RL Post-Training Builds Compositional Reasoning Strategies

Azwar Abdulsalam [1 *] **Nishil Patel** [1 *] **Andrew Saxe** [1]

Abstract

Does RL post-training merely amplify primitive skills already latent in a base model, or can it compose primitive skills into new higher-level strategies? We study this question in a fully observable rewrite-grammar environment where the pretraining distribution is known and every generated rewrite can be audited. A Transformer is pretrained on primitive symbol-rewrite chains and post-trained on a Trace-based reasoning task with only a binary final-answer reward. RL solves held-out problems that remain rarely solved by the pretrained model even under much larger sampling budgets, while rejection fine-tuning improves early but plateaus. Trace analysis shows that RL reorganizes primitive competence through a phased compositional mechanism: it first strengthens primitive reductions, then discovers valid composed procedures. These include sequential compositions, which collapse ordered chains of primitive contractions, and parallel compositions, which combine independent primitive contractions in a single step. The composed procedures are not isolated samples; they are reused and consolidated into a stable repertoire. Comparing RL with rejection fine-tuning shows that the key difference is not exploration volume but selectivity: RFT produces many shortcut-like rewrites, much of them invalid, whereas RL concentrates exploration into valid reusable structure. Pretraining ablations show that the emergence of compositional strategies is gated not by primitive exposure alone, but by whether pretraining organizes primitive competence into reduction procedures that RL can later compress. The base model provides weak procedural ingredients; RL builds them into reliable higher-level strategies.

[1]Gatsby Computational Neuroscience Unit, UCL, London, United Kingdom. Correspondence to: Azwar Abdulsalam <azwar.azwar.25@ucl.ac.uk>.

Accepted to the 2nd Workshop on Compositional Learning at ICML 2026, Seoul, South Korea. Copyright 2026 by the author(s).

1. Introduction

Does reinforcement learning (RL) post-training merely reweight behaviors already latent in a base model, or can it compose primitive skills into new higher-level strategies? Recent work has sharpened this into an active debate. Yue et al. (Yue et al., 2025) argue that RL with verifiable rewards often improves small-k success without expanding the large-k capability frontier; ProRL (Liu et al., 2025a) and Yuan et al. (Yuan et al., 2025) present evidence that prolonged RL or controlled compositional tasks can uncover behaviors inaccessible to the base model under extensive sampling; and imitation-style baselines such as rejection fine-tuning are sometimes surprisingly competitive (Chu et al., 2025; Xiong et al., 2025).

A central obstacle is that the relevant mechanisms are hard to observe in pretrained language models. When a post-trained model exhibits a new behavior, it is usually unclear whether that behavior was absent from the base model, merely low-probability, or already present in pretraining data. Aggregate metrics such as pass@k leave three questions open: what strategies are being used, whether their emergence reflects broader exploration or selective filtering, and what pretraining structure makes them reachable in the first place.

We study these questions in a fully observable rewrite-grammar environment. The task abstracts a common structure in step-by-step reasoning: complex solutions can be built by composing local transformations into reusable procedures. A Transformer is pretrained from scratch on primitive rewrite chains and then post-trained on goal-directed contraction task. Because every generated rewrite can be audited against the grammar, we can decompose behavior into primitive rule use, valid composed strategies, and spurious invalid rewrites. The valid composed strategies come in two forms: *sequential compositions*, which collapse ordered chains of primitive contractions, and *parallel compositions*, which combine independent primitive contractions in a single emitted step. This turns normally opaque post-training behavior into exactly classifiable trajectories, letting us ask whether RL reorganizes primitive competence into reusable higher-level strategies.

Contributions.

1. **A fully observable rewrite-grammar testbed.** We in-

troduce an environment in which primitive rules, pretraining histories, and generated rewrites are exactly inspectable, enabling a four-way taxonomy: primitive, macro, parallel, and spurious.

2. **Phased procedural chunking.** We show that RL first strengthens primitive reductions, then discovers, reuses, and consolidates valid compositions of the primitive contractions.
3. **Finite-budget frontier expansion.** We show that RL solves held-out problems the pretrained model rarely solves even under much larger sampling budgets, with gains that emerge late and grow with difficulty.
4. **Selection, not exploration volume.** We show that RFT also attempts many shortcut-like rewrites, but much of this behavior is spurious; RL concentrates exploration into valid reusable structure. A finite-group view of GRPO explains how same-prompt contrast can suppress spurious passenger actions that RFT clones.
5. **Pretraining as procedural substrate.** We show that strategy emergence depends on how pretraining organizes primitive competence into reduction procedures that RL can later compress and reuse.

2. Related Work

RL beyond the base model. A central question in RL with verifiable rewards is whether post-training expands a model's effective capability frontier or mainly reweights behaviors already latent in the base policy. Yue et al. (Yue et al., 2025) argue that RLVR often improves small-k success without expanding the large-k frontier, while support-based analyses emphasize that on-policy RL cannot amplify zero-probability completions (Wu et al., 2025). In contrast, ProRL (Liu et al., 2025a) argues that prolonged RL can make some problems reachable that were inaccessible under extensive base-model sampling. Closest to our setting, Yuan et al. (Yuan et al., 2025) study RL-induced composition in a controlled synthetic task. Our work studies this question in a setting where the mechanism is directly observable: the pretraining distribution is generated by a known grammar, and every emitted solution step can be checked against that grammar. We return to the comparison with prior composition experiments in Section 8.

RL versus imitation-style post-training. Rejection and supervised baselines are often strong in reasoning post-training. STaR bootstraps reasoning from self-generated rationales filtered by answer correctness (Zelikman et al., 2022), and recent work shows that rejection-style or minimalist supervised methods can be competitive with policy-gradient RL in some RLVR settings (Xiong et al., 2025; Chu et al., 2025). We therefore treat rejection fine-tuning as a strong on-policy comparator rather than a weak baseline: RL and RFT are trained on the same prompts and optimized separately under the same final-answer supervision.

Process validity, exploration, and pretraining substrate. Outcome-only rewards can hide invalid intermediate reasoning (Uesato et al., 2022; Lightman et al., 2023; Cobbe et al., 2021). This is hard to measure in natural-language tasks, where valid intermediate steps are usually not enumerable. In our environment, process validity is mechanically checkable, letting us refine recent discussions of exploration and negative samples in RLVR (Wang et al., 2025; Tang et al., 2025). We also connect to work arguing that post-training depends on capabilities and biases shaped during pretraining (Zhao et al., 2025; Liu et al., 2025b); here, because pretraining is controlled, we can ask how the base distribution determines which higher-level strategies RL can later select, compress, and consolidate.

3. Rewrite Grammar and Primitive Actions

The grammar is defined over an alphabet $\mathcal{A}$, where each source symbol maps to one or more globally unique multi-character right-hand-side strings (Figure 1a). Global uniqueness makes primitive contraction unambiguous: any substring matching a right-hand side maps back to exactly one source symbol. The environment supports two inverse primitive operations: *expansion*, which replaces a symbol by one of its right-hand sides, and *contraction*, which replaces a matching substring by its source symbol. Each step is serialized as

$$\texttt{lhs_word} \,|\, \texttt{action} \,|\, \texttt{rhs_word},$$

where the action specifies a position and rule for expansion, or a span and rule for contraction (Figure 1c).

Because the primitive rule set is known, every generated rewrite can be audited against the grammar: it can be a primitive rule application, a valid non-primitive shortcut, or an invalid rewrite. This is what makes trajectory-level strategy analysis possible.

3.1. Pretraining on Primitive Rewrite Chains

The model is pretrained from scratch on chained rewrite trajectories. Starting from an initial word, the data generator repeatedly samples a valid primitive expansion or contraction, appends the corresponding serialized triple, and uses the resulting word as the input to the next step. Thus, pretraining exposes the model to local grammar-consistent rewrite dynamics, but not to the later non-primitive shortcuts analyzed in this paper.

We control the pretraining distribution with a contraction weight ρ, which locally upweights contraction moves rela-

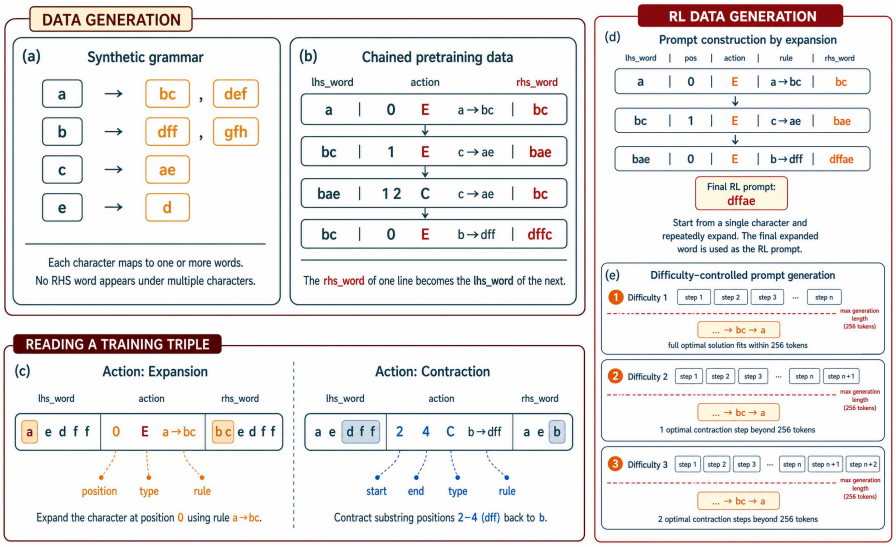

Figure 1. Controlled environment and training pipeline. **(a)** Each source character maps to one or more globally unique right-hand-side strings. **(b)** Pretraining data consists of multi-step sequences in which the output of one step becomes the input to the next. **(c)** Each training triple encodes a single primitive expansion or contraction. **(d)** RL prompts are constructed by repeated forward expansion from a target symbol. **(e)** Difficulty levels are defined by how many optimal contraction steps lie beyond the model's 256-token generation budget.

tive to expansion moves whenever both are available. Thus ρ affects the realized contraction frequency and the amount of contraction chaining, but does not directly specify either statistic. Unless otherwise stated, we use $\rho = 4$. Section 7 later ablates this parameter to test whether properties of the pretraining distribution affect which strategies RL can unlock. Full grammar-generation and optimization details are given in Appendix A.

3.2. Trace-Based Reasoning Task

After pretraining, the model is post-trained on a trace-based reasoning task. A prompt is generated by starting from a target symbol $c^\star$ and repeatedly applying primitive expansions to obtain a longer word w_{prompt}. At post-training time, the model sees only w_{prompt} and must generate a step-by-step contraction trace whose final emitted symbol is $c^\star$.

Difficulty is defined by the length of the optimal primitive contraction solution under the serialized action format. Difficulty 1 problems have an optimal primitive solution that fits within the 256-token generation budget. Difficulty 2 problems require one additional primitive contraction step beyond the budget, and Difficulty 3 problems require two additional steps. The RL training mixture contains 60% Difficulty 1, 20% Difficulty 2, and 20% Difficulty 3 examples. For evaluation, we use held-out problems spanning Difficulties 1–6, including buckets beyond the training distribution.

This difficulty definition is important: higher difficulty does not mean the grammar is different, but that the straightforward primitive contraction trace is too long for the generation budget. Solving harder buckets therefore requires shorter valid reasoning procedures, analogous to using simplification shortcuts rather than executing every local transformation one step at a time.

3.3. RL and RFT Post-Training

The post-training reward is binary and outcome-only: it is 1 if the trajectory consists of well-formed action triples in the format above and the final emitted symbol equals the target $c^\star$, and 0 otherwise. There is no supervision on intermediate rewrites — only the format of the output and the identity of the final symbol matter. Any valid non-primitive strategies that emerge must therefore arise from post-training dynamics rather than from direct reward shaping. This mirrors RLVR on math reasoning, where final-answer correctness is the only training signal (Shao et al., 2024).

Our RL method is Group Relative Policy Optimization (GRPO) (Shao et al., 2024). As an imitation baseline, we use rejection fine-tuning (RFT), which retains successful on-policy rollouts and performs next-token prediction on them. RFT is therefore a strong comparator: any RL-vs-RFT gap cannot be attributed to data or prompt mismatch. Full GRPO and RFT details are in Appendix B.

4. Utilisation of Composition to go Beyond the Base Model

Because the grammar is fully known, every generated rewrite in a successful trajectory can be audited by comparing its left- and right-hand sides. A **primitive contraction**

replaces the right-hand side of an original grammar rule with its source symbol; for example, `dff` contracts to `b` when the grammar contains $b \to$ `dff`. A **macro contraction** is a non-primitive rewrite equivalent to a 'sequential composition' of primitive contractions, such as `bae` $\to$ `a` by composing `ae` $\to$ `c` and `bc` $\to$ `a`. A **parallel contraction** applies a 'parallel composition' of primitive contractions simultaneously, such as `dffae` $\to$ `bc`. Together these two action types — macro and parallel — instantiate the procedural chunks introduced in Section 1: compressed multi-step procedures that can stand in for chains or combinations of primitive operations. Any non-primitive rewrite that is neither macro nor parallel is classified as **spurious**.

