# MODULARITY IN REINFORCEMENT LEARNING VIA ALGORITHMIC INDEPENDENCE IN CREDIT ASSIGNMENT

**Anonymous authors**
Paper under double-blind review

## ABSTRACT

Many transfer problems require re-using previously optimal decisions for solving new tasks, which suggests the need for learning algorithms that can modify the mechanisms for choosing certain actions independently of those for choosing others. However, there is currently no formalism nor theory for how to achieve this kind of modular credit assignment. To answer this question, we define modular credit assignment as a constraint on minimizing the algorithmic mutual information among feedback signals for different decisions. We introduce what we call the modularity criterion for testing whether a learning algorithm satisfies this constraint by performing causal analysis on the algorithm itself. We generalize the recently proposed societal decision-making framework as a more granular formalism than the Markov decision process to prove that for decision sequences that do not contain cycles, certain single-step temporal difference action-value methods meet this criterion while all policy-gradient methods do not. Empirical evidence suggests that such action-value methods are more sample efficient than policy-gradient methods on transfer problems that require only sparse changes to a sequence of previously optimal decisions.

## 1 INTRODUCTION

Many transfer problems require re-using some previously optimal decisions while independently modifying others. Existing work on modularity in machine learning has largely focused on enforcing the independence of mechanisms in the forward execution of a system (§C). However, a main result we show is that static factorization of the mechanisms of a learner does not guarantee that those mechanisms remain independent conditioned on the training history after a credit assignment update: feedback signals produced by the credit assignment mechanism (CAM) must be independent as well. Given the known benefits of modular systems (Sussman, 2007), we would expect that when the environment changes, CAMs that produce independent feedback would enable previously optimal decisions to be re-used with less interference from other decisions, thus transfering more efficiently.

Our main contributions are: to formally define the constraint, which we call the **modularity constraint**, on the algorithmic independence among the feedback signals that modular credit assignment mechanisms must satisfy; to translate the incomputable modularity constraint into what we call the **modularity criterion** for practically designing learning algorithms with modular credit assignment; and to theoretically evaluate the major classes of reinforcement learning (RL) algorithms against the modularity criterion. We show that the recently proposed cloned Vickrey society (Chang et al., 2020) satisfies this criterion in the most general setting, single-step temporal-difference methods satisfy it in the tabular setting, and policy gradient methods never satisfy it. We present empirical evidence that suggest algorithms with modular credit assignment are more sample efficient in transfer problems that require only sparse modifications to a sequence of previously optimal decisions.

These contributions required overcoming several theoretical challenges. For one, we need a formalism for learning that is based on algorithmic information rather than standard Shannon information because the latter is not defined for deterministic credit assignment operations. To overcome the incomputability of algorithmic information, we draw upon algorithmic causality (Janzing & Schölkopf, 2010) to develop the **algorithmic causal model of learning** (ACL), which represents

learning algorithms as causal graphs that can be inspected to verify the modularity constraint. To construct ACL, we draw upon $\lambda$-calculus to reconcile the computational graph of execution, which treats learnable mechanisms as functions, with the computational graph of learning, which treats them as data. Lastly, because the Markov decision process (MDP) is not granular enough for distinguishing feedback to the possible values an action variable could have taken, we generalize the recently-proposed societal decision-making framework (Chang et al., 2020) to a more granular computational graph for evaluating standard discrete-action RL algorithms against our criterion.

## 2 Defining Modular Credit Assignment

We first separate the **model of execution**, which specifies the computational graph of the learner, from the **model of credit assignment**, which specifies the computational graph of the learning algorithm. Definitions, background, and proofs are in the Appendix.

**Model of execution.** We represent a system of learnable mechanisms as a computational graph $\mathbf{G}$ (Def. 10) whose factor nodes, i.e. functions, represent learnable mechanisms $\mathbf{f} = \{\mathtt{f}^1, ..., \mathtt{f}^K\}$ and data nodes represent interface between the functions. Let $x_{\mathtt{f}} := (x_{\mathtt{f}}^{\text{in}}, x_{\mathtt{f}}^{\text{out}})$ denote input and output data nodes of function $\mathtt{f}$. The forward execution generates a topologically-sorted **execution trace** $\boldsymbol{\tau} = (x_{\mathtt{f}_1}, ..., x_{\mathtt{f}_M})$ that records the inputs and outputs of the $M$ functions within the computation.

**Model of credit assignment.** We represent the credit assignment mechanism $\Pi$ as a context-conditioned policy $\Pi(\boldsymbol{\tau}, \mathbf{f}) \rightarrow \boldsymbol{\delta}$ that generates modifications $\boldsymbol{\delta} = (\delta_1, ..., \delta_M)$ to the functions $\mathbf{f}$ of the system, given $\boldsymbol{\tau}$ as context. Then the computational graph of a learning algorithm can be represented as a controlled Markov process (CMP) $\mathbf{C}$ with states $\mathbf{f}$ and actions $\boldsymbol{\delta}$. The type of learning algorithm defines the transition function of $\mathbf{C}$, which we will call $\mathtt{UPDATE}(\mathbf{f}, \boldsymbol{\delta}) \rightarrow \mathbf{f}'$, and consequently the datatype of the feedback signals $\boldsymbol{\delta}$. For a learning algorithm based on gradient descent, $\delta_i^k$ is the gradient of $\mathbf{G}$'s learning objective with respect to the function $\mathtt{f}^k$ that participated at step $i$ of the execution trace. [1] Then $\mathtt{UPDATE}$ performs the following parallel operation $\mathtt{UPDATE}(\mathtt{f}^k, \sum_i \delta_i^k) \rightarrow \mathtt{f}^{k\prime}$ over all functions $\mathtt{f}^k$ where the functional form of $\mathtt{UPDATE}$ depends on the choice of optimizer like stochastic gradient descent, Adam (Kingma & Ba, 2014), etc. We assume a gradient-based learning algorithm, but our results hold for any other optimizer with the structure of $\mathtt{UPDATE}$ defined above.

We now adapt the standard definition of modularity to take the dynamic evolution of the learner into account. Modularity has traditionally be defined for static computational graphs as the algorithmic independence of functions (Pearl, 2009; Peters et al., 2017):

**Definition 1** (**static modularity**). $\forall k \neq j, \quad I\left(f^k : f^j\right) \stackrel{+}{=} 0,$

but this definition is not useful when we expect the functions to learn to share information over the course of learning. We can instead define modularity of the functions at iteration $n+1$ of learning still as algorithmic independence, but relative to the **learning history** $\mathcal{T}_n = (\mathbf{f}_1, \boldsymbol{\tau}_1, ..., \mathbf{f}_n, \boldsymbol{\tau}_n)$ as background information (Janzing & Schölkopf, 2010, §2.3). Then, because $\mathbf{C}$ is a Markov process, it is sufficient to condition on only the latest execution trace $\boldsymbol{\tau}_n$ and past learner parameters $\mathbf{f}_n$:

**Definition 2** (**dynamic modularity**). $\forall k \neq j, \quad I\left(f^{k_{n+1}} : f^{j_{n+1}} \;\middle|\; \boldsymbol{\tau}_n, \mathbf{f}_n\right) \stackrel{+}{=} 0.$

$\mathtt{f}^{k_{n+1}}$ denotes the $k$th function after $n$ credit assignment updates. Conditioning on the history $\mathcal{T}_n$ makes sense because any state of the learner $\mathbf{f}_n$ can be considered the beginning of a learning process. Static modularity is a special case of dynamic modularity when $n = 0$. As we will see, static modularity does not generally imply dynamic modularity.

We are interested in understanding what constraint the CAM must satisfy for dynamic modularity to hold for $\mathbf{G}$ at every iteration of learning. Satisfying such a constraint would allow functions to be modified independently without changing in other functions.Given execution trace $\boldsymbol{\tau}$ and previous learner parameters $\mathbf{f}$, we define the **modularity constraint** as that which imposes that the gradients $\delta_1, ..., \delta_M$ be jointly independent:

[1] Superscripts denote identity, subscripts denote position in the execution trace. $\mathtt{f}_i^k$, $\mathtt{f}_{i'}^k$ refer to calls to the $k$th function at distinct points $i$ and $i'$ of execution. $\delta_i^k$ is the feedback signal to $\mathtt{f}_i^k$.

**Definition 3** (**modularity constraint**). $I\left(\delta_1, ..., \delta_M \mid \boldsymbol{\tau}, \boldsymbol{f}\right) \stackrel{+}{=} 0.$

Define a **modular credit assignment mechanism** as one that satisfies the modularity constraint. If **G** were statically modular to start (i.e. its functions share no weights) then learning with a modular CAM enforces dynamic modularity:

**Theorem 1** (**modular credit assignment**). *Dynamic modularity is enforced at learning iteration $n$ if and only if static modularity holds, i.e. $I\left(f^{k_1} : f^{j_1}\right) \stackrel{+}{=} 0$ for all $k \neq j$, and the CAM satisfies the modularity constraint.*

The key observation, as alluded to in the introduction, is that static modularity alone does not guarantee dynamic modularity – the gradients that the CAM produces must also be independent as well. We observe that all modular CAMs must be factorized in the following way:

**Theorem 2** (**modular factorization**). *The credit assignment mechanism $\Pi(\boldsymbol{\tau}, \boldsymbol{f}) \rightarrow \boldsymbol{\delta}$ is modular if and only if $K\left(\boldsymbol{\delta} \mid \boldsymbol{\tau}, \boldsymbol{f}\right) \stackrel{+}{=} \sum_{i=1}^{M} K\left(\delta_i \mid \boldsymbol{\tau}, \boldsymbol{f}\right)$.*

For modular CAMs, the complexity of computing feedback for the entire system is minimal because all redundant information among the gradients has been "squeezed out." This connection between simplicity and modularity is another way of understanding why if a CAM were not modular it would be impossible for $\Pi$ to modify a function without simultaneously modifying another, other than due to non-generic instances when $\delta_i$ has a simple description, i.e. $\delta_i = 0$, which, unless imposed, are not likely to hold over all iterations of learning.

## 3 An Algorithmic Causal Model of Learning

We have shown that modular CAMs are required to enforce dynamic modularity in the general case, but algorithmic mutual information is generally incomputable so the modularity constraint is not practical for enabling us to evaluate existing CAMs. Our key idea is to leverage the observation from Janzing & Schölkopf (2010) that we can represent computational graphs as causal graphs to represent the learning process itself as a causal graph. We thus bypass the incomputability of algorithmic mutual information by translating the modularity constraint into a criterion on $d$-separation that can be assessed by direct inspection.

