Synthesizing Video Trajectory Queries

Stephen Mell
University of Pennsylvania
sm1@cis.upenn.edu

Favyen Bastani
Massachusetts Institute of Technology
favyen@csail.mit.edu

Steve Zdancewic
University of Pennsylvania
stevez@cis.upenn.edu

Osbert Bastani
University of Pennsylvania
obastani@seas.upenn.edu

Abstract

We propose a novel framework called QUIVR for example-based synthesis of queries to identify events of interest in video data; these queries are essentially regular expressions that operate over object trajectories predicted by a deep object tracking model. For instance, QUIVR can be used to identify instances of human driving behaviors such as lane changes, which are important for designing planning algorithms for autonomous cars. To make the synthesis efficient, we use overapproximations to prune invalid branches of the query search space, including using a quantitative variant of our query semantics to efficiently prune the search space over parameter values. We also propose two optimizations for speeding up the execution of our queries. Finally, we leverage active learning to disambiguate between multiple consistent candidate queries by collecting additional labels from the user. We evaluate QUIVR on a benchmark of 11 tasks, and demonstrate that it can synthesize accurate queries for each task given just a few examples, and that our pruning strategy and optimizations substantially reduce synthesis time.

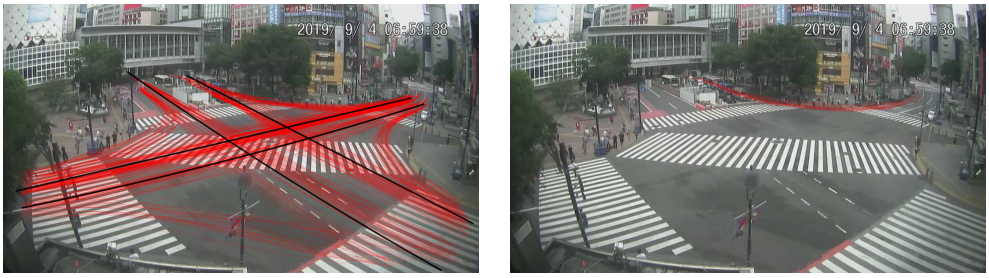
1 Introduction

An important application of deep learning is to try and classify behaviors in trajectory data. For instance, understanding behaviors of human drivers and pedestrians is critical for designing controllers for autonomous cars that interact with humans [1, 2, 3]. Thus, engineers must identify examples of driving patterns in the data to design and debug algorithms that can plan for various scenarios; furthermore, the autonomous car must detect patterns to react to them. There is also interest in querying traffic video data to quantify the frequency of potentially dangerous situations such as cars driving too close together [4] or stopping in the middle of the road [5]. Trajectory classification can also be applied to understanding animal behaviors [6, 7, 8] and to sports analytics [9, 8].

We consider the problem of classifying trajectories that have been extracted from video data. In particular, there has been a great deal of interest in designing systems for executing queries over trajectory data to identify behaviors of interest [10, 11, 12, 13, 14, 15, 16, 17]. However, writing these queries can be challenging for end users, especially since they often contain real-valued parameters that must be tuned based on the structure of the underlying data.

To address this challenge, we propose QUIVR, which consists of a novel language for querying object trajectories in video data, together with an algorithm for synthesizing queries in this language from just a few input-output examples. To enable efficient synthesis, QUIVR leverages strategies to identify and prune inconsistent branches of the search space; importantly, it uses quantitative semantics to help prune the search space over real-valued parameters of the query. It also uses sparse matrix semantics to further improve scalability. Finally, given only a few examples, there are often multiple consistent

35th Conference on Neural Information Processing Systems (aiplans 2021), Sydney, Australia.



(a) (b)

$\langle \mathsf{True} \rangle^* \,;\, \langle \mathsf{InLane1} \rangle \,;\, \langle \mathsf{True} \rangle^* \,;\, \langle \mathsf{InLane2} \rangle \,;\, \langle \mathsf{True} \rangle^*$

Figure 1: (a) A video frame from the traffic camera, along with the object trajectories (red) and the manually annotated lanes (black). (b) The trajectories selected by the query at the bottom. The query at the bottom is designed to select cars turning at the intersection.

queries; thus, QUIVR leverages active learning to disambiguate among these queries. Finally, we evaluate QUIVR on a benchmark of 11 tasks, demonstrating that it can synthesize accurate queries given just a few examples, that our pruning strategy and sparse matrix semantics substantially reduce synthesis time, and that our active learning strategy quickly reduces ambiguity.

2 Overview

For the sake of concreteness, we consider a problem in autonomous driving, where an engineer is designing a control algorithm for an autonomous car and would like to identify certain driving patterns in video data. However, the techniques outlined here are applicable to sequential data more broadly.

Video data. For simplicity, we focus on video data gathered from fixed-position traffic cameras. Such data has recently been identified as a rich source of driving behaviors [18, 19, 17]. For instance, in our evaluation, we use the YTStreams dataset [14], which includes 60 hours of video from several traffic cameras collected from live YouTube feeds. The engineer may want to write queries to identify driving patterns specific to the lanes visible in a particular video. For example, a single frame from such a video is shown in Figure 1 (a); as can be seen, we have already used an off-the-shelf object tracker to identify all object trajectories of cars in the data [20], shown in red.

Query language. We propose a novel language for expressing queries over such patterns. A query takes as input a representation of a trajectory as a sequence of object states (e.g., position, velocity, and acceleration) in successive frames of the video, and outputs whether the trajectory matches its semantics. Our language is based on regular expressions—in particular, Kleene algebras with tests [21]. Thus, a query is a composition of a user-extensible set of predicates using the conjunction, disjunction, sequencing, and iteration (Kleene star) operators

$$Q ::= \varphi \mid Q \wedge Q \mid Q \vee Q \mid Q \,;\, Q \mid Q^* \mid (Q^k \coloneqq Q \,;\, \ldots \,;\, Q).$$

A key feature of our language is that it includes predicates that operate over multiple frames. Figure 1 shows a query for identifying cars changing lanes. We provide additional details on our query language in Section A.

Predicates. Along with providing video data, the engineer uses QUIVR to develop a set of predicates that select trajectories satisfying relevant semantic constraints. QUIVR will compose these predicates to synthesize queries, and the predicates can be reused to synthesize multiple queries. In Figure 1 (a), the engineer has manually annotated the lanes of interest in this video (black), to specify four $\mathsf{InLaneK}$ predicates that select trajectories of cars driving in each lane K visible in the video. Predicates may be configured by real-valued parameters. For example, the query

$$\langle \mathsf{InLane1} \rangle^{10} \wedge \langle \mathsf{MinAvgAccel}_\theta \rangle$$

searches for trajectories where the car stays in lane 1 for 10 frames, and due to the predicate $\mathsf{MinAvgAccel}_\theta$, that the car has an average acceleration of at least θ across those same 10 frames.

Algorithm 1 Compute all queries $Q_\theta \in \bar{\mathcal{Q}}$ consistent with input-output examples $W \subseteq \mathcal{W}$.

procedure SYNTHESIZE(W)
  $\ell_0 \leftarrow \{??\}$
  $\ell^* \leftarrow \varnothing$
  while $\ell_0 \neq \varnothing$ **do**
    $Q \leftarrow \mathsf{Pop}(\ell_0)$
    if $\neg\psi_W^\#(Q)$ **then**
      continue
    else if $Q \in \mathcal{Q}_{\text{sketch}}$ **then**
      $\ell^* \leftarrow \ell^* \cup \left\{ Q_\theta \mid \theta_h \in \Theta_h \cap \Theta_h^\#(Q, W) \right\}$
    else
      $\ell_0 \leftarrow \ell_0 \cup \mathsf{Children}(Q)$
    end if
  end while
  return $\{Q_\theta \in \ell^* \mid \psi_W(Q_\theta)\}$
end procedure

While QUIVR includes a wide range of built-in operations that can be reused to specify video-specific predicates, a key feature of our framework is that the set of available predicates is highly extensible. In particular, the user can provide their own predicates as long as they provide their semantics; otherwise, there are no constraints on them.

Multi-object queries. So far, we have focused on queries that identify trajectories by processing each trajectory in isolation. A key feature of our framework is that users can express queries over multiple trajectories—for example,

$$\left(\langle \mathsf{InLane1}(B)\rangle^* \wedge \langle \mathsf{ChangeLane2To1}(A)\rangle\right) ; \langle \mathsf{InFront}(A, B)\rangle.$$

This query says that car B is in lane 1 while car A changes from lane 2 to lane 1, and car A ends up in front of car B. Note that the predicates now include variables indicating which object they refer to; in addition, it also includes the predicate $\mathsf{InFront}(A, B)$ that refers to multiple trajectories.

Synthesis. To specify a driving pattern, the engineer provides a small number of initial input-output examples, including both positive examples (i.e., trajectories they want to select) and negative examples (i.e., trajectories they want to omit). Then, QUIVR synthesizes a query that correctly labels these examples. In Figure 1 (b), we show the result of executing the query shown at the bottom, which is synthesized to identify left turns in the data.

The initial synthesis is done via top-down search over a syntax of partial queries

$$Q ::= ?? \mid \varphi_{??} \mid \varphi \mid Q \wedge Q \mid Q \vee Q \mid Q\,;\,Q \mid Q^*.$$

Note that holes (denoted $??$) can be over either expressions or predicate parameters.

Optimizations. Our synthesis algorithm uses standard techniques to prune the search space over expressions, and we introduce a novel quantitative semantics for queries that allow us to efficiently select parameter values. At a high level, for each positive example of a trajectory, it evaluates the partial query on this trajectory while making optimistic assumptions about the value of each hole; if the query still evaluates to false, then there can be no way of completing the partial query that results in a valid concrete query. A similar strategy can be used for negative examples, using pessimistic assumptions instead of optimistic ones. See Algorithm 1 and Section B for details.

Active learning. Typically, there are many queries consistent with the initial input-output examples. To disambiguate among these queries, our algorithm actively asks the engineer to label additional trajectories as positive or negative. While it may be hard for users to identify positive examples in the video data, it is usually easy for them to determine whether a given trajectory is positive or negative. Our algorithm uses an active learning strategy that greedily selects the next trajectory to label to be the one that most reduces uncertainty in expectation [22, 23, 24]; see Section B for details.

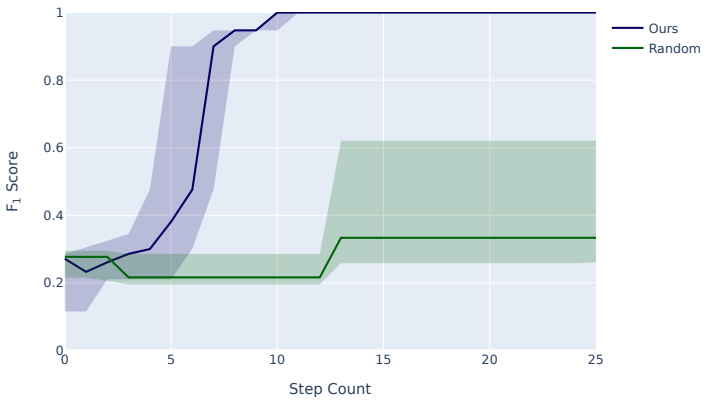


Figure 2: F_1 score as a function of the number of user labels for query A from the Warsaw 1-object dataset. The line is the median F_1 score over all queries consistent with examples so far; the shaded region is the 25th to 75th percentile of F_1 scores.

Scenario	Boolean	Quantitative	Sparse	GPU
Shibuya, 1-object	287	46	38	22
Warsaw, 1-object	184	31	33	20
Warsaw, 2-object	5277	836	538	–

Table 1: Running time of synthesis, in seconds, for the Boolean, quantitative, and sparse semantics, with 0 steps of active learning. For the GPU results, we use the quantitative semantics. Missing results indicate an out-of-memory error. Results are each averaged over 3 random seeds.

3 Evaluation

To evaluate our synthesis procedure, we wrote 11 different queries across two different videos ("Shibuya" and "Warsaw"). Five of the Warsaw queries were over pairs of objects and the others were over a single object. For each query, all of the tracks in the dataset were given ground-truth labels according to whether they matched the query or not. Tracks from the first half of each video were used for training while those from the second half were used for testing. From these labeled training examples, 2 positive and 10 negative examples were chosen at random. We then applied our synthesis algorithm, followed by active learning. We provide detailed results in Section D.

Accuracy. With few initial examples (2 positive and 10 negative) and just 5 to 10 steps of active learning, we are able to achieve near perfect accuracy on a held-out test set. Figure 2 shows the F_1 score on this test set as the number of active learning steps increases, both for our active labeling strategy and a random selection baseline.

Runtime. As shown in Table 1, our optimizations significantly speed up synthesis. In particular, the quantitative semantics speeds up synthesis by almost an order of magnitude. The use of GPUs yields further improvements, but occassionally lead to out-of-memory errors; when memory constraints do not allow for GPUs, our sparse semantics provides the best performance.

4 Conclusion

We have proposed a novel framework for synthesizing queries over video trajectory data. Our language is based on regular expressions, extended to include operators such as conjunction as well as predicates over subsequences. Given only a few IO examples, our framework can efficiently synthesize queries in our language consistent with those examples. When multiple consistent candidate queries are

found, our algorithm can also actively label additional examples to disambiguate between them. In our evaluation, we demonstrate the effectiveness of our synthesis approach on a number of queries that identify interesting driving behaviors.

Acknowledgments and Disclosure of Funding

We gratefully acknowledge support from DARPA HR001120C0015, NSF CCF-1917852, NSF CCF1910769, and ARO W911NF-20-1-0080. The views expressed are those of the authors and do not reflect the official policy or position of the Department of Defense, the Army Research Office, or the U.S. Government. We thank the anonymous reviewers for their insightful and helpful comments.

References

[1] Dorsa Sadigh, S Shankar Sastry, Sanjit A Seshia, and Anca Dragan. Information gathering actions over human internal state. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 66–73. IEEE, 2016.

[2] Dorsa Sadigh, Shankar Sastry, Sanjit A Seshia, and Anca D Dragan. Planning for autonomous cars that leverage effects on human actions. In *Robotics: Science and Systems*, volume 2. Ann Arbor, MI, USA, 2016.

[3] Edward Schmerling, Karen Leung, Wolf Vollprecht, and Marco Pavone. Multimodal probabilistic model-based planning for human-robot interaction. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1–9. IEEE, 2018.

[4] Jeffrey Wishart, Steven Como, Maria Elli, Brendan Russo, Jack Weast, Niraj Altekar, Emmanuel James, and Yan Chen. Driving safety performance assessment metrics for ads-equipped vehicles. *SAE Technical Paper*, 2(2020-01-1206), 2020.

[5] Favyen Bastani, Osbert Bastani, Arjun Balasingam, Songtao He, Ziwen Jiang, Radhika Mittal, Mohammad Alizadeh, Hari Balakrishnan, Tim Kraska, and Sam Madden. Skyquery: Optimizing video queries over uavs. In *Onward!*, 2021.

[6] David Tweed and Andrew Calway. Tracking multiple animals in wildlife footage. In *Object recognition supported by user interaction for service robots*, volume 2, pages 24–27. IEEE, 2002.

[7] Margrit Betke, Diane E Hirsh, Angshuman Bagchi, Nickolay I Hristov, Nicholas C Makris, and Thomas H Kunz. Tracking large variable numbers of objects in clutter. In *2007 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8. IEEE, 2007.

[8] Ameesh Shah, Eric Zhan, Jennifer Sun, Abhinav Verma, Yisong Yue, and Swarat Chaudhuri. Learning differentiable programs with admissible neural heuristics. *Advances in Neural Information Processing Systems*, 33, 2020.

[9] Eric Zhan, Albert Tseng, Yisong Yue, Adith Swaminathan, and Matthew Hausknecht. Learning calibratable policies using programmatic style-consistency. In *ICML*, 2020.

[10] Daniel Kang, Peter Bailis, and Matei Zaharia. Blazeit: optimizing declarative aggregation and limit queries for neural network-based video analytics. *Proceedings of the VLDB Endowment*, 13(4):533–546, 2019.

[11] Daniel Y Fu, Will Crichton, James Hong, Xinwei Yao, Haotian Zhang, Anh Truong, Avanika Narayan, Maneesh Agrawala, Christopher Ré, and Kayvon Fatahalian. Rekall: Specifying video events using compositions of spatiotemporal labels. *arXiv preprint arXiv:1910.02993*, 2019.

