $(x, y) \mapsto \big(2(x+3),\; y+(x+3)\big)$ using cross-op $\kappa =$ `add_first_to_second`.

- Global BFS: found length 5, 187 nodes, 0.0367 s.
- Naïve split: fails (cannot express y's dependence on x).
- Compositional+ ($A \to A^+$): found (equivalent) program, 25 nodes, 0.00032 s.

Intuition. Solve x first, then place a single wire where y must read x.

6.3 Scaling Study (vary $L_X \in \{4,6,8\}$, $K \in \{0,1,2,3\}$, $b_X \in \{2,3,4\}$)

Geometric-mean speedups (Global / Compositional+) across the grid:

- $K = 0$: 4.2× fewer nodes, 8.1× faster.
- $K = 1$: 19.3×, 44.3×.
- $K = 2$: 29.2×, 87.2×.
- $K = 3$: 58.7×, 183×.

Hardest slice ($L_X = 8$, $b_X = 4$): Global nodes 1609 → 32061 as $K : 0 \to 3$; Compositional+ stays near 372–387.

---

7 Conclusion

Distinguishing two abstraction layers crystallizes the method:

- A: cross-free factorization—cheap, complete for separable tasks.
- A^+: finite interface refinement—tiny extra search that restores necessary couplings.

On coupled tasks, $A \to A^+$ achieves the same solutions as global search at a fraction of the cost, validating the core thesis: solve in a simpler world, then refine only what must be coupled.