To construct this causal graph, we reconcile the computational graph of execution **G**, which uses functions $\mathtt{f}$ to process $x$ as data, with that of credit assignment **C**, which uses $\Pi$ to process $\mathtt{f}$ as data. We can "flatten" **G** to treat $\mathtt{f}$ at the same level of operation as $x$ by introducing the higher-order operation `APPLY` (Abelson & Sussman, 1996), a principle in $\lambda$-calculus known as $\beta$-reduction, into the graph construction such that $\forall \mathtt{f}, x,\ \mathtt{APPLY}(\mathtt{f}, x) := \mathtt{f}(x)$. Treating function application as a factor node enables us to treat both $\mathtt{f}$ and $x$ as data nodes in the same causal graph.

**Lemma 3** (**algorithmic causal model learning**). *Given a model of execution **G** and of credit assignment **C**, define the **algorithmic causal model of learning** (ACL) as a computational graph $\mathbb{L}$ of the learning process with data nodes $x$, $\mathtt{f}$, and $\delta$ and factor nodes `APPLY`, $\Pi$, and `UPDATE`, as well as the internal data and factor nodes that constitute the structure of `APPLY` and $\Pi$ which vary by the learning algorithm. Then $\mathbb{L}$ is a double trellis over $x$ and $\mathtt{f}$ generated via an inner loop governed by $\mathtt{APPLY}(\mathtt{f}, x^{in}_{\mathtt{f}}) \rightarrow x^{out}_{\mathtt{f}}$ that generates the execution trace $\boldsymbol{\tau}$ and an outer loop structured as the CMP **C** governed by two algorithmically independent operations $\Pi(\boldsymbol{\tau}, \boldsymbol{f}) \rightarrow \boldsymbol{\delta}$ and $\mathtt{UPDATE}(\boldsymbol{f}, \boldsymbol{\delta}) \rightarrow \boldsymbol{f}'$. Then the strings $x^{in}_{\mathtt{f}^k_i}$, $x^{out}_{\mathtt{f}^k_i}$, $\mathtt{f}^k_i$, $\delta^k_i$, and the internal data nodes of `APPLY` and $\Pi$, for all steps of credit assignment $n$, satisfy the algorithmic causal Markov condition with respect to $\mathbb{L}$.*

Representing learning algorithms as causal graphs via Lemma 3 brings tools from algorithmic causality (Janzing & Schölkopf, 2010) to bear on analyzing general learning algorithms.

**Theorem 4** (**modularity criterion**). *If $\mathbb{L}$ is faithful, the modularity constraint holds if and only if for all $i$ the outputs $\delta_i$ and $\delta_{\neq i}$ of $\Pi$ are $d$-separated by its inputs $\boldsymbol{\tau}$ and $\boldsymbol{f}$.*

Thm. 4 enables us to evaluate, before any training, whether a learning algorithm satisfies the modularity constraint by simply inspecting $\mathbb{L}$ for $d$-separation, giving us a practical tool to both design and evaluate learning algorithms on the basis of the modularity of their CAMs. We generally have access to the true causal graph, because the learning algorithm was programmed by us. Therefore,

assuming that $\mathbb{L}$ is faithful is as reasonable as assuming the operations in the programming language used to program the learning algorithm has no side-effects that interfere with $\mathbb{L}$'s data nodes, which is reasonable since we assume the programming language is agnostic to the learning task. Henceforth we assume $\mathbb{L}$ is faithful, unless otherwise stated.

## 4 THE MODULARITY OF RL ALGORITHMS

We have developed a formalism for dynamic modularity, a constraint on the CAM for enforcing this property, and a criterion for evaluating whether the constraint is satisfied. We now apply these tools to evaluate the major classes of RL algorithms (Sutton & Barto, 2020) – action-value (AV) and policy-gradient (PG) methods. The benefit of the modularity criterion (MCn) is that we can use graphical language for our analysis, which simplifies much of the proofs. For RL the functions $\mathbf{f}$ are **decision mechanisms** that each control a different value of the action variable (e.g. the probability of the action for PG methods, or the $Q$-value of the action for AV methods).

**Which RL algorithms satisfy the modularity criterion?** MCn can be violated through shared hidden variables in $\Pi$ that couple together gradients $\delta$, which cause the $\delta_i^k$'s to not be $d$-separated given $\boldsymbol{\tau}$ and $\mathbf{f}$. This hidden variable is the normalization term of the softmax for PG methods and the sum of estimated returns for $n$-step temporal difference methods where $n > 1$, abbrv. TD($n > 1$). However, MCn is not violated by TD(0) methods if the decision sequence $\boldsymbol{\tau}$ does not contain cycles.

**Corollary 4.1.** *All PG methods do not satisfy MCn.*

**Corollary 4.2.** *All TD($n > 1$) methods do not satisfy MCn.*

**Corollary 4.3.** *TD(0) methods satisfy MCn for acyclic $\boldsymbol{\tau}$.*

**Which RL algorithms enforce dynamic modularity?** Having narrowed down the classes of RL algorithms whose causal structure satisfy MCn to TD(0) methods, we consider the models of execution $\mathbf{G}$ of individual TD(0) methods – $Q$-learning (Watkins & Dayan, 1992), SARSA (Rummery & Niranjan, 1994), and CVS (Chang et al., 2020) – on whether they satisfy static modularity (and thereby dynamic modularity), i.e. Thm. 1 holds. We assume that the initial parameters of $\mathbf{f}$ are randomly initialized, so the only possible source of dependence among $\mathbf{f}$ is if they share parameters.

**Corollary 1.1.** *Thm. 1 holds for Q-learning, SARSA, CVS in the tabular setting.*

**Corollary 1.2.** *In the general function approximation setting, Thm. 1 holds for CVS.*

The main implication is that if we want dynamic modularity, then from the model of credit assignment we need the CAM to produce algorithmically independent gradients, and from the model of execution we need the decision mechanisms not share weights. Then, an RL algorithm with dynamic modularity makes it possible for individual decision mechanisms to be modified independently without an accompanying modification to other decision mechanisms, which is what we want for transferring to tasks whose optimal solution re-use previously optimal decisions.

## 5 DISCUSSION

We have (1) generalized the static definition of modularity to encompass dynamic learning systems, (2) proven the necessity of the modularity constraint that the credit assignment mechanism must satisfy to enforce this dynamic modularity, (3) introduced the connection between learning algorithms and causal graphs as means to practically achieve this constraint via the modularity criterion, and (4) evaluated a major classes of RL algorithms against this criterion. In §E we also show that (5) dynamically modular RL methods tend to correlate with more efficient transfer when previously optimal decisions must be reused. The key conceptual contribution we add to previous work on modularity is that the concept must be understood through the interaction between the learning system and credit assignment mechanism, not just from the learning system alone. The connection we have established among credit assignment, modularity, and algorithmic information theory, in particular the link between learning algorithms and algorithmic causality, opens many opportunities for future work, such as new ways of formalizing inductive bias in the algorithmic causal structure of learning systems *and* the learning algorithms that modify them.

## REFERENCES


Harold Abelson and Gerald Jay Sussman. *Structure and interpretation of computer programs*. The MIT Press, 1996.

Joshua Achiam, Ethan Knight, and Pieter Abbeel. Towards characterizing divergence in deep q-learning. *arXiv preprint arXiv:1903.08894*, 2019.

Ferran Alet, Tomás Lozano-Pérez, and Leslie P Kaelbling. Modular meta-learning. *arXiv preprint arXiv:1806.10166*, 2018.

Jacob Andreas, Marcus Rohrbach, Trevor Darrell, and Dan Klein. Neural module networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 39–48, 2016.

David Balduzzi. Cortical prediction markets. *arXiv preprint arXiv:1401.1465*, 2014.

Eric B Baum. Toward a model of mind as a laissez-faire economy of idiots. In *ICML*, pp. 28–36, 1996.

Gregory J Chaitin. On the length of programs for computing finite binary sequences. *Journal of the ACM (JACM)*, 13(4):547–569, 1966.

Gregory J Chaitin. A theory of program size formally identical to information theory. *Journal of the ACM (JACM)*, 22(3):329–340, 1975.

Michael Chang, Sidhant Kaushik, S Matthew Weinberg, Thomas L Griffiths, and Sergey Levine. Decentralized reinforcement learning: Global decision-making via local economic transactions. *arXiv preprint arXiv:2007.02382*, 2020.

Michael B Chang, Abhishek Gupta, Sergey Levine, and Thomas L Griffiths. Automatically composing representation transformations as a means for generalization. *arXiv preprint arXiv:1807.04640*, 2018.

Róbert Csordás, Sjoerd van Steenkiste, and Jürgen Schmidhuber. Are neural nets modular? inspecting functional modularity through differentiable weight masks. *arXiv preprint arXiv:2010.02066*, 2020.

Haskell Brooks Curry, Robert Feys, William Craig, J Roger Hindley, and Jonathan P Seldin. *Combinatory logic*, volume 1. North-Holland Amsterdam, 1958.

Coline Devin, Abhishek Gupta, Trevor Darrell, Pieter Abbeel, and Sergey Levine. Learning modular neural network policies for multi-task and multi-robot transfer. In *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, pp. 2169–2176. IEEE, 2017.

Edsger W Dijkstra. Letters to the editor: go to statement considered harmful. *Communications of the ACM*, 11(3):147–148, 1968.

Daniel Filan, Shlomi Hod, Cody Wild, Andrew Critch, and Stuart Russell. Neural networks are surprisingly modular. *arXiv preprint arXiv:2003.04881*, 2020.

Justin Fu, Aviral Kumar, Matthew Soh, and Sergey Levine. Diagnosing bottlenecks in deep q-learning algorithms. In *International Conference on Machine Learning*, pp. 2021–2030. PMLR, 2019.

Péter Gács, John T Tromp, and Paul MB Vitányi. Algorithmic statistics. *IEEE Transactions on Information Theory*, 47(6):2443–2463, 2001.

Noah D Goodman, Joshua B. Tenenbaum, and The ProbMods Contributors. Probabilistic Models of Cognition. `http://probmods.org/v2`, 2016. Accessed: 2021-1-29.

Anirudh Goyal, Alex Lamb, Jordan Hoffmann, Shagun Sodhani, Sergey Levine, Yoshua Bengio, and Bernhard Schölkopf. Recurrent independent mechanisms. *arXiv preprint arXiv:1909.10893*, 2019.

Daniel M Hausman and James Woodward. Independence, invariance and the causal markov condition. *The British journal for the philosophy of science*, 50(4):521–583, 1999.

Dominik Janzing and Bernhard Schölkopf. Causal inference using the algorithmic markov condition. *IEEE Transactions on Information Theory*, 56(10):5168–5194, 2010.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Louis Kirsch, Julius Kunze, and David Barber. Modular networks: Learning to decompose neural computation. In *Advances in Neural Information Processing Systems*, pp. 2414–2423, 2018.

Andrei N Kolmogorov. Three approaches to the quantitative definition of information. *Problems of information transmission*, 1(1):1–7, 1965.

Aviral Kumar, Rishabh Agarwal, Dibya Ghosh, and Sergey Levine. Implicit under-parameterization inhibits data-efficient deep reinforcement learning. *arXiv preprint arXiv:2010.14498*, 2020.

Jan Lemeire. Conditional independencies under the algorithmic independence of conditionals. *The Journal of Machine Learning Research*, 17(1):5252–5271, 2016.

Jan Lemeire and Dominik Janzing. Replacing causal faithfulness with algorithmic independence of conditionals. *Minds and Machines*, 23(2):227–249, 2013.

Ming Li, Paul Vitányi, et al. *An introduction to Kolmogorov complexity and its applications*, volume 3. Springer, 2008.

Barbara H Liskov. A design methodology for reliable software systems. In *Proceedings of the December 5-7, 1972, fall joint computer conference, part I*, pp. 191–199, 1972.

Vikash Kumar Mansinghka et al. *Natively probabilistic computation*. PhD thesis, Massachusetts Institute of Technology, Department of Brain and Cognitive . . . , 2009.

Christopher Meek. Strong completeness and faithfulness in bayesian networks. In *Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence*, UAI'95, pp. 411–418, San Francisco, CA, USA, 1995. Morgan Kaufmann Publishers Inc. ISBN 1558603859.

Marvin Minsky. Steps toward artificial intelligence. *Proceedings of the IRE*, 49(1):8–30, 1961.

Marvin Minsky. *Society of mind*. Simon and Schuster, 1988.

David L Parnas. On the criteria to be used in decomposing systems into modules. In *Pioneers and Their Contributions to Software Engineering*, pp. 479–498. Springer, 1972.

David Lorge Parnas. Information distribution aspects of design methodology. 1971.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *arXiv preprint arXiv:1912.01703*, 2019.

Deepak Pathak, Christopher Lu, Trevor Darrell, Phillip Isola, and Alexei A Efros. Learning to control self-assembling morphologies: a study of generalization via modularity. In *Advances in Neural Information Processing Systems*, pp. 2295–2305, 2019.

Judea Pearl. Causal diagrams for empirical research. *Biometrika*, 82(4):669–688, 1995.

Judea Pearl. *Causality*. Cambridge university press, 2009.

Jonas Peters, Dominik Janzing, and Bernhard Schölkopf. *Elements of causal inference*. The MIT Press, 2017.

Gavin A Rummery and Mahesan Niranjan. *On-line Q-learning using connectionist systems*, volume 37. University of Cambridge, Department of Engineering Cambridge, UK, 1994.

John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2015.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Ray J Solomonoff. A preliminary report on a general theory of inductive inference. United States Air Force, Office of Scientific Research, 1960.

Ray J Solomonoff. A formal theory of inductive inference. part ii. *Information and control*, 7(2): 224–254, 1964.

Peter Spirtes, Clark N Glymour, Richard Scheines, and David Heckerman. *Causation, prediction, and search*. MIT press, 2000.

Rupesh K Srivastava, Jonathan Masci, Sohrob Kazerounian, Faustino Gomez, and Jürgen Schmidhuber. Compete to compute. In *Advances in neural information processing systems*, pp. 2310–2318, 2013.

Gerald Jay Sussman. Building robust systems an essay. *Massachusetts Institute of Technology*, 2007.

Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2020.

Jan-Willem van de Meent, Brooks Paige, Hongseok Yang, and Frank Wood. An introduction to probabilistic programming. *arXiv preprint arXiv:1809.10756*, 2018.

Hado Van Hasselt, Yotam Doron, Florian Strub, Matteo Hessel, Nicolas Sonnerat, and Joseph Modayil. Deep reinforcement learning and the deadly triad. *arXiv preprint arXiv:1812.02648*, 2018.

William Vickrey. Counterspeculation, auctions, and competitive sealed tenders. *The Journal of finance*, 16(1):8–37, 1961.

Chihiro Watanabe. Interpreting layered neural networks via hierarchical modular representation. In *International Conference on Neural Information Processing*, pp. 376–388. Springer, 2019.

Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.

In this appendix, we provide related work, background, definitions, experiments, and additional exposition to supplement the main text.

# A BACKGROUND AND ASSUMPTIONS

We present background on algorithmic information theory and standard causality. For a more thorough treatment on the foundational mathematics and formalism, please refer to Li et al. (2008) for algorithmic information theory, to Pearl (2009) for standard causality, and to Janzing & Schölkopf (2010); Peters et al. (2017) for algorithmic causality. This section also states our assumptions for the results that we prove in §D. Lastly, we review the societal decision-making framework from Chang et al. (2020) that shows we can equivalently re-interpret a learnable discrete-action policy as composed of a society of learnable action-specific functions and a non-learnable selection mechanism.

## A.1 NOTATION

We denote with bolded uppercase monospace a computation graph at a single level of abstraction (e.g. the model of execution **G**, the model of credit assignment **C**). We denote with blackboard bold (e.g. the algorithmic causal model of learning $\mathbb{L}$) a computation graph that represents multiple levels of abstraction. We denote binary strings that represent the data nodes in $\mathbb{L}$ with lower case (e.g. $x$ or $\texttt{f}$), where script ($x$) is used to emphasize that the string typically represents data and monospace ($\texttt{f}$) is used to emphasize that the string typically represents a function. We use bolded lower case (e.g. $\boldsymbol{\tau}$, $\boldsymbol{\delta}$, $\mathbf{f}$) to indicate a group of binary strings. We denote the factor nodes and macro-operations in $\mathbb{L}$ (e.g., the transition function $\texttt{T}$ and reward function $\texttt{R}$ of **G**, the credit assignment mechanism $\Pi$, the transition function $\texttt{UPDATE}$ of **C**) with uppercase. We write $\texttt{f}(x) \rightarrow y$ to mean "a program $\texttt{f}$ that takes a string $x$ as input and produces a string $y$ as output."

## A.2 BACKGROUND ON ALGORITHMIC CAUSALITY

The formalism of algorithmic causality derives from Janzing & Schölkopf (2010); Peters et al. (2017), which builds upon algorithmic statistics (Gács et al., 2001). Here we directly restate or paraphrase additional relevant definitions, postulates, and theorems from Janzing & Schölkopf (2010) and Gács et al. (2001).

### A.2.1 ALGORITHMIC INFORMATION THEORY

**Kolmogorov complexity** Kolmogorov complexity (Solomonoff, 1960; 1964; Kolmogorov, 1965; Chaitin, 1966; 1975; Li et al., 2008) is a function $K : \{0,1\}^* \rightarrow \mathbb{N}$ from the binary strings $\{0,1\}^*$ to the natural numbers $\mathbb{N}$ that represents the amount of information contained in an object (represented by a binary string).

**Definition 4** (**Kolmogorov-complexity**). *Given a universal Turing machine and universal programming language as reference, the **Kolmogorov complexity** $K(s)$ is the length of the shortest program that generates $s$. The **conditional Kolmogorov complexity** $K(y \mid x)$ of a string $y$ given another string $x$ is the length of the shortest program that generates $y$ given $x$ as input. Let the shortest program for string $x$ be denoted as $x^*$. The **joint Kolomogorov complexity** $K(x,y)$ is defined as:*

$$K(x,y) \stackrel{+}{=} K(x) + K(x \mid y^*) \stackrel{+}{=} K(y) + K(y \mid x^*).$$

The **invariance theorem** (Kolmogorov, 1965) states that the Kolmogorov complexities of two strings written in two different universal languages differ only up to an additive constant. Therefore, we can assume any reference universal language for defining $K$ (e.g. Python) and work with equalities ($\stackrel{+}{=}$) and inequalities ($\stackrel{+}{\geq}$, $\stackrel{+}{\leq}$) up to an additive constant.

**Algorithmic mutual information** Algorithmic mutual information $I$ measures the amount of information two objects have in common:

**Definition 5** (**algorithmic mutual information**). *The **algorithmic mutual information** of two binary strings $x, y$ is*

$$I(x:y) \stackrel{+}{=} K(x) + K(y) - K(x,y).$$

*The **conditional algorithmic mutual information** of strings $x, y$ given string $z$ is*

$$I\left(x:y \mid z\right) \stackrel{+}{=} K\left(x \mid z\right) + K\left(y \mid z\right) - K\left(x, y \mid z\right)$$

We can intuitively think of $I(x : y \mid z)$ as, given $z$, the number of bits that can be saved when describing $y$ knowing the shortest program that generated $x$. Then algorithmic independence is the property of two strings that says that the description of one cannot be further compressed given knowledge of the other.

**Definition 6** (**algorithmic conditional independence**). *Given three strings $x, y, z$, $x$ is **algorithmically conditionally independent** of $y$ given $z$, denoted by $x \perp\!\!\!\perp y \mid z$, if the additional knowledge of $y$ does not allow for stronger compression of $x$, given $z$. That is:*

$$x \perp\!\!\!\perp y \mid z \;\Leftrightarrow\; I\left(x:y \mid z\right) \stackrel{+}{=} 0.$$

We further define the joint conditional independence of strings $x_1, ..., x_n$ given strings $y_1, ..., y_m$ analogously:

**Definition 7** (**algorithmic joint conditional independence**). *Strings $x_1, ..., x_n$ are **algorithmically jointly conditionally independent** given strings $y_1, ..., y_m$ if*

$$I\left(x_1, ..., x_n \mid y_1, ..., y_m\right) \stackrel{+}{=} 0, \tag{1}$$

*which means that*

$$K\left(x_1, ..., x_n \mid y_1, ..., y_m\right) \stackrel{+}{=} \sum_{i=1}^{n} K\left(x_i \mid y_1, ..., y_m\right), \tag{2}$$

meaning that, conditioned on knowing $y_1, ..., y_m$, the length of the joint description of $x_1, ..., x_n$ cannot be further compressed than sum of the lengths of the descriptions of the individual strings $x_i$. A proof of the equivalence between Eqs. 1 and 2 is given by the proof of Theorem 3 in Janzing & Schölkopf (2010).

We state as a lemma the following result from Gács et al. (2001, Corollary Π.8) that states the mutual information of strings $x$ and $y$ cannot be increased by separately processing by functions $f$ and $g$.

**Lemma 5** (**information non-increase**). *Let $f$ and $g$ be computable programs. Then*

$$I\left(\mathtt{f}(x) : \mathtt{g}(y)\right) \stackrel{+}{\leq} I\left(x:y\right) + K(\mathtt{f}) + K(\mathtt{g}). \tag{3}$$

This intuitively makes sense: if $K(\mathtt{f})$ is constant with respect to $x$ and $K(\mathtt{g})$ is constant with respect to $y$ (i.e. $K(\mathtt{f}) \stackrel{+}{=} 0$ and $K(\mathtt{g}) \stackrel{+}{=} 0$), then mutual information cannot increase between $x$ and $y$ separately with $\mathtt{f}$ and $\mathtt{g}$. In particular, if $x$ and $y$ were independent to begin with (i.e. $I(x:y) \stackrel{+}{=} 0$), then $I\left(\mathtt{f}(x) : \mathtt{g}(y)\right) \stackrel{+}{=} 0$.

The following lemma states that the mutual information between two strings is constant if we condition on one of the strings.

**Lemma 6** (**self-conditioning**). *For two strings $x$ and $y$,*

$$I(x:y \mid y) \stackrel{+}{=} 0.$$

*Proof.*

$$\begin{aligned}
I(x:y \mid y) &\stackrel{+}{=} K(x \mid y) + K(y \mid y) - K(x, y \mid y) \\
&\stackrel{+}{=} K(x \mid y) + K(y \mid y) - [K(x \mid y) + K(y \mid x^*, y)] \\
&\stackrel{+}{=} K(x \mid y) + 0 - [K(x \mid y) + 0] \\
&\stackrel{+}{=} K(x \mid y) - K(x \mid y) \\
&\stackrel{+}{=} 0.
\end{aligned}$$

□

**Terminology** In this paper, we regard the following statements about a program $\mathtt{f}(x) \to y$ as equivalent:

- "$K(\mathtt{f}) \stackrel{+}{=} 0$."
- "$\mathtt{f}$ is an $O(1)$-length program."
- "$K(\mathtt{f})$ is constant with respect to $x$."

Note that $K(\mathtt{f}) \stackrel{+}{=} 0$ implies that $\mathtt{f}$ and $x$ are algorithmically independent (i.e. $I(\mathtt{f} : x) \stackrel{+}{=} 0$) because

$$0 \stackrel{+}{\leq} I\left(x : \mathtt{f}\right) \stackrel{+}{=} K\left(\mathtt{f}\right) - K\left(\mathtt{f} \mid x^*\right) \stackrel{+}{\leq} K\left(\mathtt{f}\right) \stackrel{+}{=} 0.$$

### A.2.2 Causality

Before we review algorithmic causality, we first review some key concepts in standard causality: structural causal models and $d$-separation.

The following definition defines standard causal models over random variables as Bayesian networks represented as directed acyclic graphs (DAG).

**Definition 8** (**structural-causal-model**). *A **structural causal model** (SCM) (Pearl, 1995; 2009) represents the assignment of random variable $X$ as the output of a function, denoted by lowercase monospace (e.g. $\mathtt{f}$), that takes as input an independent noise variable $N_X$ and the random variables $\{PA_X\}$ that represent the parents of $X$ in a DAG:*

$$X := \mathtt{f}(\{PA_X\}, N_X). \tag{4}$$

*Given the noise distributions $\mathbb{P}(N_X)$ for all variables $X$ in the DAG, SCM entails a joint distribution $\mathbb{P}$ over all the variables in the DAG (Peters et al., 2017).*

The graph-theoretic concept of $d$-separation is used for determining conditional independencies induced by a directed acyclic graph (see point 3 in Thm. 9):

**Definition 9** (**$d$-separation**). *A path $p$ in a $DAG$ is said to be $d$-separated (or blocked) by a set of nodes $Z$ if and only if*

1. *$p$ contains a chain $i \to m \to j$ or fork $i \leftarrow m \to j$ such that the middle node $m$ is in $Z$, or*
2. *$p$ contains an inverted fork (or collider) $i \to m \leftarrow j$ such that the middle node $m$ is not in $Z$ and such that no descendant of $m$ is in $Z$.*

*A set of nodes $Z$ $d$-**separates** a set of nodes $X$ from a set of nodes $Y$ if and only if $Z$ blocks every (possibly undirected) path from a node in $X$ to a node in $Y$.*

### A.2.3 Algorithmic causality

We paraphrase Post. 6 and Thm. 4 from Janzing & Schölkopf (2010), which generalize structural causal models, or Bayesian networks (Pearl, 1995; 2009), to general programs, allowing us to use the language of causality to assess the independence of computable objects by inspecting the structure of the generative program that produced them.

**Definition 10** (**computational graph**). *Define a **computational graph** as a directed acyclic factor graph (DAG) of data nodes $x_1, ..., x_N$ and function nodes $\mathtt{f}_1, ..., \mathtt{f}_N$, constructed as follows. Let each $x_j$ be computed by a program $\mathtt{f}_j$ with length $O(1)$ from its parents $\{pa_j\}$ and possibly an additional noise input $n_j$. Assume the noise $n_j$ are jointly independent: $n_j \perp\!\!\!\perp \{n_{\neq j}\}$. Formally, $x_j := \mathtt{f}_j(\{pa_j\}, n_j)$, meaning that the Turing machine computes $x_j$ from the input $\{pa_j\}, n_j$ using the additional program $\mathtt{f}_j$ and halts.*

The computational graph represents a probabilistic program (van de Meent et al., 2018; Goodman et al., 2016; Mansinghka et al., 2009) in the general case, and would represent either a standard Bayesian network if every $\mathtt{f}_j$ takes in a noise variable or a deterministic program if none do. The **algorithmic causal Markov condition**, which states that $d$-separation implies conditional independence, generalizes the standard Markov condition to general programs:

**Theorem 7** (**algorithmic causal Markov condition**). *Let $\{pa_j\}$ and $\{nd_j\}$ respectively represent concatenation of the parents and non-descendants (except itself) of $x_j$ in a computational graph. Then $\forall x_j$, $x_j \perp\!\!\!\perp \{nd_j\} \mid \{pa_j\}$.*

It is typical to assume its converse, known as faithfulness:

**Postulate 8** (**faithfulness** (Spirtes et al., 2000)). *Given three sets $S$, $T$, $R$ of nodes in a computational graph, $I(S:T|R) \stackrel{+}{=} 0$ implies $R$ d-separates $S$ and $T$.*

The following theorem Janzing & Schölkopf (2010, Thm. 3) establishes the connection between the graph-theoretic concept of $d$-separation with condition algorithmic independence of the nodes of the graph.

**Theorem 9** (**equivalence of algorithmic Markov conditions**). *Given the strings $x_1, ..., x_n$ and a computational graph, the following conditions are equivalent:*

1. ***Recursive form:*** *the joint complexity is given by the sum of complexities of each node $x_j$, given the optimal compression of its parents $\{pa_j\}$:*

$$K(x_1, ..., x_n) \stackrel{+}{=} \sum_{j=1}^{n} K(x_j|\{pa_j\}^*).$$

2. ***Local Markov Condition:*** *Every node $x_j$ is independent of its non-descendants $\{nd_j\}$, given the optimal compression of its parents $\{pa_j\}$:*

$$I(x_j : nd_j|\{pa_j\}^*) \stackrel{+}{=} 0.$$

3. ***Global Markov Condition:*** *Given three sets $S$, $T$, $R$ of nodes*

$$I(S:T|R^*) \stackrel{+}{=} 0$$

*if $R$ d-separates $S$ and $T$.*

Together, Thm. 7, Post. 8, and Thm. 9 imply that data nodes in a computational graph are algorithmically independent if and only if they are $d$-separated in the computational graph.

### A.3 Assumptions

This paper analyzes learning algorithms from the perspective of algorithmic information theory, specifically algorithmic causality. To perform this analysis, we assume the following, and state our justifications for such assumptions:

1. The learning algorithm is implemented in on a universal Turing machine with a universal programming language.

   **Justification:** *This is a standard assumption in machine learning research that the machine learning algorithm can be implemented on a machine.*

2. Each initial parameter of the learnable functions **`f`** is jointly algorithmically independent of the other initial parameters.

   **Justification:** *This is a standard assumption in machine learning research that the noise from the random number generator is given background knowledge (Janzing & Schölkopf, 2010, §2.3), thus allowing us to ignore possible dependencies among the parameters induced by the random number generator in developing our algorithms.*

3. `UPDATE` is an $O(1)$-length program.

   **Justification:** *This is a standard assumption in machine learning research that the source code that computes the update rule (e.g. a gradient descent step) is agnostic to the feedback signals (e.g. gradients) it takes as input.*

4. $\mathbb{L}$ is faithful. That is, any conditional independence among the data nodes (`f`, $x$, $\delta$, and the internal data nodes of `APPLY` and $\Pi$) in $\mathbb{L}$ is due to the causal structure of $\mathbb{L}$ rather than a non-generic setting of these data nodes.

**Justification:** *Faithfulness has been justified for standard causal models (Meek, 1995). Deriving an algorithmic analog has been the subject of ongoing work (Lemeire & Janzing, 2013; Lemeire, 2016). For our work, a violation of faithfulness means that two nodes $x$, $y$ in the computational graph have $I(x : y) \stackrel{+}{=} 0$ when they are not $d$-separated in the computational graph. This would happen if $x$ and $y$ were tuned in such a way that makes one compressible given the other. Given assumption (2) above, the source of a violation of faithfulness must be the data experienced by the learning algorithm. Indeed, the data could be such that after learning certain parameters within* **f** *may be conditionally independent given the training history, as suggested by Csordás et al. (2020); Filan et al. (2020); Watanabe (2019). However, as our focus is on theoretical results that hold regardless of the data distribution the learning algorithm is trained on, we consider the specific instances where the data does induce such faithfulness violations as "non-generic" and thus out of scope of the paper.*

### A.4 Societal Decision-Making

An MDP is the standard computational graph for a sequential decision problem with $N$ discrete actions, defined with states $s \in \Omega_S$, actions $a \in \{1, ..., N\}$, transition function $\mathtt{T} : \Omega_S \times \{1, ..., N\} \to \Omega_S$, reward function $\mathtt{R} : \Omega_S \times \{1, ..., N\} \to \mathbb{R}$, and discount factor $\gamma$. We define a **decision** as a value of $a$. The MDP objective is to maximize the return $\sum_{t=0}^{T} \gamma^t \mathtt{R}(s_t, a_t)$ with a policy $\pi : \Omega_S \to \{1, ..., N\}$. However, representing the policy output as a single action $a$ is not granular enough for distinguishing the feedback into possible values of $a$.

