Research Statement

Konstantinos Mamouras
www.cis.upenn.edu/~mamouras

Programming Systems with a Rigorous Foundation

My research mission is to help programmers build efficient and reliable applications and systems. Towards this end, I draw on the theory of programming languages to design programming abstractions and software tools that facilitate guarantees of performance and correctness.

We are currently experiencing the emergence of a new ecosystem of technologies and applications under the name of *Internet of Things* (IoT), which offers the promise of creating a “smart” world of fully connected devices. The rise of IoT is accompanied by the burgeoning of the *Edge Computing* paradigm, which complements cloud computing by pushing part of the data processing and decision-making logic towards the edge of the system, near the source of the data. The resource-constrained nature and unreliable connectivity of the computing devices on the edge of an IoT system presents new research challenges in order to provide (1) ease of programmability, (2) predictable and reproducible behavior, (3) strong guarantees on resource usage, and (4) verifiability of correctness properties.

My current research focuses on the challenges posed by these technological trends, and my approach is grounded in programming language theory. I have demonstrated the fruitfulness of applying sound programming principles to practical systems development in the context of *data stream processing*, which is a core computational task for big data analytics and decision-making in IoT systems. Two crucial shortcomings of existing languages and systems for stream processing are that (i) their resource consumption is highly variable thus resulting in unpredictable performance, and (ii) they often lack a rigorous semantics and the ability to introduce parallelism in a way that guarantees the preservation of system behavior.

I have designed the StreamQRE (pronounced StreamQuery) language and runtime system [MRA⁺¹⁷], which proposes a novel set of domain-specific programming abstractions for stream processing. The StreamQRE proposal, which is based on the theoretical foundations of regular functions, offers **strong guarantees** on resource usage, whereas competing approaches suffer from unpredictable performance that depends highly on the input data set. I have also implemented a **typed framework** for specifying and deploying highly parallelizable streaming computations on distributed stream processing platforms such as Apache Storm [AMST¹⁷]. This framework is based on the idea that a computation is described as an acyclic dataflow graph, where each communication channel is typed according to a logical partial order and each processing element is guaranteed to not depend on an ad hoc linearization of the partially ordered streams. This methodology enables **semantically sound data parallelization**, whereas existing systems allow semantically unsound parallelization. Moreover, it incurs a negligible computational overhead compared to unconstrained hand-optimized code, while offering superior correctness and reproducibility guarantees.

My prior research includes work on foundational aspects of programming languages, in particular semantic models of system behavior, algebraic theories of program equivalence, verification logics, and state-based models of computation.

Stream Processing with Predictable Performance

Over the past 15 years there has been considerable interest in designing languages and systems for the processing of streaming data, in order to accommodate real-time decision making applications such as network traffic monitoring, stock and currency day trading, web log and clickstream analysis, and the processing of data feeds from sensors and medical devices. Currently, the rapid emergence of IoT systems —such as smart buildings and wearable, mobile healthcare technology— presents new opportunities for analytics over streaming data.

Most existing stream processing languages and systems are based on the abstractions of relational databases and the distributed dataflow model of computation. They typically use an SQL-like syntax augmented with count- and time-based windowing constructs for computing over a recent finite span of the data stream. Systems that are based on these abstractions have been shown to be effective for processing data at a massive scale, but their unpredