PACE: Procedural Abstractions for Communicating Efficiently

Jonathan D. Thomas
Chalmers University of Technology
Gothenburg, Sweden
jonathan.thomas@chalmers.se

Andrea Silvi
Chalmers University of Technology
Gothenburg, Sweden

Devdatt Dubhashi
Chalmers University of Technology
Gothenburg, Sweden

Vikar Garg
YaiYai Ltd and Aalto University
Espoo, Finland

Moa Johansson
Chalmers University of Technology
Gothenburg, Sweden

Abstract

A central but unresolved aspect of problem-solving in AI is the capability to introduce and use abstractions, something humans excel at. Work in cognitive science has demonstrated that humans tend towards higher levels of abstraction when engaged in collaborative task-oriented communication, enabling gradually shorter and more information-efficient utterances. Several computational methods have attempted to replicate this phenomenon, but all make unrealistic simplifying assumptions about how abstractions are introduced and learned. Our method, Procedural Abstractions for Communicating Efficiently (PACE), overcomes these limitations through a neuro-symbolic approach. On the symbolic side, we draw on work from library learning for proposing abstractions. We combine this with neural methods for communication and reinforcement learning, via a novel use of bandit algorithms for controlling the exploration and exploitation trade-off in introducing new abstractions. PACE exhibits similar tendencies to humans on a collaborative construction task from the cognitive science literature, where one agent (the architect) instructs the other (the builder) to reconstruct a scene of block-buildings. PACE results in the emergence of an efficient language as a by-product of collaborative communication. Beyond providing mechanistic insights into human communication, our work serves as a first step to providing conversational agents with the ability for human-like communicative abstractions.

1 Introduction

When presented with the opportunity for repeated interaction, human conversational partners tend towards more concise utterances [13, 24, 20]. This is done via the introduction of more informative abstractions into their language, which enable shorter utterances and thereby promote more efficient communication and cooperation. Additionally, humans use abstractions in many other procedural tasks, such as cooking and programming. In a seminal paper [14] that combines perspectives from Cognitive Science and AI it is argued that abstractions guide learning, facilitate trade-offs, and simplify computation. They identified an outstanding open problem that *"future work in both computer science and psychology will need to identify other pressures that can shape abstraction*

38th Conference on Neural Information Processing Systems (NeurIPS 2024).

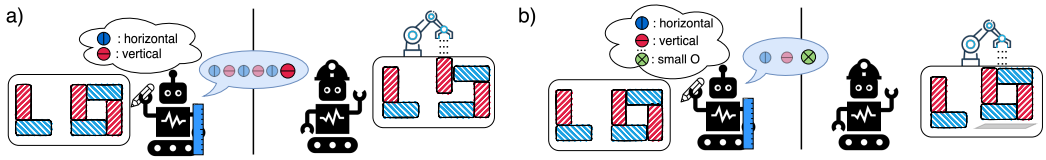


Figure 1: In early rounds of the architect-builder game, the architect messages refer to horizontal or vertical blocks (a). After multiple interactions, the architect tries to introduce an abstraction (the small O), which after a learning period allows for shorter communication to solve the task (b).

learning – such as the need to communicate and coordinate with others". We carry out the first steps to solving this problem in the setting of co-operative artificial agents. How could AI agents be designed to introduce and exploit abstractions in a similar way in co-operative problem-solving?

We take a novel *neuro-symbolic* approach to study the introduction of abstractions in the context of *emergent communication* (EC) in multi-agent systems [22, 9]. We investigate this as a *signalling game*, a foundational tool in cognitive science [23, 27] to establish how human languages are optimised to communicate concepts efficiently. Our setting is an extension of the Architect-Builder game [24], a two-agents collaborative building task. Here, the architect observes a goal-scene and instructs the builder to place blocks to recreate it. Using abstractions here can significantly increase the efficiency of the collaboration, as observed in human interactions [24] and depicted in Figure 1.

We draw on ideas from library learning [8, 1] to introduce new abstractions into the language. These correspond to common sequences of instructions that maximally reduce the length of future programs. However, it is not desirable to only naïvely consider compression, as this would result in a huge language with a word for every single shape – even those rarely seen. For humans, there is a trade-off between competing pressures: to be informative while not having to memorise too many words causing a high cognitive load [32]. To address this trade-off in our agents, we use techniques from EC and reinforcement learning previously shown to promote these pressures in other settings [4, 3].

We test our model within an extended version of the Architect-Builder game and find that it exhibits the same tendency as humans — a development toward more informative and concise communication. Interestingly, we find that after a number of abstractions have been introduced, our model naturally converges to a stable language, situating this observation within the cognitive science literature on efficient communication [17, 10, 33, 11]. Our approach addresses many limitations of existing approaches by [24] and [16] and provides a valuable framework for future exploration in this area.

2 Architect-Builder Game

In the Architect-Builder Game, the architect is provided with a set of goal-scenes consisting of horizontal and vertical blocks on a (9x9) grid. The architect needs to communicate instructions to the builder (who does not know the goal state) that allow for it to construct the goal-scene starting from a blank grid. This is inspired by the human experiments in [24]. In our work, the architect is a neuro-symbolic agent and the builder is a neural agent. The architect is initialised with a pre-defined symbolic policy π_{arch}, which for each goal-scene contains a *program* to construct it. These programs consist of a sequence of instructions in a language $\mathcal{A}$, initially containing two primitive actions: placing either a horizontal or vertical block. The agents are trained together to learn policies that enable the architect to communicate instructions to the builder that enable it to reach the goal-state. We refer to the architect and builder's communication policies as π_{comm} and π_{bldr}, respectively.