The societal decision-making (SDM) framework (Chang et al., 2020) offers a more fine-grained alternative computational graph to the MDP by interpreting a discrete-action policy as a society of $N$ agents $\alpha^n$ that each controls a different possible action value. Each agent is a tuple $\alpha^n = (\psi^n, \phi^n)$ of a bidder $\psi^n : \Omega_S \to \Omega_B$ and a fixed **transformation mechanism** $\phi^n : \Omega_S \to \Omega_S$. SDM decomposes the policy $\pi$ of the standard MDP into a composition of two operations: one that computes the bids $b_s^n := \psi^n(s)$ for all bidders $n$, and one that applies a **selection mechanism** $\mathtt{S} : \Omega_B^N \to \{1, ..., N\}$ on the bids to select the index of the agent whose transformation mechanism is the move from $s$ to $s'$ that the action $a$ represents. SDM thus refactors (Curry et al., 1958) the transition and reward functions as $\mathtt{T} : \{1, ..., N\} \to [\Omega_S \to \Omega_S]$ and $\mathtt{R} : \{1, ..., N\} \to [\Omega_S \to \mathbb{R}]$.

Chang et al. (2020) introduced the cloned Vickrey society (CVS) RL algorithm as an on-policy single-step temporal-difference action-value method. CVS interprets the Bellman optimality equation as an economic transaction between agents seeking to optimize their utilites in a Vickrey auction (Vickrey, 1961) at each time-step. The Vickrey auction is the selection mechanism that selects the highest bidding agent $i$, which receives a utility

$$\underbrace{U_{s_t}^i(\alpha^{1:N})}_{\text{utility}} = \underbrace{\mathtt{R}^\phi(\alpha^i, s_t) + \gamma \cdot \max_k b_{s_{t+1}}^k}_{\text{revenue, or valuation } v_{s_t}} - \underbrace{\max_{j \neq i} b_{s_t}^j}_{\text{price}}, \tag{5}$$

and the rest receive a utility of $0$. In CVS each agent bids twice via an arbitrary function: the highest and second highest bids are produced by the same function parameters. The auction incentivizes each agent to truthfully bid the $Q$-value of its associated transformation mechanism, independent of the identities and bidding strategies of other agents.

## B From Monolithic Policies to Decision Mechanisms

Defining the model of execution **G** for RL requires us to define $\mathtt{f}$, $x^{\text{in}}$, $x^{\text{out}}$. If we were to use the standard MDP for **G**, we would represent a single learnable function $\mathtt{f}$ as the policy $\pi(s) \to a$, whose input $x^{\text{in}}$ is the current state $s_i$ and whose output $x^{\text{out}}$ is a tuple $(a_i, s_{i+1}, r_i)$ of the action, next state, and reward. However, this is level of granularity restricts us from representing feedback signals to AV methods, which may target only a specific value the action may take rather than the entire action distribution.

We hence adopt SDM for **G**, which replaces the monolithic policy with a set of functions that control each a different value of the action variable. We generalize the "bidder" terminology from §A.4 to encompass standard AV and PG methods by defining the action-specific **decision mechanism** as a

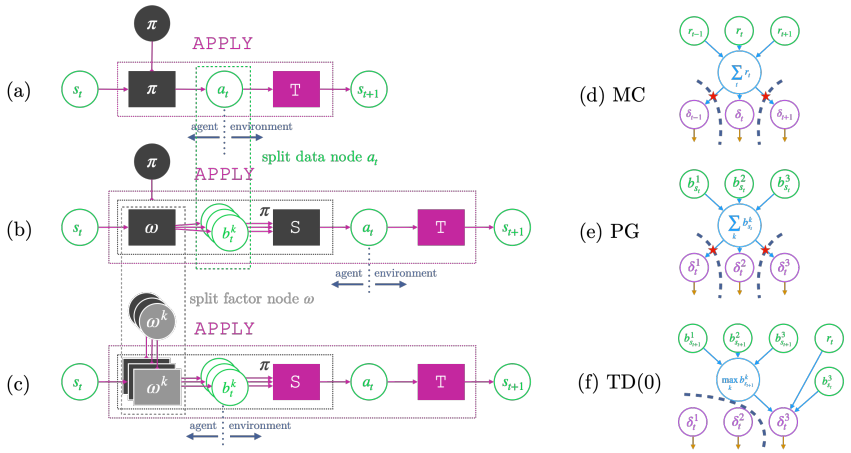


Figure 1: **Modularity in RL.** To convert the MDP model of execution (a) to the SDM model of execution (c), we split the action node into a set of nodes each representing a value the action can take (b) and split the monolithic policy into a set of decision mechanisms for each possible action (c). TD($n > 1$) methods like using Monte Carlo (MC) estimation (d) and policy gradient (PG) methods (e) do not have modular credit assignment mechanisms because they contain shared hidden variables. TD(0) methods (f) have modular credit assignment mechanisms in general. The causal edges of non-modular credit assignment cut a partitioning among the gradients $\delta$, indicated by the red star.

function that produces a bid $b_i^k$ for its corresponding action, which would correspond to an action-specific estimated $Q$-value $Q(\cdot, a)$ for AV methods or an action probability $p(a = k \mid \cdot)$ for policy gradient methods. The action is then selected from the bids via algorithmically independent selection mechanism S, such as a stochastic sampler for a PG method or an $\varepsilon$-greedy sampler for $Q$-learning. Observe that the agent-environment boundary separates the function we want to learn from other algorithmically independent functions, such as the transition function. Thus, we can then absorb S into the environment (Fig. 1a-c), thus splitting the formerly monolithic policy into a society of $K$ decision mechanisms as the learnable functions of **G**, each with input as $s_i$ and output as the tuple $(b_{s_i}^k, s_{i+1}, r_i, w_i)$, where $w_i$ is a binary flag that indicates whether the S chose its corresponding action. Define a **decision sequence** as the execution trace instantiated for RL with SDM.

## C Related Work

We synthesize perspectives from multi-agent reformulations of intelligence (Minsky, 1988; Chang et al., 2020; Balduzzi, 2014; Baum, 1996; Srivastava et al., 2013), computer programming (Liskov, 1972; Parnas, 1971; 1972), and causality (Hausman & Woodward, 1999; Pearl, 2009; Peters et al., 2017) around the theme of algorithmic independence (Janzing & Schölkopf, 2010; Kolmogorov, 1965; Li et al., 2008) to present a formalism for modularity in RL that reframes how RL is traditionally conceptualized (Sutton & Barto, 2020) in several ways. One conceptual shift is to treat the **decision-mechanism** (§B), rather than the policy, as the core primitive of decision-making, building directly off the societal decision-making framework (Chang et al., 2020) that refactors the traditional monolithic agent into a multi-agent system of mechanisms that each control a different decision. The practical benefit of a multi-agent reframing of a traditionally monolithically-framed problem enables the credit assignment (Minsky, 1961) into individual decisions to be decoupled from that into others, which motivates a second conceptual shift: that modularity is a property of not merely how the system itself is statically factorized – the view taken by most previous work on modularity in machine learning (Andreas et al., 2016; Devin et al., 2017; Goyal et al., 2019; Kirsch et al., 2018; Pathak et al., 2019; Alet et al., 2018; Chang et al., 2018; Csordás et al., 2020), but *how the outer process that changes the system is factorized as well*. In computer programming this outer process is the human reasoning process (Dijkstra, 1968); in machine learning it is the credit assignment mechanism. To analyze how the causal structure of the credit assignment mechanism interacts with the causal structure of the decision-maker that it modifies requires a third conceptual shift based on the duality of "code as data" (Abelson & Sussman, 1996) that embeds in the *same* causal graph both the

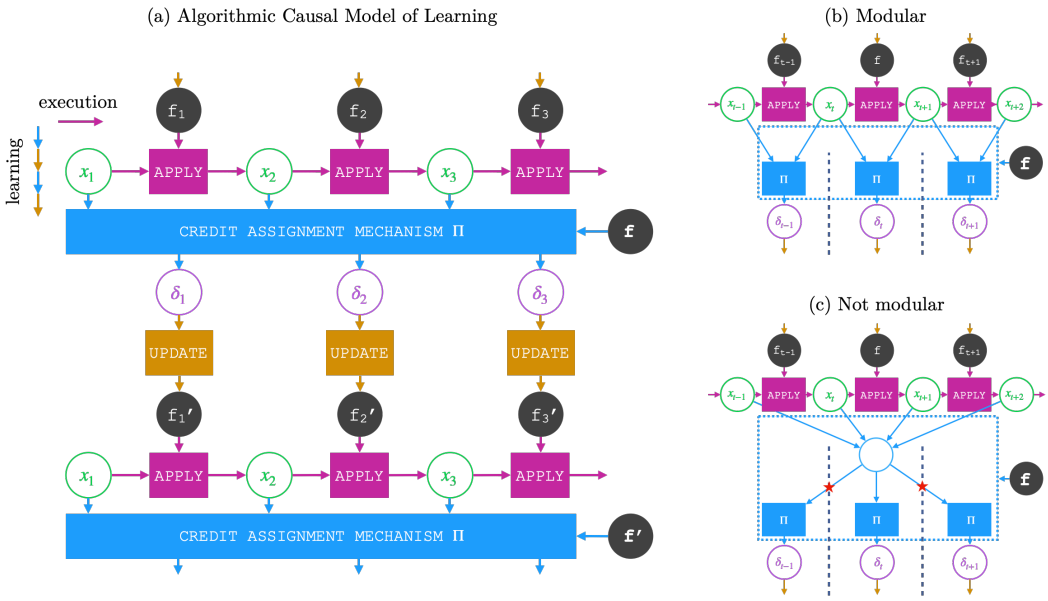


Figure 2: **Algorithmic Causal Model of Learning.** A learning algorithm with credit assignment mechanism $\Pi$ that produces gradients $\delta$ to update functions $\mathtt{f}$ to $\mathtt{f}'$ can be represented as a causal graph (a). $\Pi$ is not modular (b) if it contains a hidden variable whose outgoing causal edges cut a partitioning among the $\delta$'s (shown by the red star) and modular (c) if it does not.

decision mechanisms that transform states in the MDP and the credit assignment (meta-)mechanism that transforms the decision mechanisms themselves in the learning process.