Thus, RL first strengthens primitive reduction behavior and only later shifts toward valid compositional procedures that were never present as primitive training actions. This trace-level phase transition explains the late gains in Figure 5: on the hardest buckets, primitive paths are least compatible with the generation budget, so valid compressed rewrites make previously over-budget solutions reachable.

4.1. Dynamics of Compositional Strategy Emergence

Recent RL post-training work reports phase-like changes in reasoning behavior: models first improve lower-level solution execution and later exhibit reflection, alternative approaches, or higher-level strategic behavior (DeepSeek-AI et al., 2025; Wang et al., 2025; Cheng et al., 2025). In natural-language settings, these phases are hard to audit because intermediate reasoning steps are not mechanically checkable. Our grammar gives a trace-level analogue: each generated rewrite can be classified against the known rule system, letting us measure which kind of procedure becomes more common during training.

Figure 2 shows a delayed transition. Early successful trajectories are dominated by primitive contractions. Around iteration 5,000, macro contractions begin to rise and overtake primitive contractions by roughly iteration 12,500, while parallel contractions emerge later and grow steadily.

4.2. Discovery and Consolidation of Macro Rules

The previous subsection showed that macro contractions become increasingly important during RL training. We now ask whether this increase reflects scattered one-off shortcuts, or whether RL builds a reusable repertoire of macro rules. We distinguish between a *macro action*, a single occurrence of a macro contraction in a trajectory, and a *macro rule*, the underlying rewrite pattern that may be used repeatedly across trajectories. We measure *discovery* as the first appearance of a macro rule, and *consolidation* as the later reuse of previously discovered macro rules.

Figure 3 shows that RL's macro behavior is not just an accumulation of isolated shortcuts. RL continues discovering new macro rules throughout training, with discovery peaking around iteration 10,000. Reuse rises in parallel: by the end of training, most macro actions emitted by the RL model are reapplications of rules observed at earlier checkpoints. RFT also discovers some macro rules, but at a lower rate and with weaker reuse: it finds shortcuts, but does not consolidate them into an equally stable repertoire.

This consolidation is the bridge from discovery to capability. A macro contraction compresses a sequence of primitive contractions into a single emitted rewrite, behaving like an implicit macro-action without any option set or high-level controller being provided (Sutton et al., 1999; Konidaris & Barto, 2009). The abstractions emerge inside the sequence model and are visible only because the trace can be audited. Thus RL accumulates capability not by relying on a fresh lucky shortcut each time, but by stabilizing discovered shortcuts into reusable policy structure that can be deployed on later problems.

5. Structured Versus Unstructured Exploration

The previous section established that RL discovers and consolidates valid macro rules more effectively than RFT. A natural alternative explanation is that RL simply explores more: perhaps it generates more non-primitive actions in total and therefore finds more valid shortcuts by chance. Figure 4 rules out this explanation. Panels (a) and (b) show that both methods generate valid non-primitive behavior: RFT discovers macro contractions and parallel contractions, but RL discovers many more of both, especially later in training. Panels (c) and (d) show the key difference. RFT accumulates many more spurious contractions, and its spurious-action ratio remains high throughout training. RL instead keeps spurious contractions low and shifts an increasing fraction of its non-primitive behavior toward valid structure. The difference is therefore not raw non-primitive volume, but selectivity.

A finite-group view of GRPO explains how this selectivity can arise. Consider a binary trajectory feature $F(\tau) \in \{0, 1\}$, such as the presence of a particular macro contraction, parallel contraction, or spurious contraction. For one prompt, GRPO samples a group of G completions. Let n be the number of successful completions, k^+ the number of successful completions containing F, and k^- the number of failed completions containing F.

With binary rewards and $0 < n < G$, the group mean is

$$\mu = \frac{n}{G}.$$

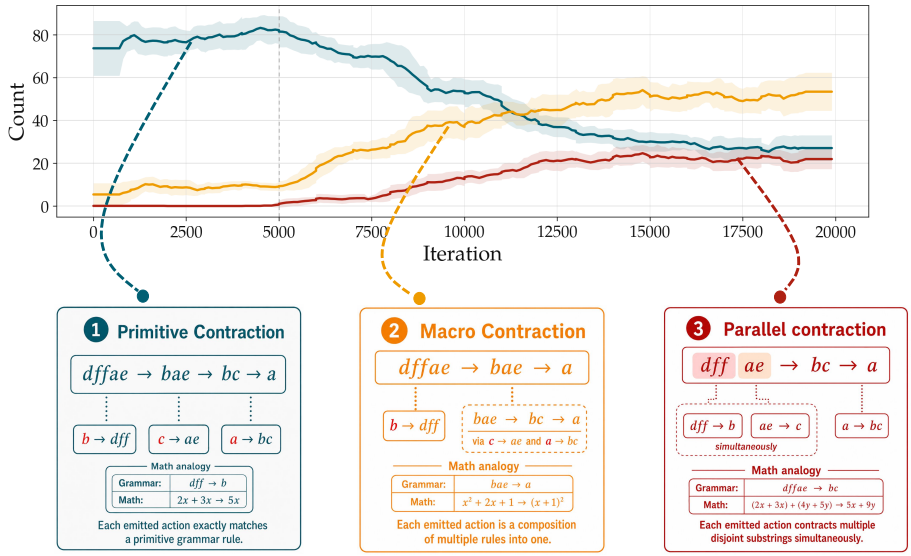

Figure 2. RL undergoes a delayed transition from primitive contractions to compressed valid shortcuts. Macro contractions accelerate and overtake primitive contractions around iteration 12,500; parallel contractions emerge later. Bottom examples show the structural analogy to algebraic simplification: primitive contractions correspond to local simplifications, macro contractions to reusable multi-step maneuvers, and parallel contractions to independent simplifications appliexd simultaneously.

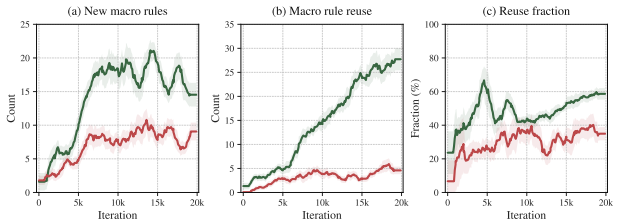

Figure 3. RL discovers and consolidates reusable macro rules. **(a)** Number of newly discovered macro rules at each checkpoint. **(b)** Reuse count of previously discovered macro rules. **(c)** Reuse fraction among all macro actions. RL continues discovering new macro rules while increasingly reusing earlier ones, indicating consolidation into a stable repertoire.