3 Abstractions for Communicating Efficiently

There are three main components to PACE, **Abstraction**, **Communication** and **Exploration/Exploitation**. In the Abstraction component, the architect utilises a library learning mechanism to identify common sub-sequences for abstraction. In the Communication component, EC allows the architect and builder to learn a language for collaboration. The Exploration/Exploitation component utilises bandit techniques to choose between alternative programs for a given goal-scene.

At a high-level interaction between the architect and builder in PACE proceeds as follows: (1) given a goal-scene the architect chooses a program via the bandit, and (2) the architect and builder communicate via EC. After multiple interactions, an abstraction is introduced through library learning. Then the loop repeats. Appendix A.2 presents more detailed algorithms explaining each phase.

3.1 Introducing Abstractions

The architect initially has a program consisting of primitive instructions to create a given goal-scene from an empty grid. As the interaction proceeds with the builder, the architect may discover common abstractions which can shorten the communication. These can be used to rewrite current programs allowing for more compressed versions which are then available as an alternative in later interactions. The architect can analyse the available programs and introduce into $\mathcal{A}$ a new abstraction for a sub-sequence that enables the maximal reduction in the length of the whole set of programs. We choose a sub-routine according to a Bayesian procedure inspired by library learning methods like DreamCoder [8]. The architect evaluates a set of sub-sequences and picks the one which maximises (1):

$$P(\mathcal{A} \cup \{a_{cand}\} | \{p_i\}_{i=1}^{N}) \propto \prod_{i=1}^{N} P(p_i | \mathcal{A} \cup \{a_{cand}\}) \tag{1}$$

This captures an inductive bias of the architect to introduce abstractions that are informative, as the probability of a program under the augmented language is proportional to the reduction in program length when re-writing the program using the candidate abstraction.

3.2 Communicating a program

Human languages are subject to pressures to be informative and to minimise cognitive load. In previous work, EC has been used to explain the language structure in the numerals and colours domains [4, 3]. We apply EC techniques to study how pressures manifest within abstraction learning. Given a program, p, selected by the architect, a one-step signalling game is created for each instruction of p which is of the form (x, a, x'): grid state x is transformed into x' by action a. We note that along with the action we pass positional information to the builder, we assume the encoding of this is pre-determined and we omit it from further notation. The architect's and builder's policies, π_{comm} and π_{bldr}, are learnable and parameterised by fully-connected neural networks. Model inference is defined as $\widehat{x'} = \pi_{bldr}(x, \pi_{comm}(a))$ which provides an estimate of the next grid-state, x'. Since our messages are discrete, we use the gumbel-softmax relaxation to sample from discrete messages which makes our model end-to-end differentiable [15]. The policies are trained to minimise the binary cross-entropy loss. The accuracy of the reconstruction is used in the bandit's reward function.

3.3 Exploration/Exploitation of Abstractions

As abstractions are introduced, there will be multiple alternative programs of different lengths available to the architect. They may differ in reconstruction accuracy, in particular, programs containing a new abstraction will initially have low accuracy. If we want the agents to learn how to use these new abstractions they must be exposed to them. Hence, this is a classic *exploration* vs. *exploitation* problem which bandit techniques are well-suited for [31]. We view the selection between programs as a contextual bandit with combinatorial actions [21]. We update the estimated value of a single action as $Q(a) \leftarrow Q(a) + \alpha(r - Q(a))$, where r is the reconstruction accuracy which is 1 if the action is successfully interpreted by the builder and 0 otherwise. The parameter α is the learning rate. The value of an entire program is calculated according as $Q(p) = \prod_{a \in |p|} \gamma Q(a)$. This captures the trade-off between program length and the communicative accuracy of instructions. The value γ is in the interval $(0, 1]$ where a lower value results in a stronger bias for shorter programs. *Exploration vs. exploitation* is undertaken using ϵ-greedy, where ϵ is held at a constant value, ensuring a constant level of exploration. Through this, we allow the architect to control the *regret*, i.e. the difference between the optimal and observed reward, arising from the introduction of new abstractions.

4 Empirical Evaluation

To evaluate PACE we investigate the following questions: 1) Does PACE result in a final language allowing for shorter communications? 2) Does communicative pressures impact what abstractions

get adopted? The behaviour of PACE is compared to two naïve baselines. The first, *No abstractions*, is PACE without the abstraction phase. The second, *Greedy*, always picks the shortest program. We use reward (also referred to as reconstruction accuracy) and average program length to compare these approaches. Unless otherwise stated results obtained are averaged over 16 runs.

PACE reduces program length PACE does indeed converge to shorter programs over interaction steps, as seen in Figure 2 (left). When we compare PACE to the *No abstraction* variant, the program lengths are halved, going from an average length of 10 to one of 4.92 ± 0.20, indicating that a more efficient language has been derived. The *Greedy* variant reduces program lengths even more, however, this comes at a price. In Figure 2 (right) we see drastic drops in accuracy for *Greedy* every time an abstraction is introduced, whereas PACE's slower introduction leads to smaller drops in accuracy. For a while, both strategies manage to recover near full accuracy, but after some time, the *Greedy* communication fails to recover and becomes progressively worse.

