A Proof of Associativity of Binary Operator

Recall that $\oplus$ is defined as:

$$a_i \oplus a_j := \begin{cases} (a_{j,a} \odot a_{i,a}, a_{j,a} \otimes a_{i,b} + a_{j,b}, a_{i,c}) & \text{if } a_{j,c} = 0 \\ (a_{j,a}, a_{j,b}, a_{j,c}) & \text{if } a_{j,c} = 1 \end{cases}$$

This is equivalent to the following:

$$a_i \oplus a_j := \begin{cases} ((a_i \bullet a_j)_a, (a_i \bullet a_j)_b, a_{i,c}) & \text{if } a_{j,c} = 0 \\ a_j & \text{if } a_{j,c} = 1 \end{cases}$$

where $\bullet$ is S5's binary operator defined in Equation 5. Note that $\bullet$'s associativity was proved in Smith et al. [38]. Using this, we can prove the associativity of $\oplus$.

Let x, y, and z refer to three elements. We will prove that for all possible values of x, y, and z, $\oplus$ retains associativity.

Case 1: $z_c = 1$

$$\begin{aligned} (x \oplus y) \oplus z &= z && (10) \\ &= y \oplus z && (11) \\ &= x \oplus (y \oplus z) && (12) \end{aligned}$$

Case 2: $z_c = 0$ and $y_c = 1$

$$\begin{aligned} (x \oplus y) \oplus z &= y \oplus z && (13) \\ \text{Note that } (y \oplus z)_c = 1 \text{ thus,} & && (14) \\ &= x \oplus (y \oplus z) && (15) \end{aligned}$$

Case 3: $z_c = 0$ and $y_c = 0$

$$\begin{aligned} (x \oplus y) \oplus z &= ((x \bullet y)_a, (x \bullet y)_b, x_c) \oplus z && (16) \\ &= (((x \bullet y) \bullet z)_a, ((x \bullet y) \bullet z)_b, x_c) && (17) \\ &= ((x \bullet (y \bullet z))_a, (x \bullet (y \bullet z))_b, x_c) && (18) \\ &= x \oplus ((y \bullet z)_a, (y \bullet z)_b, y_c) && (19) \\ &= x \oplus (y \oplus z) && (20) \end{aligned}$$

B Hyperparameters

Table 3: Hyperparameters for training A2C on Bsuite

Parameter	Value
Adam Learning Rate	3e-4
Entropy Coefficient	0.0
Encoder Layer Sizes	[256, 256]
Number of Recurrent Layers	1
Size of Recurrent Layer	256
Discount γ	0.99
TD λ	0.9
Number of Environments	1
Unroll Length	32
Number of Episodes	10000
Activation Function	ReLU

Table 4: Hyperparameters for training PPO on POPGym

Parameter	Value
Adam Learning Rate	5e-5
Number of Environments	64
Unroll Length	1024
Number of Timesteps	15e6
Number of Epochs	30
Number of Minibatches	8
Discount γ	0.99
GAE λ	1.0
Clipping Coefficient ϵ	0.2
Entropy Coefficient	0.0
Value Function Weight	1.0
Maximum Gradient Norm	0.5
Learning Rate Annealing	None
Activation Function	LeakyReLU
Encoder Layer Sizes	[128, 256]
Recurrent Layer Hidden Size	256
Action Decoder Layer Sizes	[128, 128]
Value Decoder Layer Sizes	[128, 128]
S5 Layers	4
S5 Hidden Size	256
S5 Discretization	ZOH
S5 Δ min	0.001
S5 Δ max	0.1

Table 5: Hyperparameters for training Muesli on Multi-Environment Meta-RL. These experiments were run using 64 TPUv3's.