The group standard deviation is

$$\begin{aligned}\sigma^2 &= \frac{1}{G}\left[n(1-\mu)^2 + (G-n)\mu^2\right] \\ &= \frac{1}{G}\left[n\left(1-\frac{n}{G}\right)^2 + (G-n)\left(\frac{n}{G}\right)^2\right] \\ &= \frac{n(G-n)}{G^2},\end{aligned}$$

so

$$\sigma = \frac{\sqrt{n(G-n)}}{G}.$$

Thus successful and failed completions receive normalized advantages

$$A^+ = \frac{1-\mu}{\sigma} = \sqrt{\frac{G-n}{n}}, \qquad A^- = \frac{-\mu}{\sigma} = -\sqrt{\frac{n}{G-n}}.$$

Ignoring token-level constants, clipping effects, and the KL term, the feature-count contribution to the GRPO update is proportional to

$$\begin{aligned}A^+k^+ + A^-k^- &= \sqrt{\frac{G-n}{n}}\,k^+ - \sqrt{\frac{n}{G-n}}\,k^- \\ &= \sqrt{n(G-n)}\left(\frac{k^+}{n} - \frac{k^-}{G-n}\right).\end{aligned}$$

The prefactor is positive, so the sign is determined by the same-prompt success–failure contrast

$$\Delta_F = \frac{k^+}{n} - \frac{k^-}{G-n}.$$

Here k^+/n is the feature frequency among successful completions, while $k^-/(G-n)$ is its frequency among failed completions. GRPO therefore increases features that are more common in successful completions than failed completions from the same prompt, and suppresses features enriched in failures. This is a feature-level approximation rather than a full model of Transformer optimization, but it isolates the finite-sample contrast that distinguishes GRPO from imitation on accepted rollouts.

RFT uses the same sampled rollouts differently. It discards failed completions, retains only successful trajectories, and performs next-token prediction on the full accepted sequence. For the same prompt group, the only feature frequency available to RFT is

$$\hat{f}^+ = \frac{k^+}{n},$$

so the failure-side term $k^-/(G-n)$ does not appear in the supervised targets. As a result, if a successful trajectory contains a spurious rewrite alongside useful steps, that rewrite is cloned along with the useful ones. GRPO does not identify which individual action caused success, but its within-prompt contrast can give downward pressure to action patterns that appear frequently in failed rollouts, even if they also occur inside some successful ones.

This does not mean that RFT is a weak or misaligned baseline. In the idealized infinite-sample, infinitesimal-update limit, on-policy RFT with binary rewards points in the same first-order direction as the policy gradient. Let

$$J(\theta) = \mathbb{E}_{\tau\sim\pi_\theta}[R(\tau)].$$

Using the score-function identity,

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau\sim\pi_\theta}\left[R(\tau)\nabla_\theta \log \pi_\theta(\tau)\right].$$

For binary rewards,

$$\nabla_\theta J(\theta) = P_\theta(R=1)\,\mathbb{E}_{\tau\sim\pi_\theta(\tau|R=1)}\left[\nabla_\theta \log \pi_\theta(\tau)\right].$$

On-policy RFT samples from $\pi_{\theta_{\text{old}}}$, keeps successful trajectories, and minimizes

$$\mathcal{L}_{\text{RFT}}(\theta) = -\mathbb{E}_{\tau\sim\pi_{\theta_{\text{old}}}(\tau|R=1)}[\log \pi_\theta(\tau)].$$

Evaluated at $\theta = \theta_{\text{old}}$,

$$-\nabla_\theta \mathcal{L}_{\text{RFT}}(\theta_{\text{old}}) = \frac{1}{P(R=1)}\nabla_\theta J(\theta_{\text{old}}).$$

Thus ideal on-policy RFT is first-order aligned with binary-reward policy gradient, up to a positive scalar. This explains why RFT is a strong early baseline. The later divergence arises in the finite iterative setting: RFT repeatedly clones finite accepted trajectories, while GRPO uses negative advantages from below-average completions in the same prompt group.

The same finite-sample view also sharpens recent accounts of negative samples in RLVR. Prior work has argued that negative samples can help maintain exploration, preserve diversity, or prevent premature collapse (Wang et al., 2025; Tang et al., 2025). Our setting refines this framing by making the explored behaviors directly classifiable. Because every generated rewrite can be classified, we can see that RFT already produces abundant non-primitive behavior; the bottleneck is not exploration volume. The key difference is that RL filters this explored space: it amplifies non-primitive rewrites that are valid and reward-aligned, while suppressing spurious departures from the grammar. Thus the role of negative samples is not merely to keep the policy diverse, but to make exploration structurally selective.

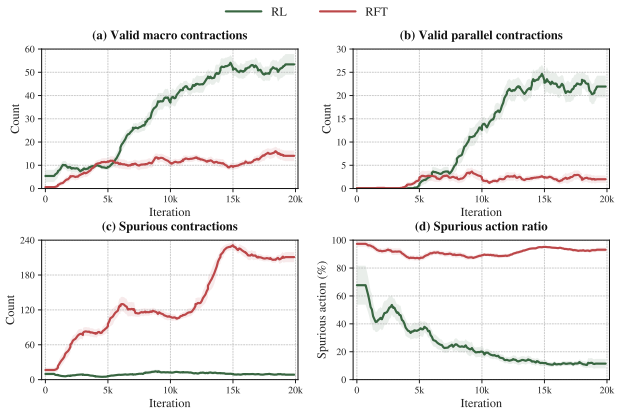

Figure 4. RL discovers valid non-primitive strategies while suppressing spurious shortcuts. **(a,b)** Counts of valid macro and parallel contractions. **(c)** Count of spurious contractions. **(d)** Fraction of non-primitive contractions that are spurious. RFT generates many non-primitive actions, but they are mostly invalid; RL produces fewer spurious rewrites and increasingly concentrates behavior into valid macro and parallel contractions.

6. RL Expands the Held-Out Capability Frontier

We first ask whether RL expands what the model can solve relative to the pretrained base policy, or merely reweights solutions already accessible under it. Our setting lets us answer this in a finite-budget, fully observable form: we can measure base-model reach under larger sampling budgets and inspect the structure of the resulting trajectories. We evaluate on the held-out difficulty buckets defined above, where higher difficulty corresponds to ordinary grammar-rule solutions becoming increasingly incompatible with the 256-token generation budget.

Figure 5 shows pass@16 during post-training. RFT improves faster at the beginning of training, especially on easier buckets, indicating that on-policy imitation can amplify solutions the pretrained model already sometimes produces (Xiong et al., 2025). RFT then plateaus, while RL improves more slowly at first but continues climbing after RFT saturates, with the largest gap appearing on the hardest buckets, where primitive contraction paths are least compatible with the generation budget.

To check whether the harder buckets are merely undersampled by the base model, we evaluate the pretrained policy at much larger sampling budgets (Table 1). Even at pass@1024, base-model success remains sparse and concentrated in the easiest over-budget buckets, with Buckets 4–5 remaining at zero. Thus, under the tested finite budgets, complete hard-bucket solutions are not practically available from the pretrained model, whereas RL makes them reliably accessible at pass@16.

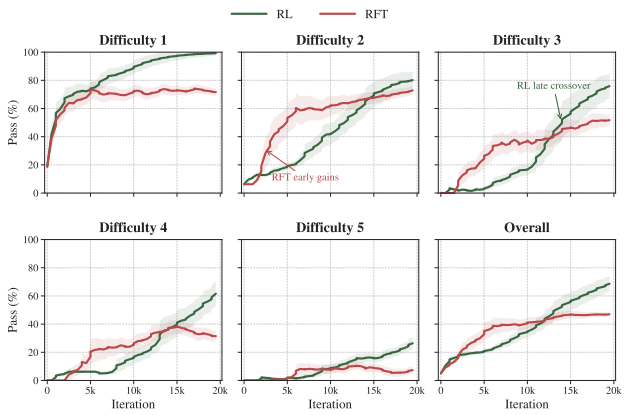

Figure 5. RL exhibits a delayed crossover over RFT, with the largest advantage on harder held-out problems. Pass@16 on contraction problems across five difficulty levels and overall. Mean over three seeds. RFT improves rapidly early in training but plateaus, while RL improves more gradually and continues climbing through the end of training.

Table 1. Base-model pass@k on held-out over-budget problems. Overall is over 80 problems; each bucket contains 16 problems.

k	Overall	B1	B2	B3	B4	B5
64	11.25%	50.00%	6.25%	0.00%	0.00%	0.00%
256	13.75%	50.00%	12.50%	6.25%	0.00%	0.00%
1024	17.50%	62.50%	18.75%	6.25%	0.00%	0.00%

This leaves the central mechanistic question: how does RL cross this finite frontier? On the harder buckets, primitive-only contraction paths exceed the generation budget, so solving them requires some form of shortcut. Such shortcuts could be invalid rewrites that exploit the outcome-only final-symbol reward, or valid compressed procedures assembled from lower-level reduction behavior. The next section distinguishes these possibilities by auditing the generated rewrite trajectories.

7. Pretraining as the Substrate for Strategy Emergence

The previous sections show that RL eventually discovers and consolidates valid non-primitive strategies. We now ask what the base model must already provide — not whether pretraining contains macro or parallel strategies, but whether it organizes primitive competence into reduction procedures that RL can later compress.

We vary how strongly contractions are locally upweighted during pretraining via a parameter ρ (Appendix C). Increasing ρ has two coupled effects: contractions occur more often overall, and once a reducible state is reached, the next step is more likely to also be a contraction — producing chained primitive reductions rather than isolated contraction events.

To separate these effects, we include a matched-fraction control: a low-ρ run with marginal contraction rate adjusted to match $\rho = 2$ but without the local upweighting that produces chaining. Figure 6a confirms the design: the control matches $\rho = 2$ on overall contraction rate (solid bars) but has less chaining (hatched bars, $P(C_t = 1 \mid C_{t-1} = 1)$).

Figures 6b,c show that this distinction matters. Low-ρ pretraining never reliably enters the macro or parallel regime; higher ρ produces earlier and stronger valid non-primitive strategies; the matched-fraction control fails to recover the $\rho = 2$ dynamics despite comparable marginal contraction exposure, behaving similarly to $\rho = 1$. What matters is not how often the model sees contractions, but whether they appear as sustained reduction procedures.

RL does not upweight a fully-formed latent strategy already present in the base model. Pretraining supplies weak procedural ingredients — primitive contractions, reduction states, and short contraction chains — that RL later selects, compresses, and consolidates into reusable higher-level strategies.

8. Discussion

The controlled grammar turns the "beyond the base model" question into an inspectable one. Higher pass@k in natural-language post-training can reflect better sampling of old behaviors, discovery of new procedures, or reward exploitation through invalid traces; here, every generated rewrite can be audited against the known grammar. This exposes a concrete mechanism: RL strengthens primitive reductions, forms valid compositional procedures, and consolidates them into a reusable repertoire.

The results point to a form of procedural chunking, or 'composition'. Chunking has long been studied in skill acquisition, for example in chess expertise (Chase & Simon, 1973). Mathematical practice has a similar flavor: multi-step derivations become reusable lemmas or standard maneuvers. Our grammar makes this process directly measurable: macro and parallel contractions are chunks that compress sequential or independent primitive reductions.

Our results challenge the view that RL merely reweights complete reasoning paths already present in the base distribution (Yue et al., 2025). In this environment, the complete hard-bucket solution is not practically present in the base model: even at pass@1024, the pretrained policy gets 0% on Buckets 4–5, while RL later solves them at pass@16. What is present are ingredients—primitive contractions and weakly supported local reduction chains. RL turns these ingredients into complete procedures: it first enters a primitive phase, strengthening local contractions and chaining them into longer reductions; only later does it enter a chunking phase, where macro and parallel contractions become reli-

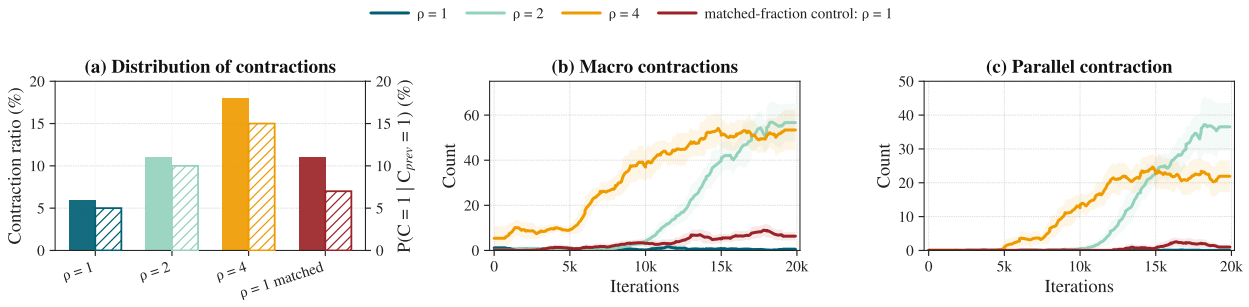

Figure 6. Strategy emergence depends on local contraction structure, not overall contraction frequency. **(a)** Solid bars (left axis): overall contraction rate. Hatched bars (right axis): contraction-after-contraction probability $P(C_t = 1 \mid C_{t-1} = 1)$. The matched-fraction control reaches a comparable overall rate to $\rho = 2$ but has substantially less chaining. **(b,c)** Macro and parallel contraction counts during RL post-training. Higher ρ produces earlier and stronger non-primitive strategy emergence; the matched-fraction control fails despite comparable marginal exposure.

able. Thus RL goes beyond the base model not by inventing from nothing (Wu et al., 2025), but by constructing usable strategies from weakly supported components. The base model provides the ingredients; RL builds the procedures.

This clarifies the comparison to Yuan et al. (Yuan et al., 2025). Their setting starts from a large pretrained model, so RL-induced composition is hard to separate from amplification of compositional patterns already supported by pretraining. It also leaves open which pretraining structures make composition reachable. Their task evaluates direct input-output transformations, whereas reasoning models often rely on extended traces where the procedure itself matters. Our setting addresses these confounds: pretraining is controlled and ablated, the primitive rule set is known, and the model emits the full rewrite trajectory. This lets us inspect which strategies make beyond-base performance possible.

The pretraining ablation shows that the relevant substrate is not merely exposure to the right primitive facts. Matching the marginal number of contraction examples does not recover the same downstream behavior. What matters is whether pretraining organizes primitive competence into reduction procedures that RL can later compress and consolidate. In this sense, pretraining supplies the procedural ingredients, while RL performs the selection and chunking that turn them into stable strategies.

The scope is controlled rather than universal: we do not claim that large language models must exhibit the same phases or taxonomy. But the grammar exposes quantities normally hidden in LLM post-training, and these results suggest that RL post-training is not only reward-driven reweighting; it can also organize weak procedural ingredients into reusable reasoning structure.

References

Chase, W. G. and Simon, H. A. Perception in chess. *Cognitive Psychology*, 4(1):55–81, 1973.

Cheng, D., Huang, S., Zhu, X., Dai, B., Zhao, W. X., Zhang, Z., and Wei, F. Reasoning with exploration: An entropy perspective. *arXiv preprint arXiv:2506.14758*, 2025.

Chu, T., Zhai, Y., Yang, J., Tong, S., Xie, S., Schuurmans, D., Le, Q. V., Levine, S., and Ma, Y. Sft memorizes, rl generalizes: A comparative study of foundation model post-training. *arXiv preprint arXiv:2501.17161*, 2025.

Cobbe, K., Kosaraju, V., Bavarian, M., Chen, M., Jun, H., Kaiser, L., Plappert, M., Tworek, J., Hilton, J., Nakano, R., Hesse, C., and Schulman, J. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

DeepSeek-AI, Guo, D., Yang, D., Zhang, H., Song, J., Zhang, R., Xu, R., Zhu, Q., Ma, S., Wang, P., Bi, X., et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

Konidaris, G. and Barto, A. Skill discovery in continuous reinforcement learning domains using skill chaining. In *Advances in Neural Information Processing Systems*, volume 22, 2009.

Lightman, H., Kosaraju, V., Burda, Y., Edwards, H., Baker, B., Lee, T., Leike, J., Schulman, J., Sutskever, I., and Cobbe, K. Let's verify step by step. *arXiv preprint arXiv:2305.20050*, 2023.

Liu, M., Diao, S., Lu, X., Hu, J., Dong, X., Choi, Y., Kautz, J., and Dong, Y. Prorl: Prolonged reinforcement learning expands reasoning boundaries in large language models. *arXiv preprint arXiv:2505.24864*, 2025a.

Liu, Z., Chen, C., Li, W., Qi, P., Pang, T., Du, C., Lee, W. S., and Lin, M. Understanding r1-zero-like training: A critical perspective. *arXiv preprint arXiv:2503.20783*, 2025b.

Shao, Z., Wang, P., Zhu, Q., Xu, R., Song, J., Zhang, M., Bi, X., Zhang, H., Li, Y. K., Wu, Y., and Guo, D. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

Sutton, R. S., Precup, D., and Singh, S. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112(1–2): 181–211, 1999.

Tang, X., Zhan, Y., Li, Z., Zhao, W. X., Zhang, Z., Wen, Z., Zhang, Z., and Zhou, J. Rethinking sample polarity in reinforcement learning with verifiable rewards. *arXiv preprint arXiv:2512.21625*, 2025.

Uesato, J., Kushman, N., Kumar, R., Song, F., Siegel, N., Wang, L., Creswell, A., Irving, G., and Higgins, I. Solving math word problems with process- and outcome-based feedback. *arXiv preprint arXiv:2211.14275*, 2022.

Wang, H., Xu, Q., Liu, C., Wu, J., Lin, F., and Chen, W. Emergent hierarchical reasoning in llms through reinforcement learning. *arXiv preprint arXiv:2509.03646*, 2025.

Wu, F., Xuan, W., Lu, X., Liu, M., Dong, Y., Harchaoui, Z., and Choi, Y. The invisible leash: Why rlvr may or may not escape its origin. *arXiv preprint arXiv:2507.14843*, 2025.

Xiong, W., Yao, J., Xu, Y., Pang, B., Wang, L., Sahoo, D., Li, J., Jiang, N., Zhang, T., Xiong, C., and Dong, H. A minimalist approach to llm reasoning: From rejection sampling to reinforce. *arXiv preprint arXiv:2504.11343*, 2025.

Yuan, L., Chen, W., Zhang, Y., Cui, G., Wang, H., You, Z., Ding, N., Liu, Z., Sun, M., and Peng, H. From f(x) and g(x) to f(g(x)): Llms learn new skills in rl by composing old ones. *arXiv preprint arXiv:2509.25123*, 2025.

Yue, Y., Chen, Z., Lu, R., Zhao, A., Wang, Z., Song, S., and Huang, G. Does reinforcement learning really incentivize reasoning capacity in llms beyond the base model? *arXiv preprint arXiv:2504.13837*, 2025.

Zelikman, E., Wu, Y., Mu, J., and Goodman, N. D. Star: Bootstrapping reasoning with reasoning. *arXiv preprint arXiv:2203.14465*, 2022.

Zhao, R., Meterez, A., Kakade, S., Pehlevan, C., Jelassi, S., and Malach, E. Echo chamber: Rl post-training amplifies behaviors learned in pretraining. *arXiv preprint arXiv:2504.07912*, 2025.

A. Additional Training and Data Generation Details

Grammar generation. Alphabet size is $N = 20$. Each character has between 1 and 3 right-hand sides drawn uniformly. Right-hand-side lengths follow a truncated Zipfian distribution over $[2, 6]$. All right-hand sides are globally unique.

Chain generation. Starting-word lengths are sampled uniformly from $[2, 12]$ and chain lengths from $[5, 50]$. For the default setup, the contraction weight is $\rho = 4$.

Model and optimization. The pretrained model uses 12 layers, hidden dimension 512, 8 attention heads, an MLP expansion ratio of 4, RoPE positional encoding, QK normalization, and a $\tanh$ soft-cap on logits. Optimization uses AdamW with a warm/flat/cool schedule, peak learning rate 8×10^{-4}, sequence length 512, bfloat16 training, and gradient clipping at norm 1.0.

Pretraining data generation. For each pretraining chain, we first sample a starting word length and a chain length from the configured distributions. Starting from a random word over the alphabet, the generator repeatedly enumerates all currently valid primitive moves: expansions at characters whose source symbol appears in the grammar, and contractions at substrings matching a right-hand side. One move is sampled from the configured rule weights, applied to the current word, and serialized as

```
lhs_word | action | rhs_word.
```

The resulting `rhs_word` becomes the next `lhs_word`. Thus pretraining chains are stochastic local rewrite trajectories, not solution-directed demonstrations. The generator records the start-word distribution, chain-length distribution, rule-weight distribution, contraction weight, and matched-fraction setting used for each dataset.

Post-training prompt generation. For RL/RFT post-training, prompts are generated by sampling a target symbol $c^\star$ and applying forward expansions to obtain a prompt word w_{prompt}. The model sees only w_{prompt} and the target; the expansion path used to construct the prompt is not provided. Prompts are filtered by the primitive contraction budget: Difficulty 1 fits within the generation budget, while higher difficulties require additional primitive contraction steps beyond the budget. The same prompt distribution is used for

GRPO and RFT; the methods differ only in how sampled completions are used for optimization.

Unless otherwise stated, learning curves are averaged over three training seeds

Compute. All experiments were run on a node with $4\times$ NVIDIA H100 GPUs.

B. GRPO and RFT Details

Reward. The reward is

$$r = \begin{cases} 1 & \text{if the trajectory is well-formed and ends with } c^\star, \\ 0 & \text{otherwise.} \end{cases}$$

A trajectory is *well-formed* if it is parseable as a sequence of `lhs_word | action | rhs_word` triples and each triple's `rhs_word` matches the next triple's `lhs_word`. This check is syntactic and does not require each intermediate rewrite to be grammar-valid.

GRPO. For each prompt x_i, GRPO samples G completions $\{y_{i,g}\}_{g=1}^{G}$ and assigns each a scalar reward $r_{i,g}$. Advantages are normalized within each prompt group:

$$A_{i,g} = \frac{r_{i,g} - \mu_i}{\sigma_i + \delta}, \qquad \mu_i = \frac{1}{G}\sum_{g=1}^{G} r_{i,g},$$

$$\sigma_i = \sqrt{\frac{1}{G}\sum_{g=1}^{G}(r_{i,g} - \mu_i)^2}.$$

Let

$$\rho_{i,g,t}(\theta) = \frac{\pi_\theta(y_{i,g,t} \mid x_i, y_{i,g,<t})}{\pi_{\text{old}}(y_{i,g,t} \mid x_i, y_{i,g,<t})}.$$

The clipped policy-gradient term is

$$\ell^{\text{clip}}_{i,g,t}(\theta) = \min(\rho_{i,g,t}A_{i,g}, \text{clip}(\rho_{i,g,t}, 1-\epsilon, 1+\epsilon)A_{i,g}).$$

The training loss is

$$\mathcal{L}(\theta) = -\frac{1}{\sum_{i,g,t} m_{i,g,t}} \sum_{i,g,t} m_{i,g,t}\left[\ell^{\text{clip}}_{i,g,t}(\theta) - \beta\kappa_{i,g,t}(\theta)\right],$$

where $m_{i,g,t}$ masks prompt and post-stop tokens and $\kappa_{i,g,t}$ is a reverse-KL penalty to the frozen pretrained reference policy. In our experiments, $G = 4$, $\epsilon = 0.2$, $\beta = 10^{-3}$, temperature is 0.8, top-k is 5, and the maximum generation length is 256 tokens.

RFT. RFT uses the same prompt distribution, group size $G = 4$, batch size, optimizer, and KL anchor as GRPO. For each prompt, it samples G completions, retains those with $r = 1$, and performs next-token prediction on the accepted rollouts using the same loss masking $m_{i,g,t}$. RFT differs from GRPO only in (i) it discards rather than down-weights failed rollouts, and (ii) its update is supervised rather than policy-gradient.

C. Pretraining Contraction Weighting

The contraction weight ρ is a local weight multiplier, not a direct contraction-frequency parameter. At each pretraining step, the generator enumerates the valid primitive expansion moves $E(w)$ and primitive contraction moves $C(w)$ from the current word w. Expansion moves keep their base weights, while contraction moves are multiplied by ρ before sampling the next rewrite.

For example, suppose the current word admits four valid primitive moves: three expansions and one contraction, all with unit base weight. With $\rho = 1$, all four moves have equal weight, so the contraction probability is

$$\frac{1}{3+1}.$$

With $\rho = 4$, the three expansions still have weight 1 each, but the contraction has weight 4, so the contraction probability becomes

$$\frac{4}{3+4}.$$

Thus increasing ρ makes contractions more likely locally whenever contractions are available. It does not directly impose a fixed global contraction rate.

More generally, if each move a has base rule weight $q(a)$, then the probability that the next move is a contraction is

$$P_\rho(C \mid w) = \frac{\rho\sum_{a\in C(w)} q(a)}{\sum_{a\in E(w)} q(a) + \rho\sum_{a\in C(w)} q(a)}.$$

The realized fraction of contractions over the full pretraining corpus depends on the grammar, the current word distribution, which moves are available, and the trajectory dynamics. This is why ρ should be interpreted as a local contraction weight rather than as the realized contraction rate.