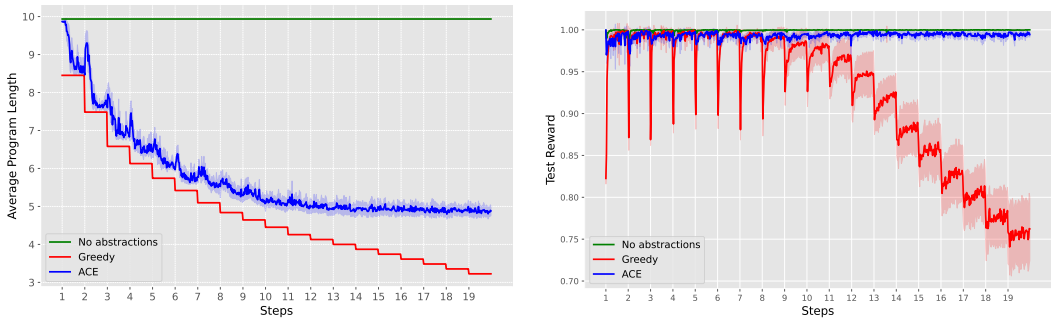
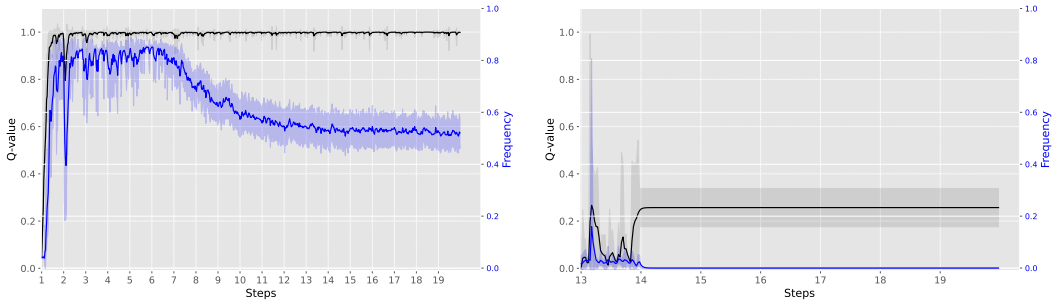


Figure 2: Comparison between PACE, *Greedy* and *No abstractions* in terms of program length and test reward over time. Line indicates mean value and shaded regions indicate the 95% confidence interval.



(a) The first abstraction being introduced successfully. (b) The thirteenth abstraction gets eventually discarded.

Figure 3: Different abstractions being introduced. We show the mean and 95% confidence interval for Q-value (black) and frequency (blue) versus epochs (of which there are 40 in a step).

The Communicative Pressures on Abstractions PACE's final language does not adopt all abstractions introduced. In Figure 3, we show some that experience different fates. Figure 3 (left) shows the Q-value and frequency of abstraction 1 from Figure 4 (a tower), which is retained in PACE's final language. Figure 3 (right) instead shows the same for abstraction 13 from Figure 4 (the letter X), which after a trial period was not retained, judged as too hard to learn by the architect. This is connected to the different frequencies of these two shapes in our dataset, with the tower being more than 20 times more common than the X. Thus, the first is much easier for the architect to learn how and when to use it than the second one.

5 Discussion & Conclusion

Addressing the challenge in [14], we initiated the study of how abstractions could be used to make communication between AI agents more efficient, as they do between humans [24]. PACE develops a compact language through interaction, resulting in shorter programs. The size of the language is the consequence of pressures that naturally arise through the need to communicate about a shared task. This is achieved through a novel combination of EC, library learning and bandits. Our work serves as a bridge between abstraction learning and efficient communication [17, 10, 33, 11, 7, 4, 3].

Beyond importance within cognitive science, we believe our results have real implications for machine learning. Significant research effort is being invested into exploring the role of natural language as a mechanism for humans to provide instructions to intelligent agents [2, 30]. Providing these systems with the capability of handling abstractions can facilitate improved cooperation between humans and agents. We believe that PACE represents a step towards equipping intelligent agents with flexible and extendable languages.

Acknowledgments and Disclosure of Funding

This work was supported by funding from CHAIR (Chalmers AI Research Center), from the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation and from the Swedish Research Council. The computations in this work were enabled by resources provided by the Swedish National Infrastructure for Computing (SNIC).

References

[1] M. Bowers, T. X. Olausson, L. Wong, G. Grand, J. B. Tenenbaum, K. Ellis, and A. Solar-Lezama. Top-down synthesis for library learning. *Proceedings of the ACM on Programming Languages*, 7(POPL):1182–1213, 2023.

[2] A. Brohan, Y. Chebotar, C. Finn, K. Hausman, A. Herzog, D. Ho, J. Ibarz, A. Irpan, E. Jang, R. Julian, et al. Do as i can, not as i say: Grounding language in robotic affordances. In *Conference on robot learning*, pages 287–318. PMLR, 2023.

[3] E. Carlsson, D. Dubhashi, and T. Regier. Iterated learning and communication jointly explain efficient color naming systems. In *Proceedings of the annual meeting of the cognitive science society*, volume 45, 2023.

[4] E. Carlsson, D. P. Dubhashi, and F. D. Johansson. Learning approximate and exact numeral systems via reinforcement learning. *Proceedings of the Annual Meeting of the Cognitive Science Society*, 43, 2021.

[5] R. Chaabouni, E. Kharitonov, D. Bouchacourt, E. Dupoux, and M. Baroni. Compositionality and generalization in emergent languages. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020*, pages 4427–4442. Association for Computational Linguistics, 2020.

[6] R. Chaabouni, F. Strub, F. Altché, E. Tarassov, C. Tallec, E. Davoodi, K. W. Mathewson, O. Tieleman, A. Lazaridou, and B. Piot. Emergent communication at scale. In *International Conference on Learning Representations*, 2022.

[7] M. Denić and J. Szymanik. Recursive numeral systems optimize the trade-off between lexicon size and average morphosyntactic complexity. *Cognitive Science*, 48(3):e13424, 2024.

[8] K. Ellis, C. Wong, M. Nye, M. Sablé-Meyer, L. Morales, L. Hewitt, L. Cary, A. Solar-Lezama, and J. B. Tenenbaum. Dreamcoder: Bootstrapping inductive program synthesis with wake-sleep library learning. In *Proceedings of the 42nd ACpM sigplan international conference on programming language design and implementation*, pages 835–850, 2021.

[9] J. Foerster, I. A. Assael, N. de Freitas, and S. Whiteson. Learning to communicate with deep multi-agent reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 29, 2016.

[10] E. Gibson, R. Futrell, J. Jara-Ettinger, K. Mahowald, L. Bergen, S. Ratnasingam, M. Gibson, S. T. Piantadosi, and B. R. Conway. Color naming across languages reflects color use. *Proceedings of the National Academy of Sciences*, 2017.

[11] E. Gibson, R. Futrell, S. P. Piantadosi, I. Dautriche, K. Mahowald, L. Bergen, and R. Levy. How efficiency shapes human language. *Trends in Cognitive Sciences*, 23(5):389 – 407, 2019.

[12] N. D. Goodman and M. C. Frank. Pragmatic language interpretation as probabilistic inference. *Trends in cognitive sciences*, 20(11):818–829, 2016.

[13] R. D. Hawkins, M. C. Frank, and N. D. Goodman. Characterizing the dynamics of learning in repeated reference games. *Cognitive science*, 44(6):e12845, 2020.

[14] M. K. Ho, D. Abel, T. L. Griffiths, and M. L. Littman. The value of abstraction. *Current Opinion in Behavioral Sciences*, 29:111–116, 2019. Artificial Intelligence.

[15] E. Jang, S. Gu, and B. Poole. Categorical reparameterization with gumbel-softmax, 2017.

[16] E. Jergéus, L. K. Oinonen, E. Carlsson, and M. Johansson. Towards learning abstractions via reinforcement learning. In *8th International Workshop on Artificial Intelligence and Cognition*, 2022.

[17] C. Kemp and T. Regier. Kinship categories across languages reflect general communicative principles. *Science (New York, N.Y.)*, 336:1049–54, 05 2012.

[18] J. Kim and A. Oh. Emergent communication under varying sizes and connectivities. *Advances in Neural Information Processing Systems*, 34:17579–17591, 2021.

[19] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[20] R. M. Krauss and S. Weinheimer. Changes in reference phrases as a function of frequency of usage in social interaction: A preliminary study. *Psychonomic Science*, 1:113–114, 1964.

[21] T. Lattimore and C. Szepesvári. *Bandit algorithms*. Cambridge University Press, 2020.

[22] A. Lazaridou and M. Baroni. Emergent multi-agent communication in the deep learning era, 2020.

[23] D. K. Lewis. *Convention: A Philosophical Study*. Wiley-Blackwell, 1969.

[24] W. McCarthy, R. Hawkins, C. Holdaway, H. Wang, and J. Fan. Learning to communicate about shared procedural abstractions. In *Proceedings of the 43rd Annual Conference of the Cognitive Science Society*, 2021.

[25] I. Mordatch and P. Abbeel. Emergence of grounded compositional language in multi-agent populations. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

[26] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

[27] T. Regier, C. Kemp, and P. Kay. Word Meanings across Languages Support Efficient Communication. *The Handbook of Language Emergence*, (January 2015):237–263, 2015.

[28] Y. Ren, S. Guo, M. Labeau, S. B. Cohen, and S. Kirby. Compositional languages emerge in a neural iterated learning model. In *International Conference on Learning Representations*, 2020.

[29] M. Rita, F. Strub, J.-B. Grill, O. Pietquin, and E. Dupoux. On the role of population heterogeneity in emergent communication. In *International Conference on Learning Representations*, 2022.

[30] L. X. Shi, Z. Hu, T. Z. Zhao, A. Sharma, K. Pertsch, J. Luo, S. Levine, and C. Finn. Yell at your robot: Improving on-the-fly from language corrections, 2024.

[31] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 1998.

[32] Y. Xu, E. Liu, and T. Regier. Numeral Systems Across Languages Support Efficient Communication: From Approximate Numerosity to Recursion. *Open Mind*, 4:57–70, 2020.

[33] N. Zaslavsky, C. Kemp, N. Tishby, and T. Regier. Color naming reflects both perceptual structure and communicative need. *Topics in Cognitive Science*, 11(1):207–219, 2019.

A Appendix

A.1 Related Literature

Emergent communication in the Architect-Builder game was first studied in humans by [24], however only on a smaller dataset with only 12 goal-scenes. They also propose a simple Bayesian model for the task, inspired by [12] and [8]. However, this is not a proper multi-agent system and is limited to assuming that abstractions are introduced automatically by a centralised process in both the architect and builder at the same time. Our work addresses this major limitation by situating the task in a multi-agent system where the introduction of abstractions is controlled by the architect, a more plausible representation. When a new abstraction is introduced, ACE allows the architect and builder to negotiate a change to their common language to incorporate it, or if it proves difficult to learn, discard it. Furthermore, our setting does not require an explicit parameter to keep the language size manageable, this appears automatically as a consequence of communicative pressures in ACE. [16] conduct a preliminary study into using multi-agent reinforcement learning for the Architect-Builder task, but make many simplifications. Most notably, there is no learning of a messaging policy and the abstractions are assumed to be immediately understood by the builder. Under this assumption, this actually collapses to a single agent setting and is not subject to pressures for efficient communication.

While EC has been used to study areas such as the impact of populations [6, 18, 29], compositionality [25, 5, 28], and naming conventions in semantic categories [3, 4], we introduce it as a tool to study the process of abstraction and combine it with library learning and bandits.

A.2 Algorithms composing PACE

Algorithm 1 shows how the components of PACE integrate. The algorithm alternates between the *Communication Phase* and the *Abstraction Phase* for s steps, where s is determined experimentally[1].

The *Communication Phase* is repeated for e iterations, determined experimentally. In each iteration, programs are sampled from π_{arch} by the bandit's ϵ-greedy strategy and are used to form a dataset of grid-state transitions. This dataset forms the basis of our signalling game and is used to train π_{comm}, π_{bldr} and to update the Q-values of actions. Here, we expect a language to emerge which is subject to pressures for efficient communication.

In the *Abstraction Phase* sub-sequences in the programs are identified. The sub-sequence that maximises Equation 1 is selected. This abstraction, a_{new}, is introduced into $\mathcal{A}$. a_{new} may allow existing programs to be rewritten: in this case, the resulting shorter programs are introduced into π_{arch} for the respective goal-scene. Finally, $Q(a_{new})$ is initialised.

We also provide more detailed algorithms for the focal points of PACE's methodology. Algorithms 2 and 3 refer to the *Communication Phase*, which we break down further into two parts: the dataset generation for the signalling game and the communication round done via EC. Algorithm 4 instead refers to the *Abstraction Phase*.

A.3 Evolution of Language and Resulting Programs

In Figure 4 we report the abstraction that PACE introduces in a single run[2]. Abstractions circled in green are in the final language. In Figure 5, we show how the composition of our language changes throughout the repeated interactions. The relative proportion of primitives is reduced in favour of abstractions referring to either one of the 31 discrete shapes in the dataset (roughly 40%), or a sub-shape (roughly 20%). We show how this change in distribution impacts the composition of

[1] We found that 20 steps were a good choice for our setting, after which the language stabilises and no more abstractions are introduced in the emergent language.

[2] Further examples, which are quite similar, can be found in our code repository.

Algorithm 1 Procedural Abstractions for Communicating Efficiently (PACE)

Require: initial π_{arch}, initial $\mathcal{A}$, q_{init}, α, ϵ, e, s
Ensure: trained π_{comm}, π_{bldr}, Q, updated π_{arch}, $\mathcal{A}$
Initialise neural networks π_{comm} and π_{bldr}
Initialise action Q-values $Q(a)$ to q_{init}, $\forall a \in \mathcal{A}$
for s steps **do**
  // Communication Phase
  for e rounds **do**
    For each goal choose a program in π_{arch} via bandit
    Create a training dataset D of triples (x, a, x') of transitions based on chosen programs
    Play signalling games with D to train π_{comm}, π_{bldr} and Q-values
  end for
  Optional: evaluate on hold-out test dataset
  // Abstraction Phase
  Find action sub-sequences in programs
  Choose new abstraction a_{new} that maximises Eq. 1
  Compress existing programs using a_{new}, if possible
  For each goal, add new programs to π_{arch}
  Add a_{new} to $\mathcal{A}$ and initialise $Q(a_{new})$ to q_{init}
end for

Algorithm 2 Dataset generation via bandit

Require: π_{arch}, Q, bandit strategy
Ensure: D *# A dataset of signalling games*
$D = \{\}$
for $g \in G$ **do**
  Get set of programs for g from π_{arch}
  Compute Q-values for programs
  Pick program, p, via bandit strategy
  break p in a set of triples (x, a, x') of transitions
  $D \leftarrow D \bigcup \{(x_i, a_i, x'_i)\}_{i \in |p|}$
end for

Algorithm 3 A round of communication of PACE

Require: D, π_{comm}, π_{bldr}, Q
Ensure: trained π_{comm}, π_{bldr}, Q
 # Play signalling game and update policies
while D is not empty **do**
  Sample batch B from D without replacement
  $loss = 0$
  for $(x, a, x') \in B$ **do**
    $m = \pi_{comm}(a)$
    $\widehat{x'} = \pi_{bldr}(x, m)$
    $loss \leftarrow loss + \mathcal{L}(x', \widehat{x'})$
    Update $Q(a)$ for all a in B
  end for
  Update π_{comm} and π_{bldr} via gradient descent
end while

Algorithm 4 Abstraction phase

Require: π_{arch}, $\mathcal{A}$
Ensure: updated $\mathcal{A}$ and π_{arch}
$Z = \{\}$ *# The set of preferred programs*
for $g \in G$ **do**
  Get set of programs for g from π_{arch}
  Compute Q-values for programs
  Pick programs, p, via greedy bandit strategy
  $Z \leftarrow Z \bigcup \{p\}$
end for
Find all sub-sequences in Z (candidate abstractions)
Select best abstraction, a_{new}, via Equation 1
$A \leftarrow A \bigcup \{a_{new}\}$
for $g \in G$ **do**
  for $p \in \pi_{arch}(g)$ **do**
    if a_{new} is applicable to p **then**
      Add p re-written in terms of a_{new} to $\pi_{arch}(g)$
    end if
  end for
end for

programs in Figure 6 with two goal-scenes examples. We see that the introduction of new abstractions enables for programs to be rewritten much more compactly.

A.4 Q-value initialisation: Pessimism is best

It is not obvious how the initialisation of the Q-values for new abstractions may impact regret. Compelling cases can be made for both optimistic (a more aggressive introduction) and pessimistic (a more gradual introduction) strategies. Empirically we find the pessimistic strategy works better in our case. Here, we establish its impact on learning of the common "tower" abstraction. We compare a pessimistic strategy ($q_{init} = 0$) to an optimistic strategy ($q_{init} = 1$) in terms of cumulative regret, defined as $Regret_T = \sum_{t=1}^{T} (r^* - r_t)$. In Figure 7, we see that a pessimistic initialisation of 0 incurs significantly less cumulative regret than an optimistic value of 1. This is attributable to the optimistic variant's tendency to force the abstraction into use before the builder has had the opportunity to become familiar with it. The pessimistic agent introduces the new abstraction more gradually while still providing the opportunity for learning of a common language.

A.5 Implementation Details

We implement PACE within Python version 3.9.18 and PyTorch version 2.2.1 [26] and run experiments on Apple M2 Pro CPU (16 GB) on MacOS 14.4.1. The communication policies, π_{comm} and π_{bldr}, are deep neural networks which have 1 and 2 hidden layers, respectively. Each hidden layer contains 200 neurons and utilises ReLU activation functions. The outputted layer of π_{comm} is comprised of 30 neurons, which is its maximum vocabulary size. The output layer of π_{bldr} is comprised of 81 neurons, which is the size of the grid. Each output neuron in π_{bldr} represents the builder's belief that the grid index is occupied. We use the gumbel-softmax relaxation with the *hard* parameter so that the networks can be differentiated end-to-end. The model is optimised to minimise binary cross entropy error and we use a learning rate of 0.0009 in conjunction with the ADAM optimiser [19]. The bandit hyperparameters α, γ, ϵ and q_{init} are set to 0.5, 0.99, 0.1, and 0.0 respectively. Note, that before an abstraction is introduced we prune all but the 3 best programs (determined empirically) for each goal-scene in π_{arch} to not incur an exponential growth in the number of programs for each goal-scene. We also set the number of epochs e to 40. All hyperparameters chosen are experimentally determined through grid search.

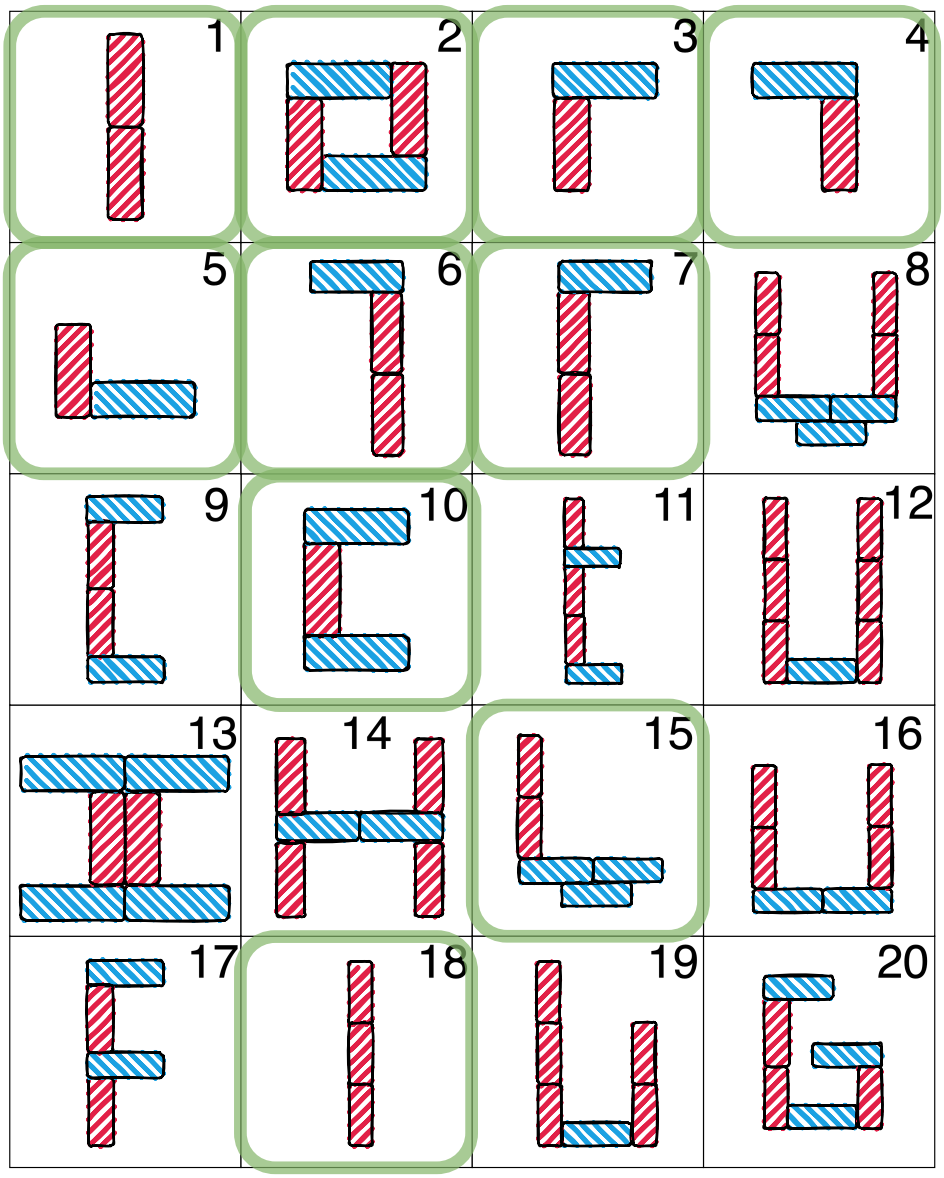


Figure 4: The abstractions introduced by PACE in an example run. We circle in green the abstractions that are adopted by the architect in its final language.