## D PROOFS

Given the assumptions stated in §A.3, we now provide the proofs for our theoretical results. We will prove Lemma 3 first. Together with the faithfulness postulate (Post. 8) and the equivalence of algorithmic Markov conditions (Thm. 9) we can prove algorithmic independence by inspecting the graph of $\mathbb{L}$ for $d$-separation.

**Lemma 3 (algorithmic causal model of learning).** *Given a model of execution* $\boldsymbol{G}$ *and of credit assignment* $\boldsymbol{C}$*, define the* ***algorithmic causal model of learning*** *(ACL) as a computational graph* $\mathbb{L}$ *of the learning process with data nodes* $x$*,* $\mathtt{f}$*, and* $\delta$ *and factor nodes* $\mathtt{APPLY}$*,* $\Pi$*, and* $\mathtt{UPDATE}$*, as well as the internal data and factor nodes that constitute the structure of* $\mathtt{APPLY}$ *and* $\Pi$ *which vary by the learning algorithm. Then* $\mathbb{L}$ *is a double trellis over* $x$ *and* $\mathtt{f}$ *generated via an inner loop governed by* $\mathtt{APPLY}(\mathtt{f}, x^{in}_{\mathtt{f}}) \to x^{out}_{\mathtt{f}}$ *that generates the execution trace* $\boldsymbol{\tau}$ *and an outer loop structured as the CMP* $\boldsymbol{C}$ *governed by two algorithmically independent operations* $\Pi(\boldsymbol{\tau}, \boldsymbol{\mathtt{f}}) \to \boldsymbol{\delta}$ *and* $\mathtt{UPDATE}(\boldsymbol{\mathtt{f}}, \boldsymbol{\delta}) \to \boldsymbol{\mathtt{f}}'$*. Then the strings* $x^{in}_{\mathtt{f}^k_i}$*,* $x^{out}_{\mathtt{f}^k_i}$*,* $\mathtt{f}^k_i$*,* $\delta^k_i$*, and the internal data nodes of* $\mathtt{APPLY}$ *and* $\Pi$*, for all steps of credit assignment* $n$*, satisfy the algorithmic causal Markov condition with respect to* $\mathbb{L}$*.*

*Proof.* $\mathbb{L}$ is a well-defined computational graph and so by Thm. 7 it satisfies the algorithmic causal Markov condition. □

*Remark.* By Lemma 5, if a set of data nodes in $\mathbb{L}$ are independent, then processing them separately with factor nodes of $\mathbb{L}$ will maintain this independence. For example, given that the $\mathtt{UPDATE}$ operation is applied in separately for each pair $(\mathtt{f}^k, \sum_i \delta^k_i)$ to produce a corresponding $\mathtt{f}^{k\prime}$, then if $(\mathtt{f}^k, \sum_i \delta^k_i)$ were independent of $(\mathtt{f}^j, \sum_i \delta^j_i)$ before applying $\mathtt{UPDATE}$, then $\mathtt{f}^{k\prime}$ would be independent of $\mathtt{f}^{j\prime}$ after applying $\mathtt{UPDATE}$.

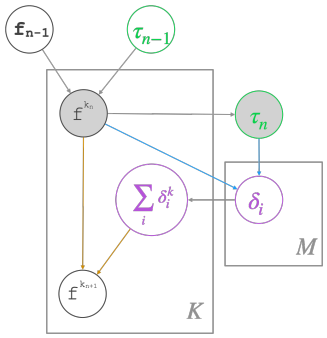


Figure 3: This figure shows the computation graph of $\mathbb{L}$ across one credit assignment update. A modular credit assignment mechanism (shown with blue edges) is equivalent to showing the gradients $\delta_i$ as conditionally independent, as shown by the plate notation labeled with $M$. Dynamic modularity at iteration $n-1$ is equivalent to showing that the functions $\mathtt{f}^{k_n}$ are inside the plate labeled with $K$. Then because the UPDATE operation, shown with yellow edges, operates only within the plate labeled with $K$, the updated functions $\mathtt{f}^{k_{n+1}}$ are also conditionally independent given $(\boldsymbol{\tau}, \mathbf{f})$.

**Theorem 1 (modular credit assignment).** *Dynamic modularity is enforced at learning iteration $n$ if and only if static modularity holds, i.e. $I\left(\mathtt{f}^{k_1} : \mathtt{f}^{j_1}\right) \stackrel{+}{=} 0$ for all $k \neq j$, and the CAM satisfies the modularity constraint.*

*Proof.* We will prove by induction on $n$. The inductive step will make use of the equivalence between $d$-separation and conditional independence.

**Base case:** $n = 1$. There is no training history, so static modularity is equivalent to dynamic modularity.

**Inductive hypothesis:** Assuming that dynamic modularity holds if and only if static modularity and modular credit assignment hold for learning iteration $n-1$, dynamic modularity holds if and only if static modularity and modular credit assignment hold for learning iteration $n$.

**Inductive step:** The modularity constraint states

$$I\left(\delta_1, ..., \delta_M \mid \boldsymbol{\tau}_n, \mathbf{f}_n\right) \stackrel{+}{=} 0.$$

Dynamic modularity at iteration $n-1$ states that

$$\forall k \neq j, \quad I\left(\mathtt{f}^{k_n} : \mathtt{f}^{j_n} \mid \boldsymbol{\tau}_{n-1}, \mathbf{f}_{n-1}\right) \stackrel{+}{=} 0.$$

These two above statements correspond to the computational graph in Fig. 3. Note that by Def. 7, disjoint subsets of $\delta_1, ..., \delta_M$ also have have zero mutual information up to an additive constant. Letting these subsets be $\sum_i \delta_i^k$ where $k$ is the index of function $\mathtt{f}^k$ in $\mathbf{f}$, then

$$I\left(\sum_i \delta_i^1, ..., \sum_i \delta_i^K \,\middle|\, \boldsymbol{\tau}_n, \mathbf{f}_n\right) \stackrel{+}{=} 0. \tag{6}$$

Then, as we can see by direct inspection in Fig. 3, $\mathtt{f}^{k_n}$ and $\mathtt{f}^{j_n}$ are $d$-separated by $(\boldsymbol{\tau}_n, \mathbf{f}_n)$, which is equivalent to saying that dynamic modularity holds for iteration $n$. □

**Theorem 4 (modularity criterion).** *If $\mathbb{L}$ is faithful, the modularity constraint holds if and only if for all $i$ the outputs $\delta_i$ and $\delta_{\neq i}$ of $\Pi$ are $d$-separated by its inputs $\boldsymbol{\tau}$ and $\mathbf{f}$.*

*Proof.* The forward direction holds by the equivalence of algorithmic causal Markov conditions (Thm. 9), and the backward direction holds by the faithfulness assumption. □

**Theorem 2 (modular factorization).** *The credit assignment mechanism $\Pi(\boldsymbol{\tau}, \mathbf{f}) \to \boldsymbol{\delta}$ is modular if and only if $K\left(\boldsymbol{\delta} \mid \boldsymbol{\tau}, \mathbf{f}\right) \stackrel{+}{=} \sum_{i=1}^{M} K\left(\delta_i \mid \boldsymbol{\tau}, \mathbf{f}\right)$.*

*Proof.* The proof comes from the definition of algorithmic mutual information.

$$K\left(\boldsymbol{\delta} \mid \boldsymbol{\tau}, \mathbf{f}\right) \stackrel{+}{=} \sum_{i=1}^{M} K\left(\delta_i \mid \boldsymbol{\tau}, \mathbf{f}\right) \tag{7}$$

$$\sum_{i=1}^{M} K\left(\delta_i \mid \boldsymbol{\tau}, \mathbf{f}\right) - K\left(\boldsymbol{\delta} \mid \boldsymbol{\tau}, \mathbf{f}\right) \stackrel{+}{=} 0 \tag{8}$$

$$I\left(\delta_1, ..., \delta_M \mid \boldsymbol{\tau}, \mathbf{f}\right) \stackrel{+}{=} 0 \tag{9}$$

□

**Corollary 4.1 (policy gradient).** *All PG methods do not satisfy MCn.*

*Proof.* It suffices to identify a single shared hidden variable that renders $\delta_1, ..., \delta_M$ not $d$-separated. Computing the policy gradient includes the log probability of the policy as one of its terms. Computing this log probability for any action involves the same normalization constant $\sum_k b^k$. This normalization constant is a hidden variable that renders $\delta_1, ..., \delta_M$ not $d$-separated, as shown in Fig. 4. □

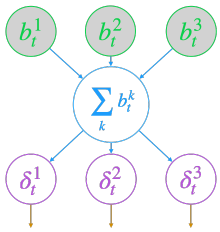


Figure 4: This figure shows part of the computational graph within $\Pi$ for policy gradient methods. Conditioning on $\boldsymbol{\tau}$ implies we condition on the lightly shaded nodes. $\sum_k b_t^k$ is the shared hidden variable that renders $\delta_1, ..., \delta_M$ not $d$-separated.

**Corollary 4.2 (TD$(n > 1)$).** *All TD($n > 1$) methods do not satisfy MCn.*

*Proof.* It suffices to identify a single shared hidden variable that renders $\delta_1, ..., \delta_M$ not $d$-separated. TD($n > 1$) methods include a sum of estimated returns or advantages at different steps of the decision sequence that is shared among multiple $\delta_i$'s. This sum is the hidden variable that renders $\delta_1, ..., \delta_M$ not $d$-separated, as shown in Fig. 5. □

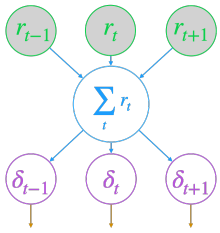


Figure 5: This figure shows part of the computational graph within $\Pi$ for TD$(n > 1)$ methods. Conditioning on $\boldsymbol{\tau}$ implies we condition on the lightly shaded nodes. $\sum_t r_t$ is the shared hidden variable that renders $\delta_1, ..., \delta_M$ not $d$-separated.