[12] Daniel Kang, Ankit Mathur, Teja Veeramacheneni, Peter Bailis, and Matei Zaharia. Jointly optimizing preprocessing and inference for dnn-based visual analytics. *arXiv preprint arXiv:2007.13005*, 2020.

[13] Daniel Kang, Deepti Raghavan, Peter Bailis, and Matei Zaharia. Model assertions for monitoring and improving ml model. *arXiv preprint arXiv:2003.01668*, 2020.

[14] Favyen Bastani, Songtao He, Arjun Balasingam, Karthik Gopalakrishnan, Mohammad Alizadeh, Hari Balakrishnan, Michael Cafarella, Tim Kraska, and Sam Madden. Miris: Fast object track queries in video. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, pages 1907–1921, 2020.

[15] Favyen Bastani, Oscar Moll, and Sam Madden. Vaas: video analytics at scale. *Proceedings of the VLDB Endowment*, 13(12):2877–2880, 2020.

[16] Oscar Moll, Favyen Bastani, Sam Madden, Mike Stonebraker, Vijay Gadepally, and Tim Kraska. Exsample: Efficient searches on video repositories through adaptive sampling. *arXiv preprint arXiv:2005.09141*, 2020.

[17] Favyen Bastani, Osbert Bastani, Arjun Balasingam, Songtao He, Ziwen Jiang, Radhika Mittal, Mohammad Alizadeh, Hari Balakrishnan, Tim Kraska, and Sam Madden. Skyquery: Optimizing video queries over uavs. 2019.

[18] A Robicquet, A Sadeghian, A Alahi, and S Savarese. Learning social etiquette: Human trajectory prediction in crowded scenes. In *European Conference on Computer Vision (ECCV)*, 2016.

[19] Julian Bock, Robert Krajewski, Tobias Moers, Steffen Runde, Lennart Vater, and Lutz Eckstein. The ind dataset: A drone dataset of naturalistic road user trajectories at german intersections. *arXiv preprint arXiv:1911.07602*, 2019.

[20] Nicolai Wojke, Alex Bewley, and Dietrich Paulus. Simple online and realtime tracking with a deep association metric. In *2017 IEEE international conference on image processing (ICIP)*, pages 3645–3649. IEEE, 2017.

[21] Dexter Kozen. Kleene algebra with tests. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 19(3):427–443, 1997.

[22] N Roy and A McCallum. Toward optimal active learning through sampling estimation of error reduction. int. conf. on machine learning, 2001.

[23] Sanjoy Dasgupta. Analysis of a greedy active learning strategy. In *Advances in neural information processing systems*, pages 337–344, 2005.

[24] Ruyi Ji, Jingjing Liang, Yingfei Xiong, Lu Zhang, and Zhenjiang Hu. Question selection for interactive program synthesis. In *PLDI*, pages 1143–1158, 2020.

[25] Leslie G Valiant. General context-free recognition in less than cubic time. *Journal of computer and system sciences*, 10(2):308–315, 1975.

[26] Rajeev Alur, Rastislav Bodik, Garvit Juniwal, Milo MK Martin, Mukund Raghothaman, Sanjit A Seshia, Rishabh Singh, Armando Solar-Lezama, Emina Torlak, and Abhishek Udupa. *Syntax-guided synthesis*. IEEE, 2013.

[27] Armando Solar-Lezama and Rastislav Bodik. *Program synthesis by sketching*. Citeseer, 2008.

[28] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*, 2018.

[29] Favyen Bastani, Songtao He, Sofiane Abbar, Mohammad Alizadeh, Hari Balakrishnan, Sanjay Chawla, Sam Madden, and David DeWitt. Roadtracer: Automatic extraction of road networks from aerial images. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4720–4728, 2018.

[30] Songtao He, Favyen Bastani, Sofiane Abbar, Mohammad Alizadeh, Hari Balakrishnan, Sanjay Chawla, and Sam Madden. Roadrunner: improving the precision of road network inference from gps trajectories. In *Proceedings of the 26th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 3–12, 2018.

[31] Alex Bewley, Zongyuan Ge, Lionel Ott, Fabio Ramos, and Ben Upcroft. Simple online and realtime tracking. In *2016 IEEE International Conference on Image Processing (ICIP)*, pages 3464–3468, 2016.

[32] Alberto Bartoli, Giorgio Davanzo, Andrea De Lorenzo, Eric Medvet, and Enrico Sorio. Automatic synthesis of regular expressions from examples. *Computer*, 47(12):72–80, 2014.

[33] Mina Lee, Sunbeom So, and Hakjoo Oh. Synthesizing regular expressions from examples for introductory automata assignments. In *Proceedings of the 2016 ACM SIGPLAN International Conference on Generative Programming: Concepts and Experiences*, pages 70–80, 2016.

[34] Qiaochu Chen, Xinyu Wang, Xi Ye, Greg Durrett, and Isil Dillig. Multi-modal synthesis of regular expressions. In *Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 487–502, 2020.

[35] Rong Pan, Qinheping Hu, Gaowei Xu, and Loris D'Antoni. Automatic repair of regular expressions. *Proceedings of the ACM on Programming Languages*, 3(OOPSLA):1–29, 2019.

[36] Kevin Ellis, Armando Solar-Lezama, and Josh Tenenbaum. Unsupervised learning by program synthesis. In *Advances in neural information processing systems*, pages 973–981, 2015.

[37] Kevin Ellis, Daniel Ritchie, Armando Solar-Lezama, and Josh Tenenbaum. Learning to infer graphics programs from hand-drawn images. In *Advances in neural information processing systems*, pages 6059–6068, 2018.

[38] Yonglong Tian, Andrew Luo, Xingyuan Sun, Kevin Ellis, William T Freeman, Joshua B Tenenbaum, and Jiajun Wu. Learning to infer and execute 3d shape programs. *arXiv preprint arXiv:1901.02875*, 2019.

[39] Halley Young, Osbert Bastani, and Mayur Naik. Learning neurosymbolic generative models via program synthesis. In *ICML*, 2019.

[40] Kevin Ellis, Maxwell Nye, Yewen Pu, Felix Sosa, Josh Tenenbaum, and Armando Solar-Lezama. Write, execute, assess: Program synthesis with a repl. In *Advances in Neural Information Processing Systems*, pages 9169–9178, 2019.

[41] Abhinav Verma, Vijayaraghavan Murali, Rishabh Singh, Pushmeet Kohli, and Swarat Chaudhuri. Programmatically interpretable reinforcement learning. In *ICML*, 2018.

[42] Osbert Bastani, Yewen Pu, and Armando Solar-Lezama. Verifiable reinforcement learning via policy extraction. In *Advances in neural information processing systems*, pages 2494–2504, 2018.

[43] Jeevana Priya Inala, Osbert Bastani, Zenna Tavares, and Armando Solar-Lezama. Synthesizing programmatic policies that inductively generalize. In *International Conference on Learning Representations*, 2019.

[44] Abhinav Verma, Hoang Le, Yisong Yue, and Swarat Chaudhuri. Imitation-projected programmatic reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 15752–15763, 2019.

[45] Lazar Valkov, Dipak Chaudhari, Akash Srivastava, Charles Sutton, and Swarat Chaudhuri. Houdini: Lifelong learning as program synthesis. In *Advances in Neural Information Processing Systems*, pages 8687–8698, 2018.

[46] Qiaochu Chen, Aaron Lamoreaux, Xinyu Wang, Greg Durrett, Osbert Bastani, and Isil Dillig. Web question answering with neurosymbolic program synthesis. In *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation*, pages 328–343, 2021.

[47] Eric Schkufza, Rahul Sharma, and Alex Aiken. Stochastic superoptimization. *ACM SIGARCH Computer Architecture News*, 41(1):305–316, 2013.

[48] Phitchaya Mangpo Phothilimthana, Aditya Thakur, Rastislav Bodik, and Dinakar Dhurjati. Scaling up superoptimization. In *Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 297–310, 2016.

[49] Zhihao Jia, James Thomas, Todd Warszawski, Mingyu Gao, Matei Zaharia, and Alex Aiken. Optimizing dnn computation with relaxed graph substitutions. In *Proceedings of the 2nd Conference on Systems and Machine Learning (SysML'19)*, 2019.

[50] Sasa Misailovic, Michael Carbin, Sara Achour, Zichao Qi, and Martin C Rinard. Chisel: Reliability-and accuracy-aware optimization of approximate computational kernels. *ACM Sigplan Notices*, 49(10):309–328, 2014.

[51] James Bornholt, Emina Torlak, Dan Grossman, and Luis Ceze. Optimizing synthesis with metasketches. In *Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 775–788, 2016.

[52] Aditya V Nori, Sherjil Ozair, Sriram K Rajamani, and Deepak Vijaykeerthy. Efficient synthesis of probabilistic programs. *ACM SIGPLAN Notices*, 50(6):208–217, 2015.

[53] Swarat Chaudhuri, Martin Clochard, and Armando Solar-Lezama. Bridging boolean and quantitative synthesis using smoothed proof search. In *Proceedings of the 41st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 207–220, 2014.

[54] Sumit Gulwani. Automating string processing in spreadsheets using input-output examples. *ACM Sigplan Notices*, 46(1):317–330, 2011.

[55] Oleksandr Polozov and Sumit Gulwani. Flashmeta: A framework for inductive program synthesis. In *Proceedings of the 2015 ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications*, pages 107–126, 2015.

[56] John K Feser, Swarat Chaudhuri, and Isil Dillig. Synthesizing data structure transformations from input-output examples. *ACM SIGPLAN Notices*, 50(6):229–239, 2015.

[57] Matej Balog, Alexander L Gaunt, Marc Brockschmidt, Sebastian Nowozin, and Daniel Tarlow. Deepcoder: Learning to write programs. In *ICLR*, 2016.

[58] Yanju Chen, Chenglong Wang, Osbert Bastani, Isil Dillig, and Yu Feng. Program synthesis using deduction-guided reinforcement learning. In *International Conference on Computer Aided Verification*, pages 587–610. Springer, 2020.

[59] Peter-Michael Osera and Steve Zdancewic. Type-and-example-directed program synthesis. *ACM SIGPLAN Notices*, 50(6):619–630, 2015.

[60] Nadia Polikarpova, Ivan Kuraj, and Armando Solar-Lezama. Program synthesis from polymorphic refinement types. *ACM SIGPLAN Notices*, 51(6):522–538, 2016.

[61] Yu Feng, Ruben Martins, Osbert Bastani, and Isil Dillig. Program synthesis using conflict-driven learning. *ACM SIGPLAN Notices*, 53(4):420–435, 2018.

[62] Yu Feng, Ruben Martins, Jacob Van Geffen, Isil Dillig, and Swarat Chaudhuri. Component-based synthesis of table consolidation and transformation tasks from examples. *ACM SIGPLAN Notices*, 52(6):422–436, 2017.

[63] Chenglong Wang, Alvin Cheung, and Rastislav Bodik. Synthesizing highly expressive sql queries from input-output examples. In *Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 452–466, 2017.

[64] Georgios E Fainekos and George J Pappas. Robustness of temporal logic specifications for continuous-time signals. *Theoretical Computer Science*, 410(42):4262–4291, 2009.

[65] Paulo Tabuada and Daniel Neider. Robust linear temporal logic. *Computer Science Logic 2016*, 2016.

[66] Jyotirmoy V Deshmukh, Alexandre Donzé, Shromona Ghosh, Xiaoqing Jin, Garvit Juniwal, and Sanjit A Seshia. Robust online monitoring of signal temporal logic. *Formal Methods in System Design*, 51(1):5–30, 2017.

[67] Kishor Jothimurugan, Rajeev Alur, and Osbert Bastani. A composable specification language for reinforcement learning tasks. In *Advances in Neural Information Processing Systems*, pages 13041–13051, 2019.

[68] Xujie Si, Mukund Raghothaman, Kihong Heo, and Mayur Naik. Synthesizing datalog programs using numerical relaxation. In *IJCAI*, 2019.

[69] Jamin Naghmouchi, Daniele Paolo Scarpazza, and Mladen Berekovic. Small-ruleset regular expression matching on gpgpus: quantitative performance analysis and optimization. In *Proceedings of the 24th ACM International Conference on Supercomputing*, pages 337–348, 2010.

[70] Vu Le, Daniel Perelman, Oleksandr Polozov, Mohammad Raza, Abhishek Udupa, and Sumit Gulwani. Interactive program synthesis. *arXiv preprint arXiv:1703.03539*, 2017.

[71] Rishabh Singh and Sumit Gulwani. Predicting a correct program in programming by example. In *International Conference on Computer Aided Verification*, pages 398–414. Springer, 2015.

[72] Xi Ye, Qiaochu Chen, Isil Dillig, and Greg Durrett. Optimal neural program synthesis from multimodal specifications. *arXiv preprint arXiv:2010.01678*, 2020.

[73] Jacob Devlin, Jonathan Uesato, Surya Bhupatiraju, Rishabh Singh, Abdel-rahman Mohamed, and Pushmeet Kohli. Robustfill: Neural program learning under noisy i/o. In *ICML*, 2017.

[74] Woosuk Lee, Kihong Heo, Rajeev Alur, and Mayur Naik. Accelerating search-based program synthesis using learned probabilistic models. *ACM SIGPLAN Notices*, 53(4):436–449, 2018.

A Query Language

We describe our query language for matching object trajectories in videos. Our system first preprocesses the video using an object tracker to obtain trajectories, which are sequences $z = (x_0, x_1, ..., x_{n-1})$ of object states $x_i \in \mathcal{X}$. Then, a query Q in our language maps each trajectory z to a value $\mathbb{B} = \{0, 1\}$ indicating whether it matches z. Our language is based on regular expressions, except where the "characters" are predicates. In particular, it is related to the Kleene algebra with tests [21], but extends it in two key ways: (i) predicates can be over arbitrary subsequences of z rather than a single object state x, and (ii) it includes the conjunction operator over expressions. Below, we describe our language in more detail.

Trajectories. We begin by describing the input to a query in our language, which is a representation of the trajectory of a single object in a video. We describe how these trajectories are constructed in Section C. Consider a space of *object states* $\mathcal{X}$, which corresponds to a single object detection in a single video frame—e.g., $x \in \mathcal{X} \subseteq \mathbb{R}^6$ might encode the 2D position, velocity, and acceleration of x in image coordinates. Then, a *trajectory* $z \in \mathcal{Z} = \mathcal{X}^*$ is a sequence $z = (x_0, x_1, ..., x_{n-1})$ of object states of length $|z| = n$. We use the notation $z_{i:j} = (z_i, z_{i+1}, ..., z_{j-1})$ to denote a subtrajectory of z.

Predicates. We assume a set of predicates Φ is given, where each predicate $\varphi \in \Phi$ matches trajectories $z \in \mathcal{Z}$; we use $\mathsf{sat}_\varphi(z) \in \mathbb{B} = \{0, 1\}$ to indicate that φ matches z. As discussed below, queries in our language compose these predicates to match more complex patterns.

Next, predicates in our language may have real-valued parameters that must be specified. We denote such a predicate φ with parameter $\theta \in \mathbb{R}$ by φ_θ. To enable our synthesis algorithm to efficiently synthesize these real-valued parameters, we leverage monotonic structure that is found in all such predicates we have used in our queries. In particular, we assume that the semantics of these predicates have the form

$$[\![\varphi_\theta]\!](z) \coloneqq \mathbb{1}(\iota_\varphi(z) \geq \theta),$$

where $\iota_\varphi : \mathcal{Z} \to \mathbb{R}$ is a scoring function. For example, for the predicate $\mathsf{MinAvgAccel}_\theta$, we have

$$\iota_{\mathsf{MinAvgAccel}}(z) = \frac{1}{n} \sum_{i=0}^{n} a_i,$$

$$
\begin{aligned}
[\![\varphi]\!](z) &\coloneqq \mathsf{sat}_\varphi(z) \\
[\![Q_1 \vee Q_2]\!](z) &\coloneqq [\![Q_1]\!](z) \vee [\![Q_2]\!](z) \\
[\![Q_1 \wedge Q_2]\!](z) &\coloneqq [\![Q_1]\!](z) \wedge [\![Q_2]\!](z) \\
[\![Q_1 \,;\, Q_2]\!](z) &\coloneqq \bigvee_{k=0}^{n} [\![Q_1]\!](z_{0:k}) \wedge [\![Q_2]\!](z_{k:n}) \\
[\![Q^*]\!](z) &\coloneqq \bigvee_{k=0}^{n} [\![Q^k]\!](z)
\end{aligned}
$$

Figure 3: Boolean semantics of our query language; $z \in \mathcal{Z}$ is a trajectory of length n and $\varphi \in \Phi$ are predicates. We omit the semantics for Q^k since it can be expressed in terms of sequencing.

where a_i is the acceleration on frame i of trajectory z. That is, the scoring function is the average acceleration for the trajectory z; thus, $\iota_{\mathsf{MinAvgAccel}}(z) \geq \theta$ says that the average acceleration is at least θ. We describe the predicates included in our system in Section C; a user can easily extend them with additional predicates.