A.6 Dataset details

We compose the goal-scene dataset G similarly to [24], where each scene is composed of two shapes placed side by side that are in turn built exclusively of the two primitive building blocks. These shapes are of different sizes and resemble either uppercase or lowercase letters from the English alphabet, resulting in 31 unique shapes, with multiple sub-shapes reoccurring in different shapes. The dataset results in a total of 961 goal-scenes. This is over 100 times bigger than that of [24], which enables training neural agents. We split the dataset into 930 training scenes and 31 test scenes. These splits are constructed to ensure the distribution over shapes is the same. However, in the test set, the ordered pair of shapes constituting each goal-scene, does *not* appear in the training set.

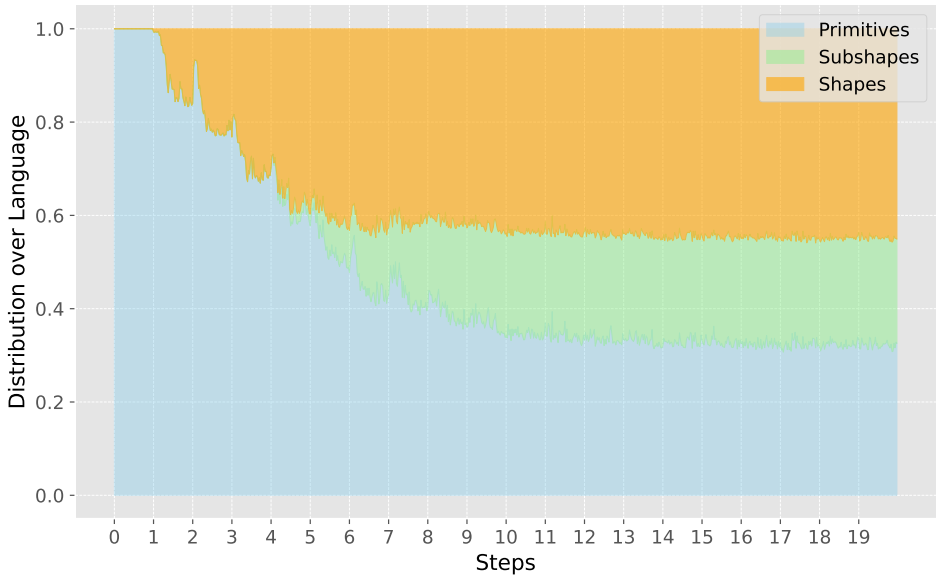


Figure 5: Relative proportion of actions as the language changes over training. Primitives refer to the initial actions, shapes refer to one of the 31 discrete shapes appearing in goal-scenes, and sub-shapes refer to anything else.

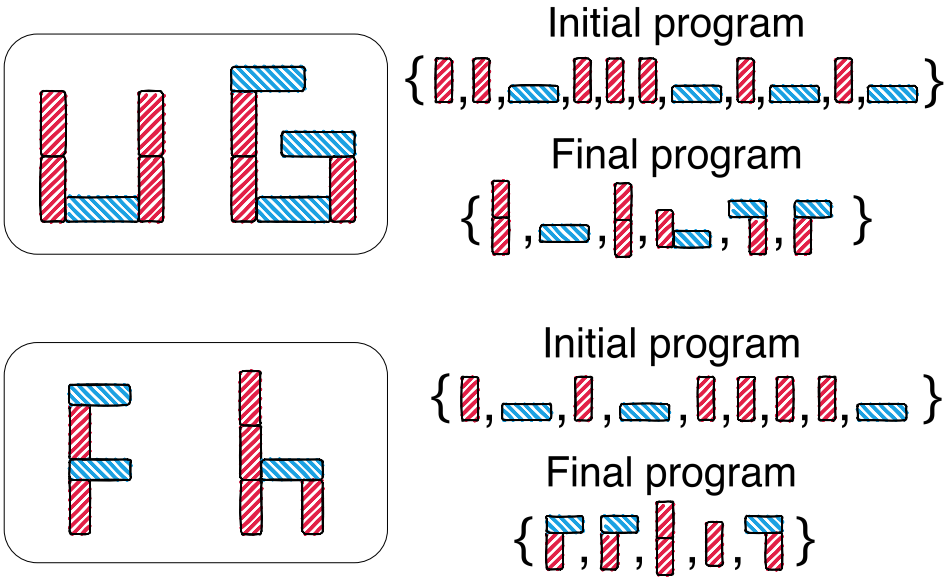


Figure 6: Two goal scenes with representations of their initial and final programs chosen by PACE.