High ρ also changes how contractions are arranged. Since contraction moves are locally upweighted whenever a reducible state is reached, the generator is more likely to remain in a reduction mode and produce consecutive contraction steps. Thus high ρ exposes the model not only to more contractions in aggregate, but also to more chained primitive reductions.

Matched-fraction control. The matched-fraction control separates aggregate contraction exposure from this local weighting mechanism. We choose a low-ρ generator and adjust its direction-sampling probability so that its realized marginal contraction rate is comparable to the $\rho = 2$ pretraining run. After choosing whether the next move is an expansion or contraction, a valid move within that direction is sampled.

The control is therefore not an unbiased $\rho = 1$ generator. It is intentionally biased to match the marginal number of

contraction examples in an intermediate contraction-weight setting. What it removes is the local reweighting of every available contraction move by a larger ρ. Consequently, contractions occur at a comparable overall rate, but they are less likely to appear as sustained contraction chains.

This distinction is what Figure 6 tests. If strategy emergence were explained only by the marginal number of contraction examples, the matched control should behave like the $\rho = 2$ model. Instead, it behaves much closer to the low-ρ model, suggesting that the organization of contractions across trajectories—especially exposure to reduction-mode behavior and chained primitive contractions—matters for later strategy emergence under RL.

D. Example Grammar and Pretraining Chains

$a \to \{\texttt{qs}\}$ $\quad b \to \{\texttt{ic}, \texttt{fe}, \texttt{sqjg}\}$
$c \to \{\texttt{jaocn}\}$ $\quad d \to \{\texttt{ba}, \texttt{nklt}, \texttt{si}\}$
$e \to \{\texttt{mj}\}$ $\quad f \to \{\texttt{hl}, \texttt{mq}\}$
$g \to \{\texttt{hheoj}, \texttt{jd}, \texttt{mi}\}$ $\quad h \to \{\texttt{aps}, \texttt{gna}\}$
$i \to \{\texttt{paolh}\}$ $\quad j \to \{\texttt{qsn}, \texttt{cnr}, \texttt{gd}\}$
$k \to \{\texttt{csj}\}$ $\quad l \to \{\texttt{dff}\}$
$m \to \{\texttt{fc}, \texttt{pt}\}$ $\quad n \to \{\texttt{dl}, \texttt{hfd}, \texttt{nq}\}$
$o \to \{\texttt{cj}, \texttt{hq}\}$ $\quad p \to \{\texttt{fk}\}$
$q \to \{\texttt{imr}, \texttt{nd}\}$ $\quad r \to \{\texttt{gn}\}$
$s \to \{\texttt{iq}, \texttt{bb}, \texttt{fflgj}\}$ $\quad t \to \{\texttt{ld}, \texttt{ka}\}$.

A pretraining example is a stochastic chain of local primitive rewrites. Each line applies one valid expansion or contraction, and the output word becomes the input to the next line. For example:

```
bqt | 0 E b>fe | feqt
feqt | 2 E q>nd | fendt
fendt | 4 E t>ka | fendka
fendka | 0 1 C b>fe | bndka
bndka | 0 E b>sqjg | sqjgndka
sqjgndka | 6 E k>csj | sqjgndcsja
```

Another sampled snippet is:

```
ltm | 0 E l>dff | dfftm
dfftm | 3 E t>ld | dffldm
dffldm | 5 E m>pt | dffldpt
dffldpt | 3 4 C t>ld | dfftpt
dfftpt | 3 E t>ka | dffkapt
dffkapt | 5 E p>fk | dffkafkt
```

These snippets illustrate the local nature of pretraining: the model sees grammar-consistent primitive rewrites, not the macro or parallel contractions analyzed in the main text.

D.1. Example RL/RFT Post-Training Prompts

The same prompt distribution is used for RL and RFT post-training. For RL, the model samples completions for each prompt and receives a binary reward. For RFT, successful on-policy completions from the same prompts are retained as next-token prediction targets. The model sees only the final expanded prompt word and the target symbol; the expansion chains below are shown only for exposition.

For readability, the examples below use an illustrative primitive-step budget $B = 3$ rather than the 256-token serialized budget used in the experiments. Thus Difficulty 1 fits within the budget, Difficulty 2 requires one additional primitive contraction step, and Difficulty 3 requires two additional primitive contraction steps.

Difficulty 1: primitive solution fits within budget. Starting from target symbol `a`, one possible prompt construction is:

```
a | 0 E a>qs | qs
qs | 0 E q>nd | nds
nds | 2 E s>iq | ndiq
```

The post-training prompt is therefore `ndiq`, with target `a`. A primitive contraction solution is:

```
ndiq | 0 1 C q>nd | qiq
qiq | 1 2 C s>iq | qs
qs | 0 1 C a>qs | a
```

This solution uses three primitive contractions, so it fits within the illustrative budget $B = 3$.

Difficulty 2: one primitive step beyond budget. Starting from target symbol `b`, one possible prompt construction is:

```
b | 0 E b>sqjg | sqjg
sqjg | 0 E s>bb | bbqjg
bbqjg | 2 E q>imr | bbimrjg
bbimrjg | 5 E j>gd | bbimrgdg
```

The post-training prompt is `bbimrgdg`, with target `b`. A primitive contraction solution is:

```
bbimrgdg | 2 4 C q>imr | bbqgdg
bbqgdg | 3 4 C j>gd | bbqjg
bbqjg | 0 1 C s>bb | sqjg
sqjg | 0 3 C b>sqjg | b
```

This requires four primitive contractions, one step beyond the illustrative budget.

Difficulty 3: two primitive steps beyond budget. Starting from target symbol `t`, one possible prompt construction is:

```
t | 0 E t>ka | ka
ka | 0 E k>csj | csja
csja | 0 E c>jaocn | jaocnsja
jaocnsja | 5 E s>iq | jaocniqja
jaocniqja | 7 E j>gd | jaocniqgda
```

The post-training prompt is `jaocniqgda`, with target `t`. A primitive contraction solution is:

```
jaocniqgda | 0 4 C c>jaocn | ciqgda
ciqgda | 1 2 C s>iq | csgda
csgda | 2 3 C j>gd | csja
csja | 0 2 C k>csj | ka
ka | 0 1 C t>ka | t
```

This requires five primitive contractions, two steps beyond the illustrative budget.

These examples illustrate the post-training data construction. RL and RFT use the same prompts and final-answer reward; they differ only in how sampled rollouts are used for optimization. GRPO updates from both successful and failed completions through group-relative advantages, while RFT retains only successful completions as supervised targets.