Multi-frame predicates. A key feature of our language is that it includes predicates that operate over multiple frames. In particular, whereas a predicate such as $\mathsf{InLane1}$ matches exactly one video frame (i.e., it checks whether the car is in lane 1 in that frame), a predicate such as $\mathsf{MinAvgAccel}_\theta$ checks whether the car's average acceleration is above some minimum value θ across some sequence of frames. In particular, this predicate is evaluated across sequences of frames rather than individual frames. For instance, the query

$$\langle \mathsf{InLane1} \rangle^{10} \wedge \langle \mathsf{MinAvgAccel}_\theta \rangle$$

says that the car is in lane 1 for 10 frames and has an average acceleration of at least θ across those same ten frames. In contrast, if $\mathsf{MinAvgAccel}_\theta$ was evaluated frame by frame, we could consider writing the query

$$\langle \mathsf{InLane1} \rangle^{10} \wedge \langle \mathsf{MinAvgAccel}_\theta \rangle^{10}.$$

However, $\langle \mathsf{MinAvgAccel}_\theta \rangle^{10}$ says that *each* of the ten frames has minimum average acceleration of θ, which is equivalent to saying that the acceleration is at least θ in each frame individually. Due to random noise in object positions predicted by the tracker, this likely would not yield expected results when the video framerate is high relative to object speeds. Thus, our system supports predicates that are evaluated over sequences instead of individual frames.

Syntax. The syntax of our language is

$$Q ::= \varphi \mid Q \,;\, Q \mid Q^k \mid Q^* \mid Q \vee Q \mid Q \wedge Q,$$

where $Q^k = Q; Q; ...; Q$ is sequencing iterated k times. The base case is a single predicate φ. These predicates can be combined using the usual regular expression operators—i.e., sequencing ($Q \,;\, Q$), iteration (finite Q^k and Kleene star Q^*), and disjunction ($Q \vee Q$). It also includes conjunction ($Q \wedge Q$), which is needed to express queries that involve trajectories of multiple objects.

Semantics. The (Boolean) semantics of queries have type

$$[\![\cdot]\!] : \mathcal{Q} \to \mathcal{Z} \to \mathbb{B}$$

where $\mathcal{Q}$ is the set of all queries in our language, $\mathcal{Z}$ is the set of trajectories, and $\mathbb{B} = \{0, 1\}$. In particular, $[\![Q]\!](z) \in \mathbb{B}$ indicates whether the query Q matches trajectory z. The semantics are defined in Figure 3. The base case of a single predicate φ checks whether φ matches z; the logical operators are straightforward; sequencing $Q_1 \,;\, Q_2$ checks if z can be split into $z = z_{0:k} z_{k:n}$ in a way that Q_1 matches $z_{0:k}$ and Q_2 matches $z_{k:n}$; and Kleene star is a disjunction over finite iterations of Q. We omit finite iteration, which is simply a disjunction over sequencing; for $k = 0$, $Q^k = \varepsilon$ is the empty string, which we take to be the predicate $\varepsilon \in \Phi$ defined by $\mathsf{sat}_\varepsilon(z) = \mathbb{1}(|z| = 0)$ (where $\mathbb{1}$ is the indicator function mapping predicates φ to binary values $\mathbb{1}(\varphi) \in \mathbb{B}$).

Computation. One way to evaluate queries in our language is using dynamic programming. In this approach, for every pair of indices $i, j \in \{0, 1, ..., n\}$ (where $n = |z|$) such that $i \leq j$ and every subexpression Q' of Q, we compute $v_{Q',i,j} = [\![Q']\!](z_{i:j})$. Then, we can compute $v_{Q',i,j}$ as a function of $v_{Q'',i',j'}$, where Q'' is a subexpression of Q' and $i \leq i' \leq j' \leq j$. This algorithm has time complexity $O(m \cdot n^3)$, where $m = |Q|$ is the number of subexpressions in Q, since there are $m \cdot n^2$ values $v_{Q',i,j}$ and computing each takes time $O(n)$.

However, our implementation instead uses the matrix semantics in Section B, which allows us to efficiently execute them using fast linear algebra libraries. These semantics implicitly implement the dynamic programming algorithm described above. In particular, they equal the Boolean semantics $[\![Q]\!]$ if we use $\iota_\varphi = \mathsf{sat}_\varphi$ for all $\varphi \in \Phi$. The time complexity of our implementation is also $O(m \cdot n^3)$, though improvements may be possible using fast matrix multiplication [25].

B Synthesis Algorithm

We describe our algorithm for synthesizing queries consistent with a given set of examples. It performs an enumerative search over the space of possible queries, represented as a grammar [26], using overapproximations to prune parts of the search space (represented by partial queries—i.e., queries with holes). Also, it uses active learning to have the user provide additional labels to further prune incorrect programs. A key challenge is managing the search space, especially over real-valued parameters. Our algorithm assumes that the dependence on real-valued parameters is monotonic; then, it computes upper and lower bounds on the parameter values using a quantitative variant of the query semantics. We begin by formulating the synthesis problem (Section B.1), and then provide an overview of our algorithm (Section B.2). Then, we describe our strategies for pruning partial queries during enumeration (Section B.3), and for using quantitative semantics to prune the search space of real-valued parameters (Sections B.4 & B.5). Next, we describe two strategies for speeding up evaluation of our quantitative semantics (Sections B.6 & B.7); these also apply to our Boolean semantics. Finally, we summarize our theoretical guarantees (Section B.8).

B.1 Problem Formulation

Partial queries. A *partial query* is an element of the grammar

$$Q ::= ?? \mid \varphi_{??} \mid \varphi \mid Q\,;\,Q \mid Q^k \mid Q^* \mid Q \vee Q \mid Q \wedge Q.$$

Note that there are two kinds of holes: (i) a *predicate hole* $h = ??$ that can be filled by a sub-query Q, and (ii) a *parameter hole* $h = \varphi_{??}$ that can be filled by a real value $\theta_h \in \mathbb{R}$. We denote the set of occurrences of predicate holes of Q by $\mathcal{H}_\varphi(Q)$, the set of parameter holes by $\mathcal{H}_\theta(Q)$, and the set of all holes by $\mathcal{H}(Q) = \mathcal{H}_\varphi(Q) \cup \mathcal{H}_\theta(Q)$. A partial query Q is a *sketch* (denoted $Q \in \mathcal{Q}_{\text{sketch}}$) [27] if $\mathcal{H}_\varphi(Q) = \varnothing$, and is *complete* (denoted $Q \in \bar{\mathcal{Q}}$) if $\mathcal{H}(Q) = \varnothing$; note that $\bar{\mathcal{Q}} \subseteq \mathcal{Q}_{\text{sketch}} \subseteq \mathcal{Q}$.

For example, consider the query $Q = ??1\,;\,??2$; here, we label each hole $h = ??i$ with an identifier $i \in \mathbb{N}$ so that we can distinguish them. This query has two predicate holes, so $\mathcal{H}_\varphi(Q) = \{??1, ??2\}$ and $\mathcal{H}_\theta(Q) = \varnothing$. Alternatively, consider the query

$$Q = \langle \mathsf{MinAvgAccel}_{??1} \rangle \wedge \langle \mathsf{MinLength}_{??2} \rangle,$$

which says that the trajectory is at least $??2$ frames and that the car has a minimum average acceleration of $??1$ across the frames in this trajectory. This query only has parameter holes, so it is a sketch. In particular, we have $\mathcal{H}_\varphi(Q) = \varnothing$, and $\mathcal{H}_\theta(Q) = \{??1, ??2\}$.

Refinements and completions. Given $Q \in \mathcal{Q}$, predicate hole $h \in \mathcal{H}_\varphi(Q)$, and production $R = Q \to f(Q_1, ..., Q_k)$, where f is an operator (i.e., a predicate $f() = \varphi_{??}$ or $f() = \varphi$, sequencing $f(Q_1, Q_2) = Q_1\,;\,Q_2$, etc.), we can *fill* h with R (denoted $Q' = \mathsf{fill}(Q, h, R)$) by replacing h with $f(??1, ..., ??k)$, where each $??i$ is a new hole. Similarly, given a parameter hole $h \in \mathcal{H}_\theta(Q)$, and a value $\theta_h \in \mathbb{R}$, we can fill h with θ_h (denoted $Q' = \mathsf{fill}(Q, h, \theta_h)$) by replacing h with θ_h. We say Q' is a *child* of Q (denoted $Q \to Q'$) if $Q' = \mathsf{fill}(Q, h, v)$ for some $h \in \mathcal{H}(Q)$ and v (a production or a value, depending on h). We say Q' is a *refinement* of Q (denoted $Q \xrightarrow{*} Q'$) if there exists a sequence $Q \to Q_1 \to ... \to Q_k \to Q'$; if furthermore $Q' \in \bar{\mathcal{Q}}$, then we say *completion* of Q. For example, we have the following derivation:

$$??1 \to ??2\,;\,??3 \to \langle \mathsf{InLane1} \rangle\,;\,??3 \to ... \to \langle \mathsf{InLane1} \rangle\,;\,\langle \mathsf{True} \rangle^*\,;\,\langle \mathsf{InLane2} \rangle$$

Then, $\langle \mathsf{InLane1} \rangle\,;\, ??3$ is a child (and refinement) of $??2\,;\, ??3$ obtained by filling the left-most hole with production $Q \to \langle \mathsf{InLane1} \rangle$—i.e.,

$$\langle \mathsf{InLane1} \rangle\,;\, ??3 = \mathsf{fill}(??2\,;\, ??3, ??2, Q \to \langle \mathsf{InLane1} \rangle).$$

Furthermore, $\langle \mathsf{InLane1} \rangle\,;\, ??4\,;\, ??5$ is in turn a child of $\langle \mathsf{InLane1} \rangle\,;\, ??3$ obtained by filling the hole $??3$ with the production $Q \to Q\,;\, Q$—i.e.,

$$\langle \mathsf{InLane1} \rangle\,;\, ??4\,;\, ??5 = \mathsf{fill}(\langle \mathsf{InLane1} \rangle\,;\, ??3, ??3, Q \to Q\,;\, Q).$$

Finally, $\langle \mathsf{InLane1} \rangle\,;\, \langle \mathsf{True} \rangle^*\,;\, \langle \mathsf{InLane2} \rangle$ is obtained by three more steps: (i) filling $??4$ with the production $Q \to ??6^*$, (ii) filling $??6$ with the production $Q \to \langle \mathsf{true} \rangle$, and (iii) filling $??5$ with the production $Q \to \langle \mathsf{InLane2} \rangle$. The final query $\langle \mathsf{InLane1} \rangle\,;\, \langle \mathsf{True} \rangle^*\,;\, \langle \mathsf{InLane1} \rangle$ has no holes, so it is complete, and is also a completion of all partial queries in the derivation (i.e., $??1$, $??2\,;\, ??3$, etc.).

Parameters. For any predicate φ_θ with a parameter $\theta \in \mathbb{R}$, we assume it has semantics

$$[\![\varphi_\theta]\!](z) \coloneqq \mathbb{1}(\iota_\varphi(z) \geq \theta),$$

for some scoring function $\iota_\varphi : \mathcal{Z} \to \mathbb{R}$. In other words, the predicate φ becomes satisfied once the score $\iota_\varphi(z)$ is sufficiently large. For example, if $\varphi_\theta = \mathsf{MinAvgAccel}_\theta$, then $\iota_{\mathsf{MinAvgAccel}}(z)$ is average acceleration of the car in trajectory z. Predicates with multiple parameters can typically be expressed as a conjunction of such expressions, in which case our algorithm can still handle them.

We let $\theta \in \mathbb{R}^{|\mathcal{H}_\theta(Q)|}$ denote a choice of parameters for all holes $h \in \mathcal{H}_\theta(Q)$, and let $\theta_h \in \Theta_h$ denote the parameter corresponding to hole h, where $\Theta_h \subseteq \mathbb{R}$ is a finite set of possible parameter values (we assume it is finite so we can enumerate all possibilities). Given a query $Q \in \mathcal{Q}$ and $\theta \in \mathbb{R}^{|\mathcal{H}_\theta(Q)|}$, we let $Q_\theta \in \mathcal{Q}$ denote the query obtained by filling each parameter hole $h \in \mathcal{H}_\theta(Q)$ with θ_h. Note that if $Q \in \mathcal{Q}_{\text{sketch}}$, then $Q_\theta \in \bar{\mathcal{Q}}$ is complete. For example, consider the sketch

$$Q = \langle \mathsf{MinAvgAccel}_{??1} \rangle \wedge \langle \mathsf{MinLength}_{??2} \rangle$$

shown above. This query has two holes, so its parameters have the form $\theta = (3.2, 5.0) \in \mathbb{R}^2$, where $\theta_{??1} = 3.2$ is used to fill hole $??1$ and $\theta_{??2} = 5.0$ is used to fill $??2$. In particular, we have

$$Q_\theta = \langle \mathsf{MinAvgAccel}_{3.2} \rangle \wedge \langle \mathsf{MinLength}_{5.0} \rangle.$$

This query says that the average acceleration should be at least 3.2, and the length of the trajectory should be at least 5.0.

Query synthesis problem. Given examples $W \subseteq \mathcal{W} = \mathcal{Z} \times \mathbb{B}$, where $\mathbb{B} = \{0, 1\}$, our goal is to compute a query $Q \in \bar{\mathcal{Q}}$ that correctly labels these examples—i.e., Q satisfies

$$\psi_W(Q) \coloneqq \bigwedge_{(z,y) \in W} ([\![Q]\!](z) = y). \tag{1}$$

Thus, $\psi_W(Q)$ indicates whether Q is consistent with the labeled examples W. Then, we can enumerate over the possibilities, though the search space is combinatorial in the number of parameter holes. Our goal is to devise a synthesis algorithm that is sound and complete—i.e., it finds a query that satisfies $\psi_W(Q) = \mathsf{true}$ if and only if one exists.

B.2 Synthesis Algorithm Overview

At a high level, our algorithm proceeds in two steps: (i) it computes a list ℓ^* of all queries Q_θ that are consistent with the current examples W (summarized in Algorithm 1), and (ii) it then uses active learning to select among the queries in ℓ^* (summarized in Algorithm 2).

Syntax-guided search. First, Algorithm 1 enumerates partial queries until it finds all complete queries $Q_\theta \in \bar{\mathcal{Q}}$ that are consistent with the given examples W. In particular, it keeps a worklist ℓ_0 of current partial programs, initialized with $Q = ??$. Then, at each step until ℓ_0 is empty, it processes a single program Q from ℓ_0. It uses overapproximations of the query semantics to check whether Q is consistent with the examples W. If so, it considers two cases: (i) if Q is a sketch, it computes all parameters θ for Q that are consistent with W and adds Q_θ to the list of consistent programs ℓ^*, or (ii) otherwise, it adds all the children of Q to ℓ_0. In the latter case, the children of Q (denoted $\mathsf{Children}(Q)$) are obtained by taking each predicate hole in Q and filling it using each

Algorithm 2 Actively query the user $\mathcal{O}$ on unlabeled examples $z \in Z$ to eliminate incorrect queries.

procedure ACTIVELEARNING(ℓ^*, Z)
  while true do
    $z^* \leftarrow \arg\max_{z \in Z} J(z; \ell^*)$
    if $J(z^*) = 0$ **then**
      return ℓ^*
    end if
    $y^* \leftarrow \mathcal{O}(z^*)$
    $\ell^* \leftarrow \{Q_\theta \in \ell^* \mid \psi_{(z^*, y^*)}(Q_\theta)\}$
  end while
end procedure

applicable production in our grammar. In the sections below, we describe how our algorithm uses overapproximations to perform pruning, and how it computes the set of consistent parameters θ.