**Corollary 4.3 (TD$(0)$).** *TD(0) methods satisfy MCn for acyclic $\boldsymbol{\tau}$.*

*Proof.* It suffices to identify a single shared hidden variable that renders $\delta_1, ..., \delta_M$ not $d$-separated. TD(0) methods produce gradients as $\delta_i^k := g(b_i^k, s_i, s_{i+1}, r_i, \mathbf{f})$ if $w_i = 1$ (the decision mechanism

was selected) and $\delta_i^k := 0$ otherwise, for some function $g$. The only hidden variable is $[\max_j b^j_{s_{i+1}}]$, and for acyclic $\boldsymbol{\tau}$ there is only one state $s_i$ in $\boldsymbol{\tau}$ that transitions into $s_{i+1}$. Therefore the hidden variable is unique to each of $\delta_1, ..., \delta_M$, so $\delta_1, ..., \delta_M$ remain $d$-separated, as shown in Fig. 6. □

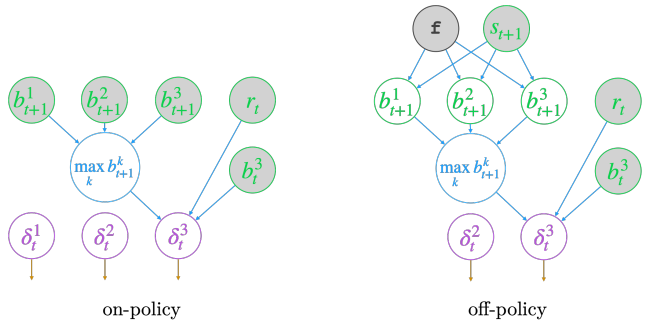


Figure 6: This figure shows part of the computational graph within $\Pi$ for on-policy and off-policy TD(0) methods. Conditioning on $(\boldsymbol{\tau}, \mathbf{f})$ implies we condition on the lightly shaded nodes. For on-policy methods such as CVS and SARSA, the hidden variable would be $\max_k b^k_{t+1}$ for CVS and the bid corresponding to the decision mechanism that was sampled through $\varepsilon$-greedy for SARSA. The figure shows $\max_k b^k_{t+1}$ for concreteness. For off-policy methods such as $Q$-learning, the bids $b_{t+1}$ are computed from $s_{t+1}$ and $\mathbf{f}$, both of which we condition on. In both cases, the hidden variable is only parent to one of the $\delta_i$'s, and thus the $\delta_1, ..., \delta_M$ remain $d$-separated.

**Corollary 1.1 (tabular).** *Thm. 1 holds for $Q$-learning, SARSA, CVS in the tabular setting.*

*Proof.* In the tabular setting, decision mechanisms are columns of the $Q$-table corresponding to each action. These columns do not share parameters, so static modularity holds. Then because $Q$-learning, SARSA, and CVS are TD(0) methods, by Corollary 4.3, their credit assignment mechanisms are modular. Therefore Thm. 1 holds. □

**Corollary 1.2 (function approximation).** *In the general function approximation setting, Thm. 1 holds for CVS.*

*Proof.* The decision mechanisms of CVS do not share weights, so static modularity holds. By Corollary 4.3 its credit assignment mechanism is modular. Therefore Thm. 1 holds. □

## E Numerical Simulations

We have defined dynamic modularity and shown what properties the RL algorithm needs to achieve it, but how dynamic modularity correlates with transfer efficiency depends on the task and is thus an empirical question. Our overarching question is: if the transfer problem requires re-using previously optimal decisions, then how does having a dynamically modular method affect transfer efficiency? There is inevitably a gap between any theoretical model and its empirical implementation, so we cannot ever claim definitive empirical validation of our theories. However, viewing our empirical results from the lens of ACL provides a useful framework for generating explanations and future hypotheses for understanding the differences in transfer behavior.

Our analysis of AV and PG methods focuses primarily on on-policy methods CVS and PPO because there are still many factors that influence the learning of off-policy methods that are still not well understood (Achiam et al., 2019; Kumar et al., 2020; Van Hasselt et al., 2018; Fu et al., 2019). We also compare with statically modular version of PPO, abbrv. PPOF, whose decision mechanisms do not share weights.

We consider transfer problems where only one decision in a previously optimal decision sequence needs to be changed. States are represented as binary vectors. The reward is only 1 at the end of the sequence if the task is solved otherwise it is 0. The relationship between the training and transfer

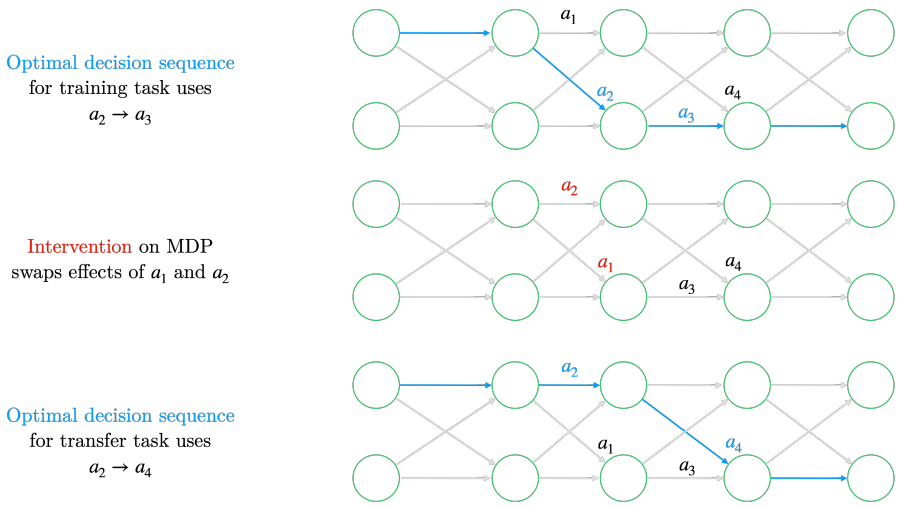


Figure 7: **Intervention on the transition function.** We consider transfer problems where the optimal decision sequence of the transfer task differs from that of the training task by a single decision. As above, the transfer MDP and the training MDP differ in that the effects of actions $a_1$ and $a_2$ get swapped; all other transitions remain the same. The agent must learn to choose action $a_4$ instead of $a_3$ while re-using other previously optimal decisions.

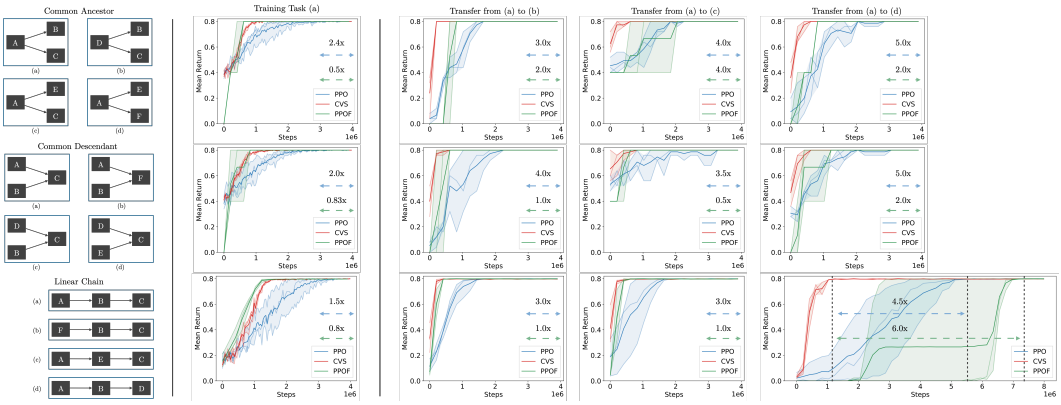


Figure 8: **Exhaustive enumeration of transfer problems.** For each task topology (leftmost column) we have a training task, labeled (a) and three independent transfer tasks, labeled (b,c,d). Each transfer task represents a different way to modify the MDP used for training. With few exceptions, CVS most consistently exhibits higher sample efficiency than both PPO and PPOF, a version of PPO whose decision mechanisms do not share weights. The consistent improvement of transfer efficiency of PPOF over PPO suggests that static modularity is helpful for this kind of transfer problem, and the generally consistent improvement of CVS over PPOF suggests that dynamic modularity, which involves modular credit assignment, is even better. Notably the gap between CVS and the other methods is so wide that we had to extend the chart width. We set the convergence time as the first time after which the return deviates by no more than $\varepsilon = 0.01$ from the optimal return, 0.8, for 30 epochs of training. Shown are runs across three seeds.

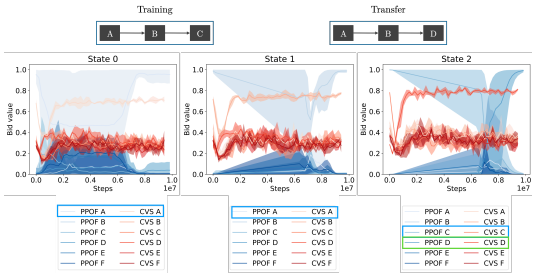


Figure 9: **How the decision mechanisms change during transfer.** Shown are the three states of the decision sequence. The optimal last decision must change from action C (blue) to action D (green). CVS modifies its bids independently. The bids for PPOF are coupled together across decision mechanisms and across time.

MDP is given by an intervention in the MDP transition function, concretely shown in Fig. 7. In the same way that analysis of $d$-separation is conducted with triplets of nodes, we exhaustively enumerated all possible topologies of triplets of decisions: linear chain, common ancestor, and common descendant (Fig. 8, left). For each topology we exhaustively enumerated all ways of making an isolated change to an optimal decision sequence. The common ancestor and common descendant topologies involve multi-task training for two decision sequences of length two, while linear chain involves single-task training for a decision sequence of length three. For each topology we have a training task and three independent transfer tasks that each start with the learner from the training task, where each transfer task represents a different way to modify the MDP used for training. This exhaustive enumeration enables us to ask a wide range of questions from a single comprehensive plot (Fig. 8).

**How does static modularity affect transfer efficiency?** We compare PPO and PPOF, which are trained exactly the same, except that the decision mechanisms of PPOF do not share weights while those of PPO do. The consistent improvement of PPOF over PPO in transfer, often to a greater degree than during training (with the exception of the bottom right of Fig. 8), suggests that decoupling decision mechanisms is itself already useful.