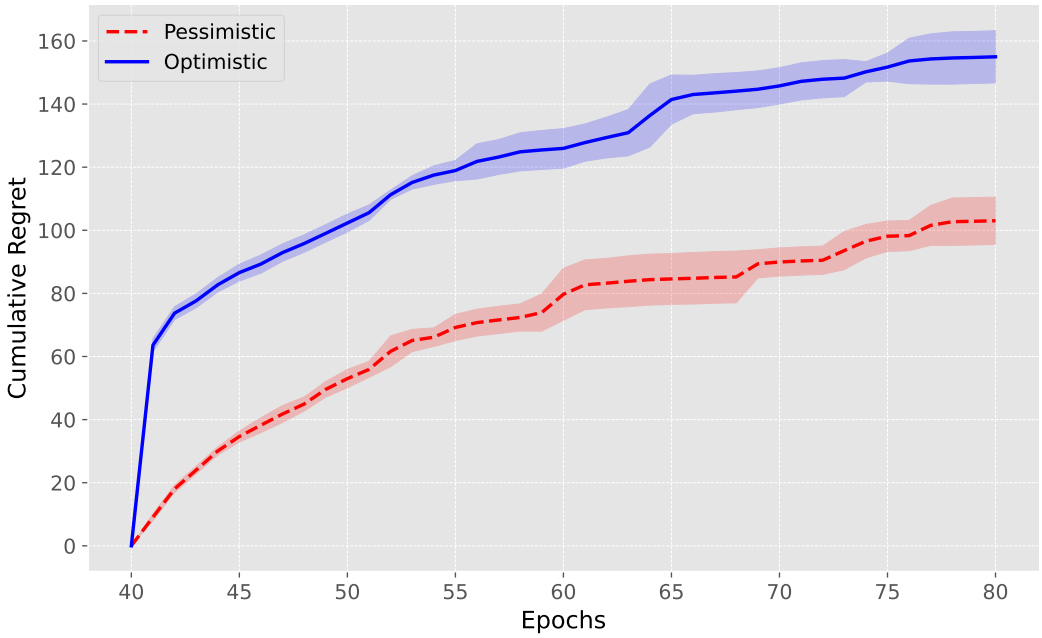


Figure 7: Cumulative regret for an optimistic and pessimistic initialisation of Q-values. Line indicates mean performance and shaded regions indicate the 95% confidence interval for 16 random seeds. Epoch interval is between step 1 and step 2.

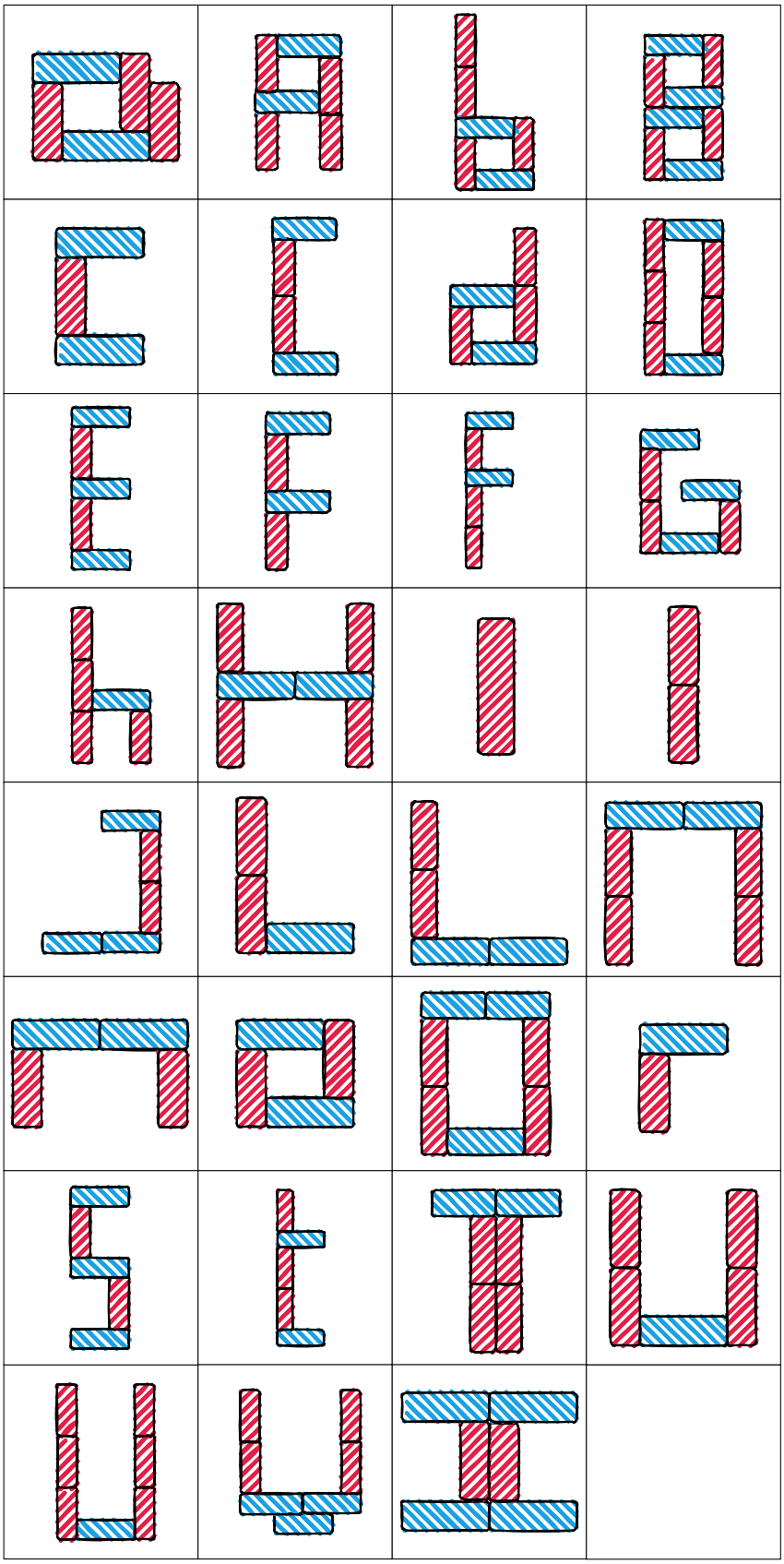


Figure 8: The 31 shapes that in pair compose each goal-scene.