Active learning. Next, Algorithm 2 actively asks the user to label additional trajectories $z \in Z$ from a set of unlabeled trajectories $Z \subseteq \mathcal{Z}$ to disambiguate among the queries $Q_\theta \in \ell^*$ returned by Algorithm 1. It does so using a standard algorithm [23]; we briefly describe this algorithm here. In particular, at each step, it selects the trajectory z^* that maximizes

$$J(z) \coloneqq \sum_{y \in \{0,1\}} J_y(z)(1 - J_y(z)), \qquad \text{where} \qquad J_y(z) \coloneqq \frac{1}{|\ell^*|} \sum_{Q_\theta \in \ell^*} \mathbb{1}(\psi_{(z,y)}(Q_\theta))$$

is the fraction of queries remaining in ℓ^* that are also consistent with (z, y). Intuitively, assuming the true query is a uniformly random query in ℓ^*, then $J_y(z)$ is the probability that y is the correct label for z. Then, $J(z)$ is the expected fraction of the search space that is pruned—i.e., with probability $J_y(z)$, the label for z is y, in which case we can prune the $1 - J_y(z)$ fraction of queries that are inconsistent with (z, y). Thus, our active learning algorithm selects the trajectory z^* that prunes the maximum number of queries in expectation over the label y. This algorithm has been proven to be near-optimal in terms of sample complexity [23]. Finally, our algorithm queries the user $\mathcal{O}$ to obtain the actual label $y^* = \mathcal{O}(z^*)$, and prunes the queries that are inconsistent with (z^*, y^*). It continues until all remaining queries have equal labels on Z, which is equivalent to $J(z^*) = 0$, since in this case, for all $z \in Z$ and $y \in \{0, 1\}$, we have either $J_y(z) = 0$ or $1 - J_y(z) = 0$.

B.3 Pruning Partial Queries

We use an overapproximation $[\![Q]\!]^\#$ of the semantics for partial queries $Q \in \mathcal{Q}$ to prune our search space. That is, it overapproximates the possible outputs of completions of Q on an input $z \in \mathcal{Z}$—i.e., $[\![\bar{Q}]\!](z) \in [\![Q]\!]^\#$ for every completion $\bar{Q}$ of Q. In particular, we define

$$[\![Q]\!]^\# \coloneqq \{[\![Q^+]\!](z), [\![Q^-]\!](z)\}.$$

Here, Q^+ is constructed by filling each predicate hole $h \in \mathcal{H}_\varphi(Q)$ in Q with $\langle \mathsf{true} \rangle^*$, and each parameter hole $h \in \mathcal{H}_\theta(Q)$ in Q with $-\infty$. Similarly, Q^- is constructed by filling each hole in Q with $\langle \mathsf{false} \rangle$, and each parameter hole $h \in \mathcal{H}_\theta(Q)$ in Q with ∞.

Lemma 1. *For any completion $\bar{Q}$ of Q, we have $[\![\bar{Q}]\!](z) \in [\![Q]\!]^\#(z)$.*

We give a proof in Section F.1. We can use the our overapproximation of the semantics to prune the search space—for a partial query $Q \in \mathcal{Q}$ and an example $(z, y) \in W$, if we have $y \notin [\![Q]\!]^\#(z)$, then we can prune Q and all of its descendants from the search space. More precisely, defining

$$\psi_W^\#(Q) \coloneqq \bigwedge_{(z,y) \in W} y \in [\![Q]\!]^\#(z),$$

we prune Q if $\psi_W^\#(Q) = \mathsf{false}$. For example, for $Q = \langle \mathsf{InLane1} \rangle ; \left\langle \mathsf{MinAvgAccel}_{??} \right\rangle ; \langle \mathsf{true} \rangle^*$, we have

$$Q^+ = \langle \mathsf{InLane1} \rangle ; \left\langle \mathsf{MinAvgAccel}_{-\infty} \right\rangle ; \langle \mathsf{true} \rangle^*, \quad Q^- = \langle \mathsf{InLane1} \rangle ; \left\langle \mathsf{MinAvgAccel}_{\infty} \right\rangle ; \langle \mathsf{false} \rangle.$$

Consider any completion $\bar{Q} = \langle \mathsf{InLane1} \rangle ; \left\langle \mathsf{MinAvgAccel}_{\theta_h} \right\rangle ; Q'$ of Q, where $Q' \in \mathcal{Q}$ and $\theta_h \in \mathbb{R}$. Now, it is easy to see that $[\![Q^+]\!](z) = 0 \Rightarrow [\![\bar{Q}]\!](z) = 0$—i.e.,

$$\left[\!\!\left[\langle \mathsf{InLane1} \rangle ; \left\langle \mathsf{MinAvgAccel}_{-\infty} \right\rangle ; \langle \mathsf{true} \rangle^* \right]\!\!\right](z) = 0 \Rightarrow \left[\!\!\left[\langle \mathsf{InLane1} \rangle ; \left\langle \mathsf{MinAvgAccel}_{\theta_h} \right\rangle ; Q' \right]\!\!\right](z) = 0.$$

In other words, if even the optimistic choice $\theta_h = -\infty$ and $Q' = \langle \mathsf{true} \rangle^*$ evaluates to 0 on z, then every completion $\bar{Q}$ of Q also evaluates to 0 on z. Thus, if $y = 1$, every completion $\bar{Q}$ of Q is inconsistent with the example (z, y), so we can safely prune Q. In this example, $[\![Q^+]\!](z) = 0$ if the first element x_0 of z does not match $\langle \mathsf{InLane1} \rangle$, so we can prune Q if $y = 1$ and x_0 does not match $\langle \mathsf{InLane1} \rangle$. Alternatively, we also have $[\![Q^-]\!](z) = 1 \Rightarrow [\![\bar{Q}]\!](z) = 1$—i.e.,

$$\left[\!\!\left[\langle \mathsf{InLane1} \rangle ; \left\langle \mathsf{MinAvgAccel}_{\infty} \right\rangle ; \langle \mathsf{false} \rangle \right]\!\!\right](z) = 1 \Rightarrow \left[\!\!\left[\langle \mathsf{InLane1} \rangle ; \left\langle \mathsf{MinAvgAccel}_{\theta_h} \right\rangle ; Q' \right]\!\!\right](z) = 1.$$

In other words, if even the pessimistic choice $\theta_h = \infty$ and $Q' = \langle \mathsf{false} \rangle$ evaluates to 1 on z, then every completion $\bar{Q}$ of Q evaluates to 1. Thus, if $y = 0$, every completion $\bar{Q}$ of Q is inconsistent with the example (z, y), so we can safely prune Q. In this example, we can never have $[\![Q^-]\!](z) = 1$ due to the $\langle \mathsf{false} \rangle$ predicate, so there is no example (z, y) with $y = 0$ that would enable us to prune Q. In general, pruning is possible if the hole occurs within a disjunction or Kleene star operator.

B.4 Pruning Parameter Values

Once our algorithm has identified a sketch $Q \in \mathcal{Q}_{\text{sketch}}$, it needs to fill the parameter holes in Q to obtain a complete query $Q_\theta \in \bar{\mathcal{Q}}$. A naïve strategy would be to perform an enumerative search over possible parameter values $\theta_h \in \Theta_h$ to fill each parameter hole $h \in \mathcal{H}_\theta(Q)$, where $\Theta_h \subseteq \mathbb{R}$ is the (finite) set of possible parameter values for hole h, using the above pruning strategy to prune the search space over parameter values.

However, we can perform pruning much more efficiently by leveraging monotonicity of parameters. In particular, letting $\varphi_{??}$ be the hole h, recall that if we fill h with θ_h, then the semantics of the resulting predicate φ_{θ_h} are $[\![\varphi_{\theta_h}]\!](z) = \mathbb{1}(\iota_\varphi(z) \geq \theta_h)$. Thus, given an example (z, y) where $y = 1$, suppose that θ_h cannot be used to fill h even under optimistic assumptions about the remaining parameters—i.e., letting $Q_{\theta_h} = \mathsf{fill}(Q, h, \theta_h)$, then $[\![Q^+_{\theta_h}]\!](z) = 0$. Then, we know that any parameters $\theta'_h \geq \theta_h$ also cannot be used to fill h, since increasing θ_h can only make φ_{θ_h} become false. Conversely, if z is a negative example and $[\![Q^-_{\theta_h}]\!](z) = 1$, then we know that $\theta'_h \leq \theta_h$ cannot be used to fill h, since decreasing θ_h can only make φ_{θ_h} become true. For example, consider the query

$$Q = \left\langle \mathsf{Dist}_{??}(A, B) \right\rangle^*,$$

which says that at every point in the trajectory, the negative distance between cars A and B is at least θ_h (i.e., the cars are close together), where θ_h is used to fill $h = ??$; equivalently, their distance is at most $-\theta_h$. Here, the parameters $\theta = (\theta_h)$ are one-dimensional. Suppose we are given an input-output example (z, y), where $z = (x_0, x_1, x_2) = (1.2, 2.0, 0.8)$ and x_i encodes the distance between A and B on step i. Suppose that $y = 1$ and $\theta_h = -1.0$; then, we have $[\![Q_{-1.0}]\!](z) = 0$, so we can prune Q_θ for all θ such that $\theta_h \geq -1.0$. Conversely, suppose that $\theta_h = -3.0$ and $y = 0$; then, we have $[\![Q_{-3.0}]\!](z) = 1$, so we can prune Q_θ for all θ such that $\theta_h \leq -3.0$. In general, the parameter values θ_h that are consistent in a given example $(z, y) \in W$ must be contained in the interval

$$\Theta^{\#}_h(Q, z, y) \coloneqq \begin{cases} (-\infty, \theta^+_h] & \text{if } y = 1 \\ (\theta^-_h, \infty) & \text{if } y = 0, \end{cases}$$

where

$$\theta^+_h \coloneqq \inf\{\theta_h \in \Theta_h \mid [\![(Q_{\theta_h})^+]\!](z) = 1\} \qquad \text{and} \qquad \theta^-_h \coloneqq \sup\{\theta_h \in \Theta_h \mid [\![(Q_{\theta_h})^-]\!](z) = 0\},$$

and where $(Q_{\theta_h})^+$ (resp., $(Q_{\theta_h})^-$) is defined as above—i.e., by filling predicate holes with $\langle \mathsf{true} \rangle^*$ (resp., $\langle \mathsf{false} \rangle$) and parameter holes with $-\infty$ (resp., ∞). In other words, θ^+_h is the largest possible parameter value such that $[\![Q_{\theta^+_h}]\!](z) = 1$, even under optimistic assumptions about how the remaining holes are filled. Conversely, θ^-_h is the smallest possible parameter value for which $[\![Q_{\theta^-_h}]\!](z) = 0$. For our example $Q = \left\langle \mathsf{Dist}_{??1}(A, B) \right\rangle^*$ and $z = (1.2, 2.0, 0.8)$, we have

$$\Theta^{\#}_{??1}(Q, z, y) = \begin{cases} (-\infty, -2.0] & \text{if } y = 1 \\ (-2.0, \infty) & \text{if } y = 0. \end{cases}$$

$$
\begin{aligned}
[\![\varphi_{??}]\!]^q(z) &\coloneqq \iota_\varphi(z) \\
[\![\varphi]\!]^q(z) &\coloneqq \begin{cases} \infty & \text{if } \mathsf{sat}_\varphi(z) = 1 \\ -\infty & \text{if } \mathsf{sat}_\varphi(z) = 0. \end{cases} \\
[\![Q_1 \vee Q_2]\!]^q(z) &\coloneqq \max\{[\![Q_1]\!]^q(z), [\![Q_2]\!]^q(z)\} \\
[\![Q_1 \wedge Q_2]\!]^q(z) &\coloneqq \min\{[\![Q_1]\!]^q(z), [\![Q_2]\!]^q(z)\} \\
[\![Q_1 \,;\, Q_2]\!]^q(z) &\coloneqq \max_{k \in \{0,1,\ldots,n\}} \min\{[\![Q_1]\!]^q(z_{0:k}), [\![Q_2]\!]^q(z_{k:n})\} \\
[\![Q^*]\!]^q(z) &\coloneqq \max_{k \in \{0,1,\ldots,n\}} \{[\![Q^k]\!]^q(z)\}
\end{aligned}
$$

Figure 4: Quantitative semantics of our language; ι_φ is a quantitative variant of sat_φ, and n is the length of z.

Note that in this case, we have $\theta_h^+ = \theta_h^-$; this equality holds when the only hole in Q is the single parameter hole h we are trying to fill. In particular, $\theta_h^0 = \theta_h^- = \theta_h^+$ is the value of θ_h where the query transitions from $[\![Q_{\theta_h}]\!](z) = 1$ (for $\theta_h \leq \theta_h^0$) to $[\![Q_{\theta_h}]\!](z) = 0$ (for $\theta_h \geq \theta_h^0$). In the general case where Q has multiple holes (discussed below), this equality may not hold.

Next, given multiple input-output examples W, the set of valid parameters for all of W is

$$
\Theta_h^\#(Q, W) \coloneqq \bigcap_{(z,y) \in W} \Theta_h^\#(Q, z, y).
$$

The remaining challenge is computing $\Theta_h^\#(Q, W)$. Our algorithm computes θ_h^+ and θ_h^- compositionally; we give details in Section B.5. Finally, our algorithm adds all completions Q_θ of Q with parameters $\theta_h \in \Theta_h \cap \Theta_h^\#(Q, W)$ to ℓ^*.

Importantly, the interval $\Theta_h^\#(Q, W)$ is for a *single* parameter hole $h \in \mathcal{H}_\theta(Q)$ in isolation. When there are multiple parameter holes, there may be parameters $\theta \in \mathbb{R}^{|\mathcal{H}_\theta(Q)|}$ such that $\theta_h \in \Theta_h^\#(Q, W)$ for every hole $h \in \mathcal{H}_\theta(Q)$, but θ is still inconsistent with W. For example, consider the query

$$
Q = \langle \mathsf{Dist}(A, B)_{??1} \rangle^* \,;\, \langle \mathsf{Dist}(B, C)_{??2} \rangle^*,
$$

along with the length one trajectory $z = (x_0)$, where $x_0 = (1.0, 1.0) \in \mathbb{R}^2$ (here, the first component encodes the distance from A to B and the second component encodes the distance from B to C), and label $y = 1$. Then, suppose we fill $??2$ with $\theta_{??2} = -\infty$ to obtain $Q' = \langle \mathsf{Dist}(A, B)_{??1} \rangle^* \,;\, \langle \mathsf{Dist}(B, C)_{-\infty} \rangle^*$. Since $[\![(Q')_{\theta_{??1}}]\!](z) = 1$ for any $\theta_{??1} \in \mathbb{R}$, it follows that

$$
\Theta_{??1}^\#(Q, z, y) = (-\infty, \infty),
$$

i.e., $\theta_{??1}$ can take any value. Similarly, if we fill $??1$ with $\theta_{??1} = -\infty$, then $\theta_{??2}$ could take any value. Thus, we cannot prune the search space over $\theta_{??1}$ or $\theta_{??2}$ individually. However, not all parameter choices are valid—e.g., if $\theta = (0.0, 0.0)$, then

$$
Q_\theta = \langle \mathsf{Dist}(A, B)_{0.0} \rangle^* \,;\, \langle \mathsf{Dist}(B, C)_{0.0} \rangle^*.
$$

In this case, $[\![Q_\theta]\!](z) = 0$, so Q_θ is inconsistent with the given example. The issue is that the two predicates in Q can each match z by themselves when filled with $\theta_h = -\infty$. However, in Q_θ, neither hole is filled with $\theta_h = -\infty$, so this guarantee no longer holds. As a consequence, our algorithm still needs to enumerate over all possible queries $Q_\theta \in \ell^*$ to omit ones that are inconsistent with W. It does so on the last step, returning only queries with parameter values that are consistent with W.

B.5 Quantitative Semantics for Pruning Parameter Values

In this section, we give details on our algorithm for using a quantitative semantics of our queries to compute the pruned search space $\Theta_h^\#(Q, z, y)$ over parameters for hole h. In particular, given

a partial query $Q \in \mathcal{Q}$, input-output example $(z, y) \in \mathcal{W}$, and a hole $h \in \mathcal{H}_\theta(Q)$, our algorithm computes the space of parameter values $\Theta_h^\#(Q, z, y) \subseteq \mathbb{R}$ that can be used to fill the hole h in a way that is consistent with (z, y). Note that we can compute $\Theta_h^\#(Q, W)$ considering all possible $\theta_h \in \mathbb{R}$; then, we can intersect it with Θ_h to obtain the set of valid parameter values θ_h.

Now, suppose that the hole h is on predicate $\varphi_{??}$, and consider filling $??$ with θ_h. The resulting predicate φ_{θ_h} has Boolean semantics $[\![\varphi_{\theta_h}]\!] = \mathbb{1}(\iota_\varphi(z) \geq \theta_h)$, which is then used by the remainder of the query Q. First, suppose that $Q \in \mathcal{Q}_{\text{sketch}}$, and that Q has a single parameter hole h. Rather than fill in h with a single value θ_h, we define an alternative, quantitative semantics $[\![\cdot]\!]^q$, that leaves h as a hole, and lets $[\![\varphi_{??}]\!]^q = \iota_\varphi(z)$. Then, $[\![\cdot]\!]^q$ propagates these values in the remainder of the query Q by replacing disjunction ($\vee$) with maximum ($\max$), and conjunction ($\wedge$) with minimum ($\min$). In particular, these semantics have type

$$[\![\cdot]\!]^q : \mathcal{Q} \to \mathcal{Z} \to \mathbb{R},$$

and are defined in Figure 4. They satisfy the following key property:

Lemma 2. *Assume that $Q \in \mathcal{Q}_{sketch}$ has a single parameter hole $h \in \mathcal{H}_\theta(Q)$. Then, we have*

$$[\![Q_{\theta_h}]\!](z) = \mathbb{1}([\![Q]\!]^q(z) \geq \theta_h).$$

We give a proof in Section F.2. In other words, $[\![Q]\!]^q$ computes the threshold θ_h at which the Boolean semantics of Q changes from 1 to 0. Thus, we can take

$$\Theta_h^\#(Q, z, y) \coloneqq \begin{cases} (-\infty, [\![Q]\!]^q(z)] & \text{if } y = 1 \\ ([\![Q]\!]^q(z), \infty) & \text{if } y = 0. \end{cases}$$

For example, for the query $Q = \langle \mathsf{Dist}_{??1}(A, B) \rangle^*$ and the trajectory $z = (1.2, 2.0, 0.8)$, where each step x encodes the distance between A and B, we have

$$\begin{aligned} [\![Q]\!]^q(z) = \min_{i \in \{0,1,2,3\}} \left\{ [\![\langle \mathsf{Dist}_{??}(A, B) \rangle]\!]^q(z_{i:i+1}) \right\} &= \min_{i \in \{0,1,2,3\}} \{\iota_{\mathsf{Dist}(A,B)}(z_{i:i+1})\} \\ &= \min\{-1.2, -2.0, -0.8\} \\ &= -2.0, \end{aligned}$$

since $\langle \mathsf{Dist}_{??1}(A, B) \rangle$ is only applicable at a single time step $z_{i:i+1}$, so the maximum over k for the semantics of the Kleene star happens at $k = 3$. Thus, the set of valid parameters is

$$\Theta_{??1}^\#(Q, z, y) = \begin{cases} (-\infty, -2.0] & \text{if } y = 1 \\ (-2.0, \infty) & \text{if } y = 0. \end{cases}$$

In general, Q may have additional parameter holes or predicate holes. In this case, we overapproximate the holes $h' \neq h$ similar to before. In particular, we optimistically fill the parameter holes $h' \in \mathcal{H}_\theta(Q)$ such that $h' \neq h$ with $-\infty$ and the predicate holes with $\langle \mathsf{true} \rangle^*$ to obtain $Q^{h,+}$. In addition, we pessimistically fill parameter holes $h' \neq h$ with ∞ and predicate holes with $\langle \mathsf{false} \rangle$ to obtain $Q^{h,-}$. Then, we have

$$\Theta_h^\#(Q, z, y) \coloneqq \begin{cases} (-\infty, [\![Q^{h,+}]\!]^q(z)] & \text{if } y = 1 \\ ([\![Q^{h,-}]\!]^q(z), \infty) & \text{if } y = 0. \end{cases}$$

B.6 Matrix Semantics

Next, we describe how we can efficiently compute $[\![\cdot]\!]^q$ using matrix operations. A standard approach is to use dynamic programming, but matrices allow us to use existing fast linear algebra packages to compute $[\![\cdot]\!]^q$. In particular, our matrix semantics $[\![\cdot]\!]^M$ has type

$$[\![\cdot]\!]^M : \mathcal{Q} \to \mathcal{Z} \to \bigcup_{n \in \mathbb{N}} \mathbb{R}^{(n+1)\times(n+1)}.$$

In other words, $[\![Q]\!]^M(z)$ maps a trajectory z of length n to an $(n+1) \times (n+1)$ matrix with entries in $\mathbb{R}$. We use $i, j \in \mathcal{I}_n \coloneqq \{0, 1, ..., n\}$ to index matrices $M \in \mathbb{R}^{(n+1)\times(n+1)}$; as discussed below, these correspond to indices of z. The matrix semantics are defined in Figure 5. The base case is the

$$[\![\varphi_{??}]\!]^M(z) \coloneqq \begin{bmatrix} \iota_\varphi(z_{0:0}) & \iota_\varphi(z_{0:1}) & \cdots & \iota_\varphi(z_{0:n}) \\ -\infty & \iota_\varphi(z_{1:1}) & \cdots & \iota_\varphi(z_{1:n}) \\ \vdots & \vdots & \ddots & \vdots \\ -\infty & -\infty & \cdots & \iota_\varphi(z_{n:n}) \end{bmatrix}$$

$$[\![Q_1 \vee Q_2]\!]^M(z) \coloneqq \max\{[\![Q_1]\!]^M(z), [\![Q_2]\!]^M(z)\}$$

$$[\![Q_1 \wedge Q_2]\!]^M(z) \coloneqq \min\{[\![Q_1]\!]^M(z), [\![Q_2]\!]^M(z)\}$$

$$[\![Q_1 \,;\, Q_2]\!]^M(z) \coloneqq [\![Q_1]\!]^M(z) \cdot [\![Q_2]\!]^M(z)$$

Figure 5: The matrix semantics of our language; $z \in \mathcal{Z}$ is a trajectory of length n, $\max$ and $\min$ are taken elementwise, and $\cdot$ is the matrix product where addition is replaced with $\max$ and multiplication with $\min$. We omit the case $[\![\varphi_\theta]\!]^M(z)$, which is the same as $[\![\varphi_{??}]\!]^M(z)$, except $\iota_\varphi(z_{i:j})$ is replaced with $[\![\varphi]\!]^q(z_{i:j})$.

semantics of predicate $\varphi_{??}$ with holes (and predicates φ without holes); in this case, it constructs the matrix $M = [\![\varphi_{??}]\!]^M(z)$, where $M_{i,j} = \iota_\varphi(z_{i:j})$ for $i \leq j$ (i.e., how well φ matches $z_{i:j}$), and $M_{i,j} = -\infty$ for $i > j$. We use $-\infty$ in the bottom-left of $[\![\varphi]\!]^M(z)$ since these values ensure the semantics are consistent with sequencing. Next, disjunction is elementwise maximum—i.e.,

$$[\![Q_1 \vee Q_2]\!]^M(z)_{i,j} = \max\{[\![Q_1]\!]^M(z)_{i,j}, [\![Q_2]\!]^M(z)_{i,j}\}$$

for all $i, j \in \{0, 1, ..., n\}$. Similarly, conjunction is elementwise minimum—i.e.,

$$[\![Q_1 \wedge Q_2]\!]^M(z)_{i,j} = \min\{[\![Q_1]\!]^M(z)_{i,j}, [\![Q_2]\!]^M(z)_{i,j}\}$$

for all $i, j \in \{0, 1, ..., n\}$. Finally, sequencing is matrix multiplication, except with addition replaced by maximum and multiplication by minimum—i.e.,

$$[\![Q_1 \,;\, Q_2]\!]^M(z)_{i,j} = \max_{k \in \{i,...,j\}} \min\{[\![Q_1]\!]^M(z)_{i,k}, [\![Q_2]\!]^M(z)_{k,j}\}.$$

Then, we have the following correspondence between $[\![\cdot]\!]^M$ and $[\![\cdot]\!]^q$:

Lemma 3. *For any $z \in \mathcal{Z}$ of length n, and any $i, j \in \{0, 1, ..., n\}$ such that $i \leq j$, we have*

$$[\![Q]\!]^M(z)_{i,j} = [\![Q]\!]^q(z_{i:j}),$$

where $[\![Q]\!]^M(z)_{i,j}$ denotes the (i, j) entry of the matrix $[\![Q]\!]^M(z)$.

We give a proof in Section F.3. In other words, $[\![Q]\!]^M$ simultaneously computes the quantitative semantics of Q for every subsequence $z_{i:j}$ of z. We have the following straightforward consequence:

Theorem 1. *We have $[\![Q]\!]^q(z) = [\![Q]\!]^M(z)_{0,n}$.*

B.7 Sparse Matrix Semantics

Finally, we describe an optimization that allows us to reduce the size of the matrices involved in computing $[\![Q]\!]^M(z)$ for a given trajectory z. One caveat is that this optimization involves an expensive preprocessing step for each trajectory z. In our synthesis algorithm, this cost is amortized across a many queries since we are evaluating many queries on each training example $(z, y) \in W$.

Let z be a trajectory of length n, and $\varphi_{??}$ be a predicate with a hole, so $[\![\varphi_{??}]\!]^M(z) \in \mathbb{R}^{(n+1)\times(n+1)}$. Consider two indices $i, i' \in \{0, 1, ..., n\}$. Intuitively, suppose the rows i, i' and columns i, i' of $[\![\varphi_{??}]\!]^M(z)$ are equal across all $\varphi_{??}$ (and similarly for all φ); then, the operations performed on these matrices in Figure 5 preserve this equality (this fact can straightforwardly be proven by structural induction). Thus, for any Q, the rows i, i' and columns i, i' of $[\![Q]\!]^M(z)$ are equal. The interesting case is $Q = Q_1 \,;\, Q_2$, which corresponds to matrix multiplication. To illustrate, consider regular matrix multiplication (i.e., with addition and multiplication instead of minimum and maximum):

$$\begin{bmatrix} 1 & 1 & 2 \\ 1 & 1 & 2 \\ 3 & 3 & 4 \end{bmatrix} \cdot \begin{bmatrix} 2 & 2 & 1 \\ 2 & 2 & 1 \\ 3 & 3 & 5 \end{bmatrix} = \begin{bmatrix} 10 & 10 & 12 \\ 10 & 10 & 12 \\ 24 & 24 & 26 \end{bmatrix}.$$

$$[\![\varphi_{??}]\!]^R_{i,i'}(z) \coloneqq \bigwedge_{j=0}^{n} [\![\varphi_{??}]\!]^M_{i,j}(z) \geq [\![\varphi_{??}]\!]^M_{i',j}(z) \qquad [\![\varphi_{??}]\!]^C_{j,j'}(z) \coloneqq \bigwedge_{i=0}^{n} [\![\varphi_{??}]\!]^M_{i,j}(z) \geq [\![\varphi_{??}]\!]^M_{i,j'}(z)$$

$$[\![Q_1 \,;\, Q_2]\!]^R(z) \coloneqq [\![Q_1]\!]^R(z) \qquad [\![Q_1 \,;\, Q_2]\!]^C(z) \coloneqq [\![Q_2]\!]^C(z)$$

$$[\![Q_1 \vee Q_2]\!]^R(z) \coloneqq [\![Q_1]\!]^R(z) \wedge [\![Q_2]\!]^R(z) \qquad [\![Q_1 \vee Q_2]\!]^C(z) \coloneqq [\![Q_1]\!]^C(z) \wedge [\![Q_2]\!]^C(z)$$

$$[\![Q_1 \wedge Q_2]\!]^R(z) \coloneqq [\![Q_1]\!]^R(z) \wedge [\![Q_2]\!]^R(z) \qquad [\![Q_1 \wedge Q_2]\!]^C(z) \coloneqq [\![Q_1]\!]^C(z) \wedge [\![Q_2]\!]^C(z)$$

Figure 6: Semantics for computing row and column domination relationships. The $\wedge$ operator is applied elementwise. The semantics for φ are identical to those for $\varphi_{??}$.

In this example, the condition holds for the input matrices for $i = 1$ and $i' = 2$, and it also holds for the output. While we have used regular matrix multiplication, it is easy to check that this property is preserved by matrix multiplication with minimum and maximum as well.

More generally, we can obtain similar results if instead of being identical, the entries in row i are larger than the corresponding entries in row i', and similarly for columns i, i'.[1]

Definition 2. Given $A \in \mathbb{R}^{(n+1)\times(n+1)}$ and $i, i', j, j' \in \mathcal{I}_n = \{0, 1, ..., n\}$ such that $i \neq i'$, we say that i *A-row dominates* i' (denoted $i \succeq^R_A i'$) if $A_{i,j} \geq A_{i',j}$ for all $j \in \mathcal{I}_n \coloneqq \{0, 1, ..., n\}$, and we say j *A-column dominates* j' (denoted $j \succeq^C_A j'$) if $A_{i,j} \geq A_{i,j'}$ for all $i \in \mathcal{I}_n$.

The operators in our language preserve certain domination relationships; which ones are preserved depends on the operator. In fact, we can compute a conservative underapproximation of the row and column domination relationships satisfied by $[\![Q]\!]^M(z)$ as a function of the domination relationships of $[\![\varphi_{??}]\!]^M(z)$ and $[\![\varphi]\!]^M(z)$. In particular, we define the semantics

$$[\![\cdot]\!]^R, [\![\cdot]\!]^C : \mathcal{Q} \to \mathcal{Z} \to \mathbb{B}^{(n+1)\times(n+1)}$$

where $[\![Q]\!]^R(z) \in \mathbb{B}^{(n+1)\times(n+1)}$ is a binary matrix with the intent that

$$\left([\![Q]\!]^R(z)_{i,i'} = 1\right) \Rightarrow \left(i \succeq^R_{[\![Q]\!]^M(z)} i'\right),$$

i.e., $[\![Q]\!]^R_{i,i'}(z) = 1$ implies that row i dominates row i' in $[\![Q]\!]^M(z)$, and similarly for $[\![Q]\!]^C(z)$. These semantics are defined in Figure 6. For conjunctions and disjunctions, the row (resp., column) relationships are the intersections of the row (resp., column) relationships of subexpressions. For sequencing $Q = Q_1 \,;\, Q_2$, the row domination relationships are those of Q_1, and the column relationships are those of Q_2. For example, we might have

$$\underbrace{\begin{bmatrix}1 & 1 & 2\\ 1 & 1 & 2\\ 3 & 3 & 4\end{bmatrix}}_{A=[\![Q_1]\!]^M(z)} \cdot \underbrace{\begin{bmatrix}2 & 2 & 1\\ 2 & 2 & 1\\ 1 & 1 & 5\end{bmatrix}}_{B=[\![Q_2]\!]^M(z)} = \underbrace{\begin{bmatrix}1 & 1 & 2\\ 1 & 1 & 2\\ 1 & 1 & 4\end{bmatrix}}_{C=[\![Q_1 \,;\, Q_2]\!]^M(z)},$$

where now we have used our matrix semantics (i.e., $\max$ instead of addition and $\min$ instead of multiplication). Note that in A and C (but not B), row 3 dominates rows 1 and 2, which is consistent with our rules for row domination in sequencing. Also, note that column 3 dominates columns 1 and 2 in C, even though this relationship does not occur in B, which can happen since our computed domination relationships underapproximate the true relationships.