**Dynamic modularity vs static modularity.** We compare CVS and PPOF. While PPOF largely transfers almost as efficiently as CVS for other transfer problems, the interesting comparison is the bottom right of Fig. 8, where PPOF transfers about six times slower. This is the transfer problem where the last decision must be changed between training and transfer. At the beginning of transfer, the (suboptimal) reward propagates immediately to all the decision mechanisms involved at all timesteps for PPOF during a single credit assignment update, whereas it only propagates back one step for CVS. The effect of this can be seen in how CVS and PPOF modify their decision mechanisms over time in Fig. 9, where we observe that bids of CVS move independently whereas a change in one bid in PPOF is accompanied by a corresponding change in other bids, even those that do not need to be changed. This coupling of gradient signal across both time and decision mechanisms could partially explain why CVS recovers much faster than PPOF.

**Bandit** The following experiment specifically targets the transfer differences between CVS, which is dynamically modular, and PPOF, which is statically modular but not dynamically modular, in a way that studies in isolation the effect of independent gradients into different decision mechanisms on transfer efficiency. Both are on-policy algorithms, so the main difference between the two is the gradients in CVS are independent while the gradients in PPOF are not. We consider a single-step MDP, or a bandit, with four actions: $A$, $B$, $C$, $D$. During training, taking action $A$ gives a reward of 0.8, action $B$ a reward of 0.6, action $C$ a reward of 0.4, and action $D$ a reward of 0.2. During transfer the rewards for action $A$ and $D$ are swapped. Thus to transfer requires only to swap the bidding behavior of the decision mechanism for action $A$ and the decision mechanism for action $D$. This is the simplest transfer task to compare static and dynamic modularity because it removes the temporally extended nature of MDPs as a source of dependency in the gradients. As we can see from Fig. 10, CVS transfers about 3.9 times faster than PPOF even though they have a similar sample efficiency during the training task. This result is significant because it shows that even when the transfer task is very simple and the decision mechanisms do not share weights, independence in gradients makes a big difference in transfer efficiency, thereby providing support for our overarching

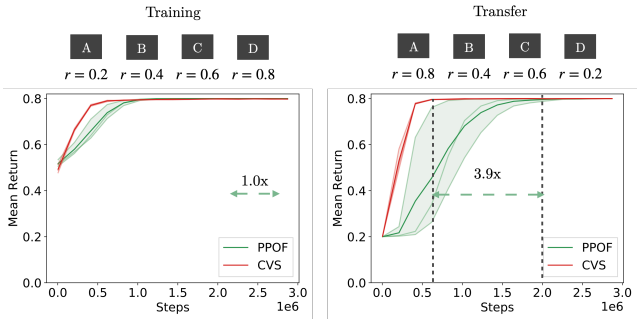


Figure 10: **Bandit**. This task removes the temporal dimension of the MDP, allowing us to focus solely on the difference in transfer efficiency between a method that is dynamically modular (CVS) and one that is only statically modular (PPOF). CVS transfers about 3.9 times faster than PPOF even though they have a similar sample efficiency during the training task.

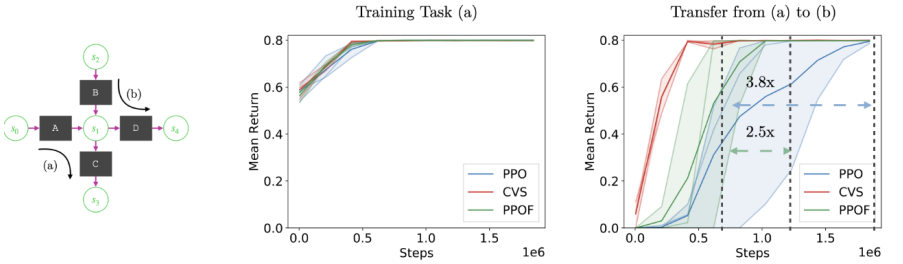


Figure 11: **Disjoint**. By transferring on a novel task (b) whose optimal action sequence is completely distinct from the pretraining task (a), we can focus on the impact of previously learned behaviors on learning new ones. The modular method CVS isolates the credit assignment across action weights(spatially) and gradients(temporally), shielding the non optimal actions from task (a). This allows CVS to be 2.5 times more sample efficient than PPOF and 3.8 times more sample efficient than PPO.

hypothesis that modular credit assignment plays a major role in realizing the extensibility benefits of modularity in machine learning.

**Disjoint** We next test the performance of these methods on transfer to a novel task that shares no components of with the pretraining task. More specifically, we begin by training on task (a) in Fig. 11 which has an optimal action sequence A-C with a reward of 0.8. Once converged, we transfer the model to task (b) where the optimal action sequence is now B-D, also with a reward of 0.8. This allows us to investigate any interference of the previously learned behaviour in how the new task is learned. We see that while all three methods converge within the same number of steps in the environment for Training Task (a), PPO takes 3.8 times more training steps than CVS to converge in Transfer Task (b). Similarly, PPOF is 2.6 times slower than CVS to converge for this transfer task. We expect a modular method to be able to transfer efficiently when the optimal solution is disjoint since previously learned task's optimal behavior is isolated to the corresponding set of agents. Whereas the significantly slower performance of the policy gradient methods in comparison to CVS suggests that the dependencies among the gradients of PPO and PPOF and also those in the shared weights for PPO play a role in making them transfer slower. We also note that both Training and Transfer tasks (a) and (b) are of the same difficulty and thus the greater sample efficiency of CVS in task (b) cannot be attributed to CVS being innately more sample efficient on any task as all methods are equally sample efficient for task (a).

**In conclusion, how does dynamic modularity affect transfer efficiency?** Fig. 9 shows that CVS is able to modify different decisions independently, while this modification is coupled for PPOF, which does not have modular credit assignment. Fig. 8 shows CVS has the highest transfer efficiency most consistently. PPOF is less consistent, possibly due to its lack of modular credit

assignment, but its generally better performance than PPO suggests that modularity in some form plays a key role in efficient transfer.

# F SIMULATION DETAILS

We implemented our simulations using the PyTorch library (Paszke et al., 2019).

## F.1 IMPLEMENTATION DETAILS

The underlying PPO (Schulman et al., 2017) implementation used for CVS, PPO, and PPOF used a policy learning rate of $4 \times 10^{-5}$, a value function learning rate of $5 \times 10^{-3}$, a clipping ratio of $0.2$, a GAE (Schulman et al., 2015) parameter of $0.95$, a discount factor of $0.99$, entropy coefficient of $0.1$, and the Adam (Kingma & Ba, 2014) optimizer. For all algorithms, the policy and value functions for our algorithms were implemented as neural network function approximators that used one hidden layer with rectified linear units. For all algorithms, performed a PPO update every 4096 samples with a minibatch size of 256. These hyperparameters were chosen from

## F.2 TRAINING DETAILS

All learning curves are plotted from three random seeds, with a different learning algorithm represented by a different hue. The dark line represents the mean over the seeds. The lighter lines represent the curves for individual seeds.

Our protocol for transfer is as follows. A transfer problem is defined by a (training, transfer) task pair, where the initial network parameters for the transfer task are the network parameters learned the training task for $T$ samples. In our simulations, we set $T$ to $10^7$ because that was about double the number of samples for all algorithms to visually converge on the training task for all seeds. To calculate the relative sample efficiency of CVS over PPO and PPOF (e.g. 4.5x and 6.0x respectively in the bottom-up right corner of Fig. 8), we set the criterion of convergence as the number of samples after which the return deviates by no more than $\varepsilon = 0.01$ from the optimal return for 30 epochs of training, where each epoch of training trains on 4096 samples.

## F.3 ENVIRONMENT DETAILS

The environment for our experiments shown in Fig. 8 represented as discrete-state, discrete-action MDPs. Each state is represented by a binary-valued vector.

The structure of the MDP can best be explained via an analogy to a room navigation task, which we will explain in the context of the $A \to B \to C$ task in the Linear Chain topology. In this task, there are four rooms, room 0, room 1, room 2, and room 3. Room 0 has two doors, labeled $A$ and $F$, that lead to room 1. Room 1 has two doors, labeled $B$ and $E$, that lead to room 2. Room 2 has two doors, labeled $C$ and $D$. Doors are unlocked by keys. The state representation is a concatenation of two one-hot vectors. The first one-hot vector is of length four; the "1" indicates the room id. The second one-hot vector is of length six; the "1" indicates the presence of a key for door $A$, $B$, $C$, $D$, $E$, or $F$. Only one key is present in a room at any given time. If the agent goes through the door corresponding to the key present in the room, then the agent transitions into the next room; otherwise the agent stays in the same room. In the last room, if the agent opens the door corresponding to the key that is present in the room, then the agent receives a reward of $1$. All other actions in every other state receive a reward of $0$. Therefore the agent only gets a positive reward if it opens the correct sequence of doors. For all of our experiments, the optimal policy is acyclic, but a suboptimal decision sequence could contain cycles

Therefore, for the training task in the Linear Chain topology where the optimal solution is $A \to B \to C$, the possible states are

```
[1, 0, 0, 0 ; 1, 0, 0, 0, 0, 0]  # room 0 with key for A
[0, 1, 0, 0 ; 0, 1, 0, 0, 0, 0]  # room 1 with key for B
[0, 0, 1, 0 ; 0, 0, 1, 0, 0, 0]  # room 2 with key for C.
```

For the transfer task whose optimal solution is $A \to B \to D$, the possible states are

```
[1, 0, 0, 0 ; 1, 0, 0, 0, 0, 0]  # room 0 with key for A
[0, 1, 0, 0 ; 0, 1, 0, 0, 0, 0]  # room 1 with key for B
[0, 0, 1, 0 ; 0, 0, 0, 1, 0, 0]  # room 2 with key for D.
```

Whereas for the Linear Chain topology the length of the optimal solution is three actions, for the Common Ancestor and Common Descendant topologies this length is two actions. Common Ancestor and Common Descendant are multi-task problems. As a concrete example, in the training task for Common Ancestor is a mixture of two tasks, one whose optimal solution is $A \rightarrow B$ and one whose optimal solution is $A \rightarrow C$. Following the analogy to room navigation, this task is set up such that after having gone through door $A$, half the time there is a key to open door $B$ and half there is a key to open door $C$.

### F.4 Computing Details

The computing infrastructure was an AWS c5d.18xlarge instance. The average runtime training on $10^7$ samples was three hours for PPO and PPOF and 6 hours for CVS.