Parameter	Value
Adam Learning Rate	3e-4
Value Function Weight	1.0
Muesli Auxiliary Loss Weight	3.0
Muesli Model Unroll Length	1.0
Encoder Layer Sizes	[512, 512]
Number of Environments	1024
Discount γ	0.995
Rollout Length	1000
Offline Data Fraction	0.0
Total Frames	2e9
LSTM Hidden Size	512
Projected Observation Size	12
Projected Action Size	2
S5 Layers	10
S5 Hidden Size	256
S5 Discretization	ZOH
S5 Δ min	0.001
S5 Δ max	0.1

C POPGym Discussion

	Stateless CartPole Hard	Noisy Stateless CartPole Hard	Stateless Pendulum Hard	Noisy Stateless Pendulum Hard	Repeat Previous Hard
S5 (ours)	$\mathbf{1.0 \pm 0.0}$	$\mathbf{0.28 \pm 0.0}$	$\mathbf{0.79 \pm 0.01}$	0.55 ± 0.01	$\mathbf{0.91 \pm 0.01}$
GRU (ours)	$\mathbf{1.0 \pm 0.0}$	0.27 ± 0.0	0.75 ± 0.0	$\mathbf{0.61 \pm 0.01}$	-0.46 ± 0.01
MLP (ours)	0.26 ± 0.0	0.22 ± 0.0	0.41 ± 0.02	0.34 ± 0.01	-0.48 ± 0.00
GRU	$\mathbf{1.000 \pm 0.000}$	0.390 ± 0.007	$\mathbf{0.828 \pm 0.001}$	$\mathbf{0.657 \pm 0.002}$	-0.428 ± 0.002
MLP	0.265 ± 0.002	0.229 ± 0.002	0.477 ± 0.030	0.351 ± 0.012	-0.486 ± 0.002
IndRNN	$\mathbf{1.000 \pm 0.000}$	$\mathbf{0.404 \pm 0.005}$	0.804 ± 0.023	0.521 ± 0.109	-0.384 ± 0.013
LMU	0.987 ± 0.007	0.352 ± 0.019	0.806 ± 0.006	0.563 ± 0.014	$\mathbf{0.191 \pm 0.041}$
S4D	0.127 ± 0.026	0.207 ± 0.007	0.303 ± 0.014	0.289 ± 0.011	-0.491 ± 0.001
FART	$\mathbf{0.996 \pm 0.000}$	0.366 ± 0.002	0.698 ± 0.077	0.553 ± 0.007	-0.485 ± 0.001

Table 6: Results in POPGym's suite. The reported number is the max-mean episodic reward (MMER) used in Morad et al. [27]. To calculate this, we take the mean episodic reward for each epoch, and then take the maximum over all epochs. For our results above, the mean and standard deviation across eight seeds are reported. The results below are selected architectures from Morad et al. [27], which also includes the best-performing one from each environment. They report the mean and standard deviation across three trials.

Morad et al. [27] used RLLib's [24] implementation of PPO, which differs significantly from standard implementations of PPO. It uses a dynamic KL-divergence coefficient on top of the clipped surrogate objective of PPO [37]. Furthermore, they use advanced orchestration to return full episode trajectories, rather than using the more commonly-studied "stored state" [19] strategy.

Instead, we follow the design decisions outlined in the StableBaselines3 [35] and CleanRL's [18] recurrent PPO implementations. While this results in different results shown in Table 6 for the same architecture, it recovers similar performance across the environments. Notably, our S5 architecture far outperforms the best performing architecture in Morad et al. [27] in the "RepeatPreviousHard" environment.

We used the learning rate, number of environments, unroll length, timesteps, epochs, and minibatches, GAE λ, and model architectures from Morad et al. [27]. However, we used the standard clipping coefficient ϵ of 0.2 instead of 0.3 to account for the lack of a dynamic KL-divergence coefficient. Note that we also adjusted the S5 architecture to contain approximately the same number of *parameters* as the GRU implementation instead of matching the size of the *hidden state*, which was done in Morad et al. [27].

We did not evaluate our architecture across the full POPGym suite. To enable more rapid experimentation, we implement our algorithms and environments end-to-end in JAX [3, 25]. While the original POPGym results took 2 hours per trial with a GRU with a Quadro RTX 8000 and 24 CPUs, we could run our experiments using only 3 *minutes* per trial on an NVIDIA A40. Because of this, we selected environments from Morad et al. [27] to implement in JAX. We chose the CartPole, Pendulum, and Repeat environments because they are modified versions of existing environments in Lange [22]. We found that the "Easy" and "Medium" versions of these environments were not informative, as most models perform well on them and only report the "Hard" difficulty results.

We attach our code in the supplementary materials.