Lemma 4. *If* $[\![Q]\!]^R(z)_{i,i'}$*, then* $i \succeq^R_{[\![Q]\!]^M(z)} i'$*, and if* $[\![Q]\!]^C(z)_{j,j'}$*, then* $j \succeq^C_{[\![Q]\!]^M(z)} j'$*.*

We give a proof in Section F.4. That is, $[\![\cdot]\!]^R$ and $[\![\cdot]\!]^C$ compute conservative underapproximations of the row and column domination relationships, respectively.

Next, we describe how we can use $[\![Q]\!]^R(z)$ and $[\![Q]\!]^C(z)$ to reduce the dimensionality of the matrices involved in computing $[\![Q]\!]^M(z)$. To this end, we define a sparse variant $[\![\cdot]\!]^S_{\cdot,\cdot}$ of the matrix semantics

[1] We note that our more general sparsification strategy is specific to our quantitative semantics, and does not work for regular matrix multiplication.

$$
\begin{aligned}
[\![\varphi_{??}]\!]^S_{I,J}(z) &\coloneqq [\![\varphi_{??}]\!]^M(z)_{I,J} \\
[\![Q_1 \wedge Q_2]\!]^S_{I,J}(z) &\coloneqq \min\{[\![Q_1]\!]^S_{I,J}(z), [\![Q_2]\!]^S_{I,J}(z)\} \\
[\![Q_1 \vee Q_2]\!]^S_{I,J}(z) &\coloneqq \max\{[\![Q_1]\!]^S_{I,J}(z), [\![Q_2]\!]^S_{I,J}(z)\} \\
[\![Q_1 \,;\, Q_2]\!]^S_{I,J}(z) &\coloneqq [\![Q_1]\!]^S_{I,K}(z) \cdot [\![Q_2]\!]^S_{K,J}(z) \qquad (K = \text{maximal}([\![Q_1]\!]^C(z) \wedge [\![Q_2]\!]^R(z)))
\end{aligned}
$$

Figure 7: The sparse matrix semantics. Here $I, J \in \mathcal{L}(\mathcal{I}_n)$ are lists of indices. In the rule for $\varphi_{??}$, given $A \in \mathbb{R}^{(n+1)\times(n+1)}$, the matrix $A_{I,J} \in \mathbb{R}^{|I|\times|J|}$ consists of the rows of A indexed by I and columns of A indexed by J. The rule for φ is identical to that for $\varphi_{??}$. Finally, given $A \in \mathbb{B}^{(n+1)\times(n+1)}$, the list maximal$(A) \in \mathcal{L}(\mathcal{I}_n)$ denotes the maximal elements of the partial order defined by A. Matrix multiplication is defined as in Figure 5.

$[\![\cdot]\!]^M$ that only computes a given subset of rows and columns of the output matrix. These semantics have type

$$[\![\cdot]\!]^S_{I,J} : \mathcal{Q} \to \mathcal{Z} \to \mathbb{R}^{|I|\times|J|},$$

where $I, J \in \mathcal{L}(\mathcal{I}_n)$ are lists of integers, and are defined to satisfy

$$[\![Q]\!]^S_{I,J}(z) = [\![Q]\!]^M(z)_{I,J},$$

where given a matrix $A \in \mathbb{R}^{(n+1)\times(n+1)}$, we define $A_{I,J} \in \mathbb{R}^{|I|\times|J|}$ to be the matrix consisting of entries of A in the rows indexed by I and columns indexed by J. These semantics are shown in Figure 7. For the case $\varphi_{??}$, we evaluate $[\![\varphi_{??}]\!]^M(z)$ and return its rows I and columns J. Conjunction and disjunction are straightforward. The main case of interest is sequencing $Q = Q_1 \,;\, Q_2$, which uses the row domination relationships of Q_1 and column domination relationships of Q_2. In particular, note that a set of domination relationships form a partial order on $\mathcal{I}_n$. Thus, the matrix $A = [\![Q_1]\!]^C(z) \wedge [\![Q_2]\!]^R(z)$ encodes the partial order obtained by the intersection of the column domination relationships of $[\![Q_1]\!]^M$ and the row domination relationships of Q_2. Then, the quantity

$$K = \text{maximal}(A) \in \mathcal{L}(\mathcal{I}_n)$$

is the list of indices that are maximal elements according to the partial order encoded by A—i.e., indices $i \in K$ are not dominated by any other $i' \in \mathcal{I}_n$ (the order of indices in K can be arbitrary).[2]

Lemma 5. *For any $I, J \in \mathcal{L}(\mathcal{I}_n)$, we have $[\![Q]\!]^S_{I,J}(z) = [\![Q]\!]^M(z)_{I,J}$.*

We give a proof in Section F.5. That is, the sparse matrix semantics coincides with the regular matrix semantics on the indices evaluated. Together with Theorem 1, we have the following:

Theorem 3. *We have $[\![Q]\!]^q(z) = [\![Q]\!]^S_{\{0\},\{n\}}(z)$.*

In other words, we can evaluate our quantitative semantics using our sparse matrix semantics.

B.8 Theoretical Guarantees

We have the following guarantees for our algorithm.

Theorem 4. *Algorithm 1 computes the set of all $\bar{Q} \in \bar{\mathcal{Q}}$ such that $\psi_W(\bar{Q}) = 1$.*

This result follows directly from Lemma 1, which says that our pruning strategy does not prune any programs that are consistent with W (i.e., $\psi^{\#}_W(Q) = 0$ implies that $\psi_W(\bar{Q}) = 0$ for all completions $\bar{Q}$ of Q), and from Lemma 2, which says the same thing holds for our parameter pruning strategy (i.e., given candidate parameters θ, if $\theta_h \notin \Theta^{\#}_h(Q, W)$ for any h, then $\psi_W(Q_\theta) = 0$). In addition, note that all programs $\bar{Q} \in \ell^*$ returned by Algorithm 1 satisfy $\psi_W(\bar{Q}) = 1$ due to the check performed at the end. Thus, Algorithm 1 solves our synthesis problem.

Theorem 5. *For any $\bar{Q}, \bar{Q}' \in \ell^*$ returned by Algorithm 2, $[\![\bar{Q}]\!](z) = [\![\bar{Q}']\!](z)$ for all $z \in Z$.*

In other words, our active learning algorithm returns queries that are indistinguishable on the given unlabeled examples Z. This result follows due to the stopping condition of Algorithm 2.

[2] One detail is that there may be ties—i.e., $i \succeq_A i'$ and $i' \succeq_A i$, where $\succeq_A$ is the partial order encoded by A. If i, i' are otherwise maximal, we include exactly one of i, i' in K. Multi-way ties are resolved similarly.

Predicate	Description
InLaneK	Each lane K is represented by (i) a differentiable 2D curve $\zeta : \mathbb{I} \to \mathbb{R}^2$ in image coordinates, where $\mathbb{I} = [0, 1]$, (ii) a radius $r \in \mathbb{R}$, and (iii) an angle tolerance $\alpha \in [0, 2\pi)$. This predicate is false except for length-one trajectories. It checks if the distance between p and the closest point p' on ζ is less than the radius, and if the direction of the velocity v is within the angle tolerance of the direction of ζ at p' (i.e., the gradient of ζ at p').
$\mathsf{MinLength}_\theta$	This predicate applies to trajectories of any length. It checks if duration in seconds of the sequence is at least θ.
$\mathsf{MinAvgVel}_\theta$	This predicate applies to trajectories of nonzero length. It checks if the average velocity over the trajectory is at least θ.
$\mathsf{MinAvgAccel}_\theta$	This predicate applies to trajectores of nonzero length. It checks if the average acceleration over the trajectory is at least θ.
$\mathsf{Dist}_\theta(A, B)$	This multi-object predicate applies to length-one trajectories. It checks if the positions p_A and p_B of A and B, respectively, are within distance θ.
$\mathsf{SpeedRatio}_\theta(A, B)$	This multi-object applies to length-one trajectories. It checks if the ratio of speeds $\|v_A\|$ and $\|v_B\|$ of A and B, respectively, is greater than θ.

Table 2: The predicates included in QUIVR. Note that predicates evaluate to false when they are not applicable.

C Implementation

We describe the implementation of our system QUIVR, including the object tracking used to construct trajectories $z \in \mathcal{Z}$ from a video, the predicates Φ included in our experiments, and additional optimizations to our synthesis algorithm.

C.1 Object Tracking

We represent a video V as a sequence of video frames $V = (I_0, I_1, ..., I_{T-1})$—i.e., each I_t is a single 2D color image. We preprocess the video by running an object tracker on it, which outputs:

- **Object detections:** For each image I_t, a set of detections $D_t = \{d_1, d_2, ..., d_k\} \subseteq \mathbb{R}^4$, where a detection $d = (x, y, w, h)$ encodes a bounding box centered at image coordinates (x, y) with width w and height h.
- **Object associations:** For each image I_t, an edge $e \in E_t = D_t \times (D_{t+1} \cup \{\varnothing\})$ associating detections $d \in D_t$ in I_t with detections $d' \in D_{t+1}$ in the successive frame, or $\varnothing$ if no subsequent detection exists (e.g., the object exits the image, the object tracker misses a detection, or $t = T - 1$ so I_t is the last frame in the video).

Based on the object detections and assocations, we construct *object tracks* $s \in \mathcal{S} = \mathcal{D}^*$, which are sequences of detections of the same object—in particular, $s = (d_0, d_1, ..., d_{n-1})$ is a sequence such that $(d_i, d_{i+1}) \in E_t$ for all $i < n - 1$ (where $d_i \in D_t$), and $(d_{n-1}, \varnothing) \in E_{t'}$ (where $d_{n-1} \in D_{t'}$). Note that the relative index i of an object state in a trajectory z can be offset from the absolute index t of the corresponding frame in the video. Finally, we compute the corresponding sequence of object states $z = (x_0, x_1, ..., x_{n-1})$ for each object track s. In our implementation, we include the 2D position $p_i = (x, y) \in \mathbb{R}^2$ of $d_i = (x, y, w, h)$, as well as its velocity $v_i = p_i - p_{i-1}$ and acceleration $a_i = v_i - v_{i-1}$, and form the state $x_i = (p_i, v_i, a_i) \in (\mathbb{R} \cup \{\varnothing\})^6$. Note that that these values do not exist for all frames; when they do not exist, we use $\varnothing$. The object state can easily be extended by the user if desired. Our implementation uses an object tracker called DeepSORT [20], which is a simple object tracker that, given object detections, associates positions of the same object based on a combination of spatial cues (bounding box overlap and motion prediction) and image cues (similarity of the pixels corresponding to each detection). To obtain object detections, we use a YOLOv3 model [28] that is pretrained on ImageNet and fine-tuned on approximately 500 images hand-labeled with bounding boxes of cars and pedestrians.

C.2 Domain Specific Language

Predicates. Table 2 shows the predicates in our system. Note that the in-lane predicates rely on curves of lanes ζ. Since we are focusing on the setting of a small number of videos with a large number of trajectories each, we hand-annotated this information. In practice, there has been work on automatically inferring this information by clustering the trajectory data [29, 30].

Multi-object queries. For queries that refer to multiple objects, our system automatically constructs the product state space—i.e., $\mathcal{X} \subseteq (\mathbb{R} \cup \{\varnothing\})^{6k}$, where k is the number of objects. Predicates can either directly operate over the product state space (i.e., $\varphi : \mathcal{X} \to \mathbb{B}$) or operate over a subset of objects. In the latter case, the query must indicate which objects the predicate is referring to.

C.3 Synthesis Algorithm

Additional pruning. We can use our quantitative semantics to perform pruning not just after finding a sketch, but also at the level of partial queries. In particular, note that if $\Theta_h^{\#}(Q, W) = \varnothing$ for any parameter hole $h \in \mathcal{H}_\theta(Q)$, then no valid θ_h exists, which means that Q cannot be completed into a valid query. Thus, we can prune Q in this case—i.e., we redefine $\psi_W^{\#}$ to be

$$\psi_W^{\#}(Q) = \left(\bigwedge_{(z,y) \in W} y \in [\![Q]\!]^{\#}(z) \right) \wedge \left(\bigwedge_{h \in \mathcal{H}_\theta(Q)} \Theta_h^{\#}(Q, W) \neq \varnothing \right).$$

In particular, we have added a second term which requires that the parameter holes $h \in \mathcal{H}_\theta(Q)$ all have a nonempty set of consistent parameter values. For example, consider $Q = \langle \mathsf{Dist}_{??1}(A, B) \rangle^*$ and examples $W = \{(z_1, y_1), (z_2, y_2)\}$, where $z_1 = (1.0, 1.0)$, $y_1 = 0$, $z_2 = (2.0, 2.0)$, and $y_2 = 1$. Then, we have $\Theta_{??1}^{\#}(Q, z_1, y_1) = (-1.0, \infty)$ and $\Theta_{??1}^{\#}(Q, z_2, y_2) = (-\infty, -2.0]$. As a consequence, $\Theta_{??1}^{\#}(Q, W) = \varnothing$, which implies that we can prune Q.

Evaluating the Boolean semantics. We note that the matrix semantics and sparsification optimizations described in Section B.5 can directly be applied to our Boolean semantics, since the Boolean semantics is a special case of the quantitative semantics where we take

$$\iota_\varphi(z) = \begin{cases} \infty & \text{if } [\![\varphi]\!](z) = 1 \\ -\infty & \text{otherwise.} \end{cases}$$

Thus, we use these optimizations for both our Boolean semantics and our quantitative semantics.

Synthesis search space. To make search more tractable, we restrict the search space used in our evaluation in several ways. First, we omit disjunctions from the search space, nested Kleene star operators, as well as Kleene star around sequencing, since these constructs do not occur in any of our queries. In addition, we remove semantically equivalent duplicates during enumeration, accounting for associativity of sequencing and conjunction as well as commutativity of conjunction. Finally, we bound the number of atomic predicates in our queries by 5, and their depth by 3; here, depth is counted in terms of nesting different constructs—e.g., $\varphi_1 \,;\, \varphi_2 \,;\, \varphi_3$ has depth 1, whereas $\varphi_1 \wedge (\varphi_2 \,;\, \varphi_3)$ has depth 2 and $\varphi_1 \wedge (\varphi_2 \,;\, \varphi_3^*)$ has depth 3. This last restriction ensures that our search space is finite (though large), which enables us to enumerate all consistent queries.

Sampling-based active learning. For some of our queries (in particular, multi-object queries), evaluating our active learning objective exactly is computationally intractable, since it involves evaluating all consistent examples $Q \in \ell^*$ on all unlabeled trajectories $z \in Z$. For these queries, we use a sampling-based approximation. In particular, we subsample both the queries $Q \in \ell^*$ and the unlabeled trajectories $z \in Z$, and select the best trajectory z^* to label based on these samples. The remainder of the algorithm continues without subsampling; in particular, we prune all queries $Q \in \ell^*$ that are inconsistent with (z^*, y^*), where $y^* = \mathcal{O}(z^*)$ is the user-provided label.

Additional optimizations. We use two additional optimizations in our query evaluation. First, we memoize the results of sub-queries across *all* queries encountered during synthesis. This strategy significantly improves performance since sub-queries are often shared across many queries in the search space. Second, evaluating our matrix semantics for Kleene star involves raising a matrix to a power k—i.e., M^k. We use the standard technique where we compute the powers of two $I, M, M^2, M^4, ...$, and then multiply the appropriate matrices to obtain M^k.

ID	Query
1	$\langle \mathsf{True} \rangle^* \,;\, \langle \mathsf{InLane1} \rangle \,;\, \langle \mathsf{True} \rangle^* \,;\, \langle \mathsf{InLane2} \rangle \,;\, \langle \mathsf{True} \rangle^*$ This query selects cars that turn from lane 1 into lane 2.
2	$\langle \mathsf{True} \rangle^* \,;\, \langle \mathsf{InLane1} \rangle^* \wedge \left\langle \mathsf{MinAvgAccel}_{??} \right\rangle \wedge \left\langle \mathsf{MinLength}_5 \right\rangle \,;\, \langle \mathsf{True} \rangle^*$ This query selects cars that are in lane 1 and accelerate rapidly.
3	$\langle \mathsf{True} \rangle^* \,;\, \left(\left\langle \mathsf{InLane1}(A) \right\rangle \,;\, \langle \mathsf{True} \rangle^* \,;\, \left\langle \mathsf{InLane2}(A) \right\rangle \right) \wedge \left\langle \mathsf{InLane2}(B) \right\rangle \,;\, \langle \mathsf{True} \rangle^*$ This query selects pairs where car A is turning into a lane while car B is in that lane.
4	$\langle \mathsf{True} \rangle^* \,;\, \left(\left\langle \mathsf{InLane1}(A) \right\rangle \wedge \left\langle \mathsf{InLane2}(B) \right\rangle \right)^* \wedge \left\langle \mathsf{SpeedRatio}_{??}(A,B) \right\rangle \wedge \left\langle \mathsf{MinLength}_5 \right\rangle \,;\, \langle \mathsf{True} \rangle^*$ This query selects pairs where the cars are going very different speeds and in adjacent lanes.
5	$\langle \mathsf{True} \rangle^* \,;\, \left(\left\langle \mathsf{InLane1}(A) \right\rangle \wedge \left\langle \mathsf{InLane2}(B) \right\rangle \wedge \left\langle \mathsf{Dist}_{??}(A,B) \right\rangle \right)^* \wedge \left\langle \mathsf{MinLength}_5 \right\rangle \,;\, \langle \mathsf{True} \rangle^*$ This query selects pairs where the cars are close and in adjacent lanes.
6	$\langle \mathsf{True} \rangle^* \,;\, \left(\left\langle \mathsf{InLane1}(A) \right\rangle \wedge \left\langle \mathsf{InLane1}(B) \right\rangle \wedge \left\langle \mathsf{Dist}_{??}(A,B) \right\rangle \right)^* \wedge \left\langle \mathsf{MinLength}_5 \right\rangle \,;\, \langle \mathsf{True} \rangle^*$ This query selects pairs where the cars are close and in the same lane.

Table 3: Six examples of queries in our language written for the YTStreams dataset. The first two are single-object queries, and the last four are multi-object queries.

D Evaluation

We evaluate our approach by showing how it can be used to synthesize queries to solve interesting tasks, achieving good performance given just a small number of initial training examples. In particular, we address the following questions:

- Can our language capture interesting examples of human driving behaviors?
- Can our algorithm synthesize queries that achieve good performance on a held-out test set given just a few initial training examples?
- Do our optimizations described in Section B.5 reduce the running time of our synthesizer?

We first describe our experimental setup, and address the first question by demonstrating several examples of interesting queries that can be expressed in our language (Section D.1). Then, we provide performance results for our synthesis algorithm, showing it is both data-efficient, requiring only a handful of training examples to achieve high accuracy (Section D.2), and compute-efficient, executing substantially faster with our performance optimizations (Section D.3).

D.1 Experimental Setup

Video data. Our evaluation leverages video data from the YTStreams dataset [14]. This dataset includes 60 hours of video collected from live YouTube feeds of several fixed-position traffic cameras. We use five hours of video from two of the available cameras—namely, one in Tokyo and one in Warsaw. We extracted trajectories of cars and pedestrians from this video using a variant of the SORT method for multi-object tracking [31] and a YOLOv3 object detection model [28] fine-tuned to this dataset on approximately 500 images labeled with bounding boxes.

Ground truth queries. To evaluate QUIVR, we manually wrote 6 ground truth queries, including both the sketch and the parameters; these queries are shown in Table 3. Several queries apply to multiple configurations (e.g., different pairs of lanes), resulting in a total of 11 queries. The ground

Figure 8: Trajectories selected by multi-object queries. Each image shows two trajectories; the color of each one changes from red to green to blue to denote the progression of time. Top: Unprotected right turn into lane with oncoming traffic, as in Figure 1. Middle: The car on the bottom drives faster than the one on top and passes it. Bottom: One car driving closely behind the other.

truth parameters were chosen by manually evaluating the query on the dataset and visually examining whether they were selecting the desired trajectories. As can be seen, these queries cover a wide range of behaviors. For instance, accounting for the behavior of nearby human drivers when making an unprotected turn is an important challenge for autonomous cars [2], as is accounting for the behavior of cars that are trying to pass [3]. Our language can be used to express queries capturing these kinds of behaviors. Finally, we show examples of trajectories selected using three of our multi-object queries in Figure 8.

Synthesis. To apply our approach on the video data, we initialize it with a randomly sampled set of labeled examples, followed by up to 25 actively labeled examples using our active learning algorithm. In particular, we first divide the set Z of all trajectories in the video into a train set Z_{train} and a test set Z_{test}; we do so temporally—i.e., trajectories occurring in the first half of the video are used for training, those in the second half are used for testing. We then use the ground truth query Q^* to construct a small set of initial labeled examples $W \subseteq Z_{\text{train}} \times \mathbb{B}$. Specifically, we construct W by sampling 2 positive and 10 negative examples Z_{init} uniformly at random from Z_{train}; then, we let

$$W = \{(z, y) \mid z \in Z_{\text{init}},\ y = [\![Q^*]\!](z)\}.$$

The subsequent active learning steps choose among the remaining unlabeled examples in Z_{train}—i.e., the available unlabeled examples are $Z = Z_{\text{train}} \setminus Z_{\text{init}}$. Then, we simulate the human by evaluating each trajectory z selected by our active learning algorithm under Q^*—i.e., $\mathcal{O}(z) = [\![Q^*]\!](z)$.

D.2 Accuracy of Synthesized Queries

First, we show QUIVR is able to synthesize queries with few labeled examples, by comparing our full approach to a baseline that replaces the active learning component with uniform randomly sampling examples $z \sim \text{Uniform}(Z)$ to label.

ID	Query	0 Steps		5 Steps		10 Steps		25 Steps	
		Ours	Random	Ours	Random	Ours	Random	Ours	Random
1	Shibuya, 1-object, A	0.06	0.06	1.00	0.06	1.00	1.00	1.00	1.00
1	Shibuya, 1-object, B	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
1	Shibuya, 1-object, C	0.39	0.39	1.00	0.39	1.00	0.39	1.00	0.42
1	Shibuya, 1-object, D	0.99	0.99	1.00	0.99	1.00	0.99	1.00	0.99
2	Warsaw, 1-object, A	0.27	0.28	0.38	0.22	1.00	0.22	1.00	0.33
2	Warsaw, 1-object, B	0.54	0.50	0.91	0.72	0.98	0.72	0.98	0.80
3	Warsaw, 2-object, A	0.24	0.55	0.50	0.46	0.67	0.46	0.91	0.50
4	Warsaw, 2-object, B	0.07	0.07	0.38	0.08	0.55	0.10	1.00	0.09
4	Warsaw, 2-object, C	0.13	0.13	0.32	0.14	0.63	0.14	0.78	0.27
5	Warsaw, 2-object, D	0.10	0.10	0.46	0.31	0.53	0.32	0.53	0.22
5	Warsaw, 2-object, E	0.08	0.07	0.40	0.13	0.60	0.12	0.71	0.12

Table 4: Median F_1 score after steps of active learning, with our algorithm for selecting tracks to label ("Ours") and a baseline uniformly random choice ("Random"). "ID" shows the corresponding query in Table 3 (recall that each query may have multiple instantiations), and "Query" describes the query.

Metrics. We evaluate the F_1 score of the query synthesized using our algorithm given the initial examples W together with a varying number of actively labeled examples. One challenge is that our algorithm returns a list ℓ^* containing all queries consistent with the current examples. Thus, we report the median F_1 score across queries in ℓ^* compared to the ground truth query Q^* on the held-out test set Z_{test}. Also for 2-object queries, the size of $|Z_{\text{test}}|$ is significantly larger since it contains pairs of object trajectories; thus, evaluating all queries in ℓ^* on all trajectories in Z_{test} is intractable. Thus, in these cases, if there are more than 100 test examples (resp., 1000 candidate queries), we sample 100 examples (resp., 1000 candidate queries) uniformly at random, and report results based on these subsets.

Baselines. We compare our algorithm against an ablation where we substitute the active learning component with a simple sampling method that selects trajectories z uniformly at random from the remaining unlabeled training examples Z to ask users to label.

Results. We show the F_1 score of each of the 11 queries in Table 4 after 0, 5, 10, and 25 steps of active learning. After just 10 steps, our approach provides F_1 score of 1.0 on 5 of 11 queries. Furthermore, after 25 steps, it yields an F_1 score of 0.9 on average across all queries. Thus, by selecting examples for the user to label that most reduce uncertainty in expectation, QUIVR is able to rapidly synthesize queries with relatively little user input. In Figure 2, we show the learning curve of an individual run, with both our approach and the uniform sampling baseline. While our approach provides a similar accuracy to uniform sampling early on, accuracy with active learning rapidly improves on this run after 7 active learning steps, showing that QUIVR effectively selects trajectories that best disambiguate between the multiple consistent queries present on each step.

D.3 Synthesis Running Time

Next, we show QUIVR efficiently synthesizes queries, with its quantitative semantics and matrix sparsification optimizations providing substantial speedups.

Metrics. We evaluate the running time, focusing on the enumerative search phase of our algorithm (i.e., Algorithm 1). During active learning, we found that the speedups from sparsification remain similar (though pruning may have a smaller advantage here).

Baselines. We compare the running time of our system with two ablations. First, we compare to omitting the parameter space pruning based on quantitative semantics—i.e., we treat each parameter value $\theta_h \in \Theta_h$ as a separate query. Second, we compare to omitting our matrix sparsification optimization. Finally, we also compare to a third approach where we run the query on a GPU. This comparison demonstrates an additional advantage of our matrix semantics—when a GPU is available, we can in some cases use it to further improve performance.

Results. In Figure 5, we report the running time of our algorithms on a CPU (an Intel Xeon 6148 CPU at 2.40GHz) and a GPU (an Nvidia GeForce RTX 2080 Ti). On the GPU, the sparse semantics

Query	Boolean	Quantitative	Sparse	GPU
Shibuya, 1-object, A	333 ± 4	51 ± 2	41 ± 1	23 ± 11
Shibuya, 1-object, B	235 ± 2	39 ± 1	33 ± 1	19 ± 10
Shibuya, 1-object, C	251 ± 2	40 ± 3	34 ± 2	20 ± 10
Shibuya, 1-object, D	329 ± 3	54 ± 1	44 ± 1	24 ± 13
Warsaw, 1-object, A	160 ± 4	27 ± 2	30 ± 1	21 ± 11
Warsaw, 1-object, B	209 ± 2	35 ± 2	36 ± 1	20 ± 11
Warsaw, 2-object, A	5830 ± 1	891 ± 59	604 ± 79	–
Warsaw, 2-object, B	5073 ± 28	814 ± 30	469 ± 10	–
Warsaw, 2-object, C	5436 ± 6	870 ± 9	552 ± 23	–
Warsaw, 2-object, D	5023 ± 6	797 ± 38	538 ± 11	–
Warsaw, 2-object, E	5021 ± 15	807 ± 26	528 ± 11	–

Table 5: Running time of synthesis (mean $\pm$ stddev, in seconds) for the Boolean, quantitative, and sparse semantics, with 0 steps of active learning. For the GPU results, we use quantitative semantics. Missing results indicate an out-of-memory error. These numbers are over 3 trials.

was slower than the quantitative semantics, and the Boolean semantics always ran out of memory, so we use the quantitative semantics. On the CPU, the quantitative semantics provide a $6.2\times$ speedup on average, as well as a significant reduction in memory usage. The sparse semantics provide an additional $1.3\times$ speedup across all queries. As can be seen, the sparse semantics are especially useful for slower, 2-object queries, where it rises to a $1.6\times$ speedup. When available and when it does not run out of memory, the GPU provides a $1.9\times$ speedup on average.

E Related Work

Querying video data. There has been recent work on querying object detections and trajectories in video data [10, 11, 12, 13, 14, 15, 16, 17]. While the primary focus of this line of work has been on improving scalability, some of the approaches also touch on the language-design aspect [11, 14, 17]. However, these approaches focus on SQL-like operators such as select, inner-join, group-by, etc. over predefined predicates. These operators cannot capture compositions such as the sequencing and iteration operators in our language, which are necessary for identifying more complex behaviors.

Regular expression synthesis. There has been recent interest in synthesizing regular expressions from examples—for instance, using genetic algorithms [32]. Most closely related is work on syntax-guided synthesis of regular expressions [33]; our pruning strategy builds on their approach of overapproximating holes using true and false. There has also been work on multimodal synthesis of regular expressions (e.g., from both examples and natural language descriptions) [34], as well as work on repairing regular expressions rather than synthesizing them from scratch [35]. In contrast to this line of work, our synthesis problem is significantly more challenging due to the possibility of real-valued holes in predicates in our queries. Our language also includes additional complexities, particularly the use of atomic predicates that match subsequences rather than individual inputs.

Neurosymbolic models. There has been recent work on leveraging program synthesis in the context of machine learning. For instance, there has been work on using programs to represent high-level structure in images [36, 37, 38, 39, 40], as well as for reinforcement learning [41, 42, 43, 44]. In contrast, we use programs to classify predicted trajectories.

The most closely related work in this direction is on synthesizing functional programs with neural components operating over lists [45, 8], including work on synthesizing web scraping programs that use neural components to handle word meaning [46]. Our work includes key constructs tailored to our domain that are not included in their languages. Most importantly, we include a sequencing operator; in their functional language, such an operator would need to be represented as a nested series of if-then-else operators. In addition, their language does not support predicates that match variable-length inputs; while such a predicate could be added to their language, none of their operators (e.g., map and fold) are designed to compose such predicates (e.g., sequencing). While we include iteration, this operator can be captured using fold.

Quantitative synthesis. There has been recent work on program synthesis with quantitative properties—e.g., on synthesis for producing optimized code [47, 48, 49], for approximate comput-

ing [50, 51], for probabilistic programming [52], and for embedded control [53]. These approaches largely focus on search-based synthesis, either using constraint optimization [50], continuous optimization [53], enumerative search [51, 48], or stochastic search [47, 52, 49]. We leverage these ideas in the context of our application domain.

Programming by example. There has been a great deal of recent interest in programming by example—for instance, to synthesize string processing programs [54, 55], list processing programs [56, 57, 58], functional programs [59, 60], table transformations [61, 62], and SQL queries [63]. A key challenge in our setting is the need to search over real-valued parameters.

Quantitative semantics. There has been work on quantitative regular expressions (QREs). In general, QREs cannot be efficiently evaluated due to their nondeterminism; our language is restricted to ensure efficient computation. More broadly, there has been work on quantitative semantics for temporal logic for robust constraint satisfaction [64, 65, 66], and to guide reinforcement learning [67]. Similarly, there has been work using the Viterbi semiring to obtain quantitative semantics for Datalog programs [68], which they use in conjunction with gradient descent to learn the rules of the Datalog program. In contrast, we use our quantitative semantics to efficiently compute cutoffs at which we can prune the parameter search space; in particular, we prove that these semantics compute this cutoff. Finally, there has been work on using GPUs to evaluate regular expressions [69]; however, they focus on evaluating traditional regular expressions over strings.

Active learning for program synthesis. There has been recent interest in leveraging active learning to reduce ambiguity in the context of program synthesis [70], including the use of greedy active learning strategies similar to ours [24]. Our focus is on the query language and synthesis algorithm, but show that leveraging existing active learning strategies can reduce ambiguity in our domain.

Machine learning for program synthesis. There has been work on using machine learning to learn an objective to disambiguate among different programs that satisfy an ambiguous specification [71, 72]; however, the goal is to synthesize a logical program rather than one operating over noisy inputs. More broadly, there has been work using machine learning to guide synthesis [57, 62, 73, 74, 61, 58], but again, the goal is to synthesize a logical program.

F Proofs

F.1 Proof of Lemma 1

It suffices to show that if there exists a completion $\bar{Q}$ of Q such that if $[\![\bar{Q}]\!] = 1$, then $[\![Q^+]\!] = 1$, and if $[\![\bar{Q}]\!] = 0$, then $[\![Q^-]\!] = 0$. We prove by structural induction on Q. The case $Q = ??$ follows since $Q^+ = \langle \mathsf{true} \rangle^*$, so $[\![Q^+]\!](z) = 1$, and $Q^- = \langle \mathsf{false} \rangle$, so $[\![Q^-]\!](z) = 0$. For the case $Q = \varphi_{??}$, recall that $[\![\varphi_\theta]\!](z) = \mathbb{1}(\iota_\varphi(z) \geq \theta)$, and that $Q^+ = \varphi_{-\infty}$ and $Q^- = \varphi_\infty$. Thus, for any $\theta \in \mathbb{R}$, if $[\![\varphi_\theta]\!](z) = 1$, then $[\![Q^+]\!](z) = 1$, and if $[\![\varphi_\theta]\!](z) = 0$, then $[\![Q^-]\!](z) = 0$, so the claim follows. The case φ follows since it has no holes, so $\bar{Q} = Q^+ = Q^-$ and $Q = \bar{Q}$ for every completion $\bar{Q}$ of Q.

Next, consider the case $Q = Q_0 \,;\, Q_1$. Note that $Q^+ = Q_0^+ \,;\, Q_1^+$, and $Q^- = Q_0^- \,;\, Q_1^-$. Furthermore, for any completion $\bar{Q}$ of Q, we have $\bar{Q} = \bar{Q}_0 \,;\, \bar{Q}_1$, where $\bar{Q}_0$ is a completion of Q_0 and $\bar{Q}_1$ is a completion of Q_1. Suppose that $[\![\bar{Q}]\!](z) = 1$. Then, there must exist $k \in \{0, 1, ..., n\}$, where $n = |z|$, such that $[\![\bar{Q}_0]\!](z_{0:k}) = [\![\bar{Q}_1]\!](z_{k:n}) = 1$. By induction, we have $[\![Q_0^+]\!](z_{0:k}) = [\![Q_1^+]\!](z_{k:n}) = 1$, which implies that $[\![Q^+]\!](z) = 1$, as claimed. Conversely, suppose that $[\![\bar{Q}]\!](z) = 0$. Then, for any $k \in \{0, 1, ..., n\}$, we have either $[\![\bar{Q}_0]\!](z_{0:k}) = 0$ or $[\![\bar{Q}_1]\!](z_{k:n}) = 0$. By induction, for any $k \in \{0, 1, ..., n\}$, we have either $[\![Q_0^-]\!](z_{0:k}) = 0$ or $[\![Q_1^-]\!](z_{k:n}) = 0$, which implies that $[\![Q^-]\!](z) = 0$, as claimed.

Next, consider the case $Q = Q_0 \vee Q_1$. Note that $Q^+ = Q_0^+ \vee Q_1^+$, and $Q^- = Q_0^- \vee Q_1^-$. Furthermore, for any completion $\bar{Q}$ of Q, we have $\bar{Q} = \bar{Q}_0 \vee \bar{Q}_1$, where $\bar{Q}_0$ is a completion of Q_0 and $\bar{Q}_1$ is a completion of Q_1. Suppose that $[\![\bar{Q}]\!](z) = 1$. Then, we must have either $[\![\bar{Q}_0]\!](z) = 1$ or $[\![\bar{Q}_1]\!](z) = 1$. By induction, we have either $[\![Q_0^+]\!](z) = 1$ or $[\![Q_1^+]\!](z) = 1$, which implies that $[\![Q^+]\!](z) = 1$, as claimed. Conversely, suppose that $[\![\bar{Q}]\!](z) = 0$. Then, we must have $[\![\bar{Q}_0]\!](z) = [\![\bar{Q}_1]\!](z) = 0$. By induction, we have $[\![Q_0^-]\!](z) = [\![Q_1^-]\!](z) = 0$, which implies that $[\![Q^-]\!] = 0$, as claimed. The case $Q = Q_0 \wedge Q_1$ follows by an analogous argument, so the lemma statement follows. $\square$

F.2 Proof of Lemma 2

We prove by structural induction on Q. First, consider the case $Q = \varphi_{??1}$, where $h = ??1$ (since we have assumed that Q only has a single parameter hole). Then, we have

$$[\![Q_{\theta_h}]\!](z) = \mathbb{1}(\iota_\varphi(z) \geq \theta_h) = \mathbb{1}([\![Q]\!]^q(z) \geq \theta_h),$$

The case $Q = ??$ cannot occur since we have assumed that Q has a single parameter hole. For the case $Q = \varphi$, we have

$$[\![Q]\!]^q(z) = \begin{cases} \infty & \text{if } [\![Q]\!](z) = 1 \\ -\infty & \text{if } [\![Q]\!](z) = 0, \end{cases}$$

so for $\theta_h \in \mathbb{R}$ (i.e., $\theta_h \neq \pm\infty$), it follows that $[\![Q]\!](z) = \mathbb{1}([\![Q]\!]^q(z) \geq \theta_h)$.

Next, consider the case $Q = Q_0 \,;\, Q_1$. Letting $n = |z|$, by induction, we have

$$\begin{aligned}
[\![Q]\!] &= \bigvee_{k \in \{0,1,\ldots,n\}} ([\![Q_0]\!](z_{0:k}) \wedge [\![Q_1]\!](z_{k:n})) \\
&= \bigvee_{k \in \{0,1,\ldots,n\}} \mathbb{1}([\![Q_0]\!]^q(z_{0:k}) \geq \theta_h) \wedge \mathbb{1}([\![Q_1]\!]^q(z_{k:n}) \geq \theta_h) \\
&= \bigvee_{k \in \{0,1,\ldots,n\}} \mathbb{1}\big(\max\{[\![Q_0]\!]^q(z_{0:k}), [\![Q_1]\!]^q(z_{k:n})\} \geq \theta_h\big) \\
&= \mathbb{1}\left(\min_{k \in \{0,1,\ldots,n\}} \max\{[\![Q_0]\!]^q(z_{0:k}), [\![Q_1]\!]^q(z_{k:n})\} \geq \theta_h\right) \\
&= \mathbb{1}([\![Q]\!]^q \geq \theta_h).
\end{aligned}$$

Next, consider the case $Q = Q_0 \vee Q_1$. By induction, we have

$$\begin{aligned}
[\![Q]\!] &= [\![Q_0]\!](z) \vee [\![Q_1]\!](z) \\
&= \mathbb{1}([\![Q_0]\!]^q(z) \geq \theta_h) \vee \mathbb{1}([\![Q_1]\!]^q(z) \geq \theta_h) \\
&= \mathbb{1}\big(\max\{[\![Q_0]\!]^q(z), [\![Q_1]\!]^q(z)\} \geq \theta_h\big) \\
&= \mathbb{1}([\![Q]\!]^q \geq \theta_h).
\end{aligned}$$

Finally, consider the case $Q = Q_0 \wedge Q_1$. By induction, we have

$$\begin{aligned}
[\![Q]\!] &= [\![Q_0]\!](z) \wedge [\![Q_1]\!](z) \\
&= \mathbb{1}([\![Q_0]\!]^q(z) \geq \theta_h) \wedge \mathbb{1}([\![Q_1]\!]^q(z) \geq \theta_h) \\
&= \mathbb{1}\big(\min\{[\![Q_0]\!]^q(z), [\![Q_1]\!]^q(z)\} \geq \theta_h\big) \\
&= \mathbb{1}([\![Q]\!]^q \geq \theta_h).
\end{aligned}$$

The claim follows. □

F.3 Proof of Lemma 3

We prove by structural induction on Q. The base case $Q = \varphi$ follows by definition. Finally, for the case $Q = Q_1 \,;\, Q_2$, by induction, we have

$$\begin{aligned}
[\![Q_1 \,;\, Q_2]\!]^M(z)_{i,j} &= \max_{k \in \mathcal{I}_n} \min\{[\![Q_1]\!]^M(z)_{i,k}, [\![Q_2]\!]^M(z)_{k,j}\} \\
&= \max_{k \in \{i,i+1,\ldots,j\}} \min\{[\![Q_1]\!]^q(z_{i:k}), [\![Q_2]\!]^q(z_{k:j})\} \\
&= [\![Q_1 \,;\, Q_2]\!]^q(z_{i:j}),
\end{aligned}$$

where the second line follows since $\min\{[\![Q_1]\!]^M(z)_{i,k}, [\![Q_2]\!]^M(z)_{k,j}\} = -\infty$ if $k < i$ or $k > j$ (this fact itself follows by structural induction on Q), which is then ignored by the $\max$ over $k \in \mathcal{I}_n$. Next, for the case $Q = Q_1 \vee Q_2$, by induction, we have

$$\begin{aligned}
[\![Q_1 \vee Q_2]\!]^M(z)_{i,j} &= \max\{[\![Q_1]\!]^M(z)_{i,j}, [\![Q_2]\!]^M(z)_{i,j}\} \\
&= \max\{[\![Q_1]\!]^q(z_{i:j}), [\![Q_2]\!]^q(z_{i:j})\} \\
&= [\![Q_1 \vee Q_2]\!]^q(z_{i:j}).
\end{aligned}$$

Finally, for the case $Q = Q_1 \wedge Q_2$, by induction, we have

$$\begin{aligned}\llbracket Q_1 \wedge Q_2 \rrbracket^M(z)_{i,j} &= \min\{\llbracket Q_1 \rrbracket^M(z)_{i,j}, \llbracket Q_2 \rrbracket^M(z)_{i,j}\} \\ &= \min\{\llbracket Q_1 \rrbracket^q(z_{i:j}), \llbracket Q_2 \rrbracket^q(z_{i:j})\} \\ &= \llbracket Q_1 \wedge Q_2 \rrbracket^q(z_{i:j}).\end{aligned}$$

The claim follows. □

F.4 Proof of Lemma 4

We show the case $\llbracket \cdot \rrbracket^R$; the case $\llbracket \cdot \rrbracket^C$ follows by an analogous argument. We prove by structural induction on Q. First, the cases $Q = \varphi_{??}$ and $Q = \varphi$ hold by definition.

Next, suppose that $Q = Q_1 \,;\, Q_2$, and assume $\llbracket Q \rrbracket^R(z)_{i,i'} = 1$. Then, by definition, we have $\llbracket Q_1 \rrbracket^R(z)_{i,i'} = 1$, so by induction, we have $\llbracket Q_1 \rrbracket^M(z)_{i,j} \geq \llbracket Q_1 \rrbracket^M(z)_{i',j}$ for all $j \in \mathcal{I}_n$. Thus, we have

$$\llbracket Q \rrbracket^M(z)_{i,j} = \max_{k \in \mathcal{I}_n} \min\{\llbracket Q_1 \rrbracket^M_{i,k}, \llbracket Q_2 \rrbracket^M_{k,j}\} \geq \max_{k \in \mathcal{I}_n} \min\{\llbracket Q_1 \rrbracket^M_{i',k}, \llbracket Q_2 \rrbracket^M_{k,j}\} = \llbracket Q \rrbracket^M(z)_{i',j}.$$

Next, suppose that $Q = Q_1 \vee Q_2$, and assume $\llbracket Q \rrbracket^R_{i,i'}(z) = 1$. Then, by definition, we have $\llbracket Q_1 \rrbracket^R(z)_{i,i'} = \llbracket Q_2 \rrbracket^R(z)_{i,i'} = 1$, so by induction, we have $\llbracket Q_1 \rrbracket^M_{i,j} \geq \llbracket Q_1 \rrbracket^M(z)_{i',j}$ and $\llbracket Q_2 \rrbracket^M(z)_{i,j} \geq \llbracket Q_2 \rrbracket^M(z)_{i',j}$ for all $j \in \mathcal{I}_n$. Thus, we have

$$\begin{aligned}\llbracket Q_1 \vee Q_2 \rrbracket^M(z)_{i,j} &= \max\{\llbracket Q_1 \rrbracket^M(z)_{i,j}, \llbracket Q_2 \rrbracket^M(z)_{i,j}\} \\ &\geq \max\{\llbracket Q_1 \rrbracket^M(z)_{i',j}, \llbracket Q_2 \rrbracket^M(z)_{i',j}\} \\ &= \llbracket Q_1 \vee Q_2 \rrbracket^M(z)_{i',j}.\end{aligned}$$

Finally, suppose that $Q = Q_1 \wedge Q_2$, and assume $\llbracket Q \rrbracket^R(z)_{i,i'} = 1$. Then, by definition, we have $\llbracket Q_1 \rrbracket^R(z)_{i,i'} = \llbracket Q_2 \rrbracket^R(z)_{i,i'} = 1$, so by induction, we have $\llbracket Q_1 \rrbracket^M_{i,j} \geq \llbracket Q_1 \rrbracket^M(z)_{i',j}$ and $\llbracket Q_2 \rrbracket^M(z)_{i,j} \geq \llbracket Q_2 \rrbracket^M(z)_{i',j}$ for all $j \in \mathcal{I}_n$. Thus, we have

$$\begin{aligned}\llbracket Q_1 \wedge Q_2 \rrbracket^M(z)_{i,j} &= \min\{\llbracket Q_1 \rrbracket^M(z)_{i,j}, \llbracket Q_2 \rrbracket^M(z)_{i,j}\} \\ &\geq \min\{\llbracket Q_1 \rrbracket^M(z)_{i',j}, \llbracket Q_2 \rrbracket^M(z)_{i',j}\} \\ &= \llbracket Q_1 \wedge Q_2 \rrbracket^M(z)_{i',j}.\end{aligned}$$

The claim follows. □

F.5 Proof of Lemma 5

We prove by structural induction on Q. First, the cases $Q = \varphi_{??}$ and $Q = \varphi$ hold by definition. Next, suppose that $Q = Q_1 \,;\, Q_2$. By induction, for $i \in \mathcal{I}_{|I|}$ and $j \in \mathcal{I}_{|K|}$, we have

$$\begin{aligned}\llbracket Q_1 \,;\, Q_2 \rrbracket^S_{I,J}(z)_{i,j} &= \max_{k \in \mathcal{I}_{|K|}} \min\{\llbracket Q_1 \rrbracket^S_{I,K}(z)_{i,k}, \llbracket Q_2 \rrbracket^S_{K,J}(z)_{k,j}\} \\ &= \max_{k \in \mathcal{I}_{|K|}} \min\{(\llbracket Q_1 \rrbracket^M(z)_{I,K})_{i,k}, (\llbracket Q_2 \rrbracket^M(z)_{K,J})_{k,j}\} \\ &= \max_{k \in K} \min\{\llbracket Q_1 \rrbracket^M(z)_{I_i,k}, \llbracket Q_2 \rrbracket^M(z)_{k,J_j}\} \\ &= \max_{k \in \mathcal{I}_n} \min\{\llbracket Q_1 \rrbracket^M(z)_{I_i,k}, \llbracket Q_2 \rrbracket^M(z)_{k,J_j}\} \\ &= \llbracket Q_1 \,;\, Q_2 \rrbracket^M(z)_{I_i,J_j},\end{aligned}$$

where $I_i \in \mathcal{I}_n$ is the ith element of the list I and $J_j \in \mathcal{I}_n$ is the jth element of J. The key step is the fourth line. To see this step, note that by maximality of K (and by Lemma 4), for any $k' \notin K$, there exists $k \in K$ such that $\llbracket Q_1 \rrbracket^M_{i,k} \geq \llbracket Q_1 \rrbracket^M_{i,k'}$ and $\llbracket Q_2 \rrbracket^M_{k,j} \geq \llbracket Q_2 \rrbracket^M_{k',j}$, which implies that

$$\min\{\llbracket Q_1 \rrbracket^M(z)_{I_i,k}, \llbracket Q_2 \rrbracket^M(z)_{k,J_j}\} \geq \min\{\llbracket Q_1 \rrbracket^M(z)_{I_i,k'}, \llbracket Q_2 \rrbracket^M(z)_{k',J_j}\},$$

which in turn implies the fourth line above. As a consequence, we have

$$\llbracket Q_1 \,;\, Q_2 \rrbracket^S_{I,J}(z) = \llbracket Q_1 \,;\, Q_2 \rrbracket^M(z)_{I,J}.$$

Next, suppose that $Q = Q_1 \vee Q_2$. By induction, we have

$$\begin{aligned}
[\![Q_1 \vee Q_2]\!]^S_{I,J}(z) &= \max\{[\![Q_1]\!]^S_{I,J}(z), [\![Q_2]\!]^S_{I,J}(z)\} \\
&= \max\{[\![Q_1]\!]^M(z)_{I,J}, [\![Q_2]\!]^M(z)_{I,J}\} \\
&= [\![Q_1 \vee Q_2]\!]^M_{I,J}.
\end{aligned}$$

The case $Q = Q_1 \wedge Q_2$ follows by an analogous argument, so the claim follows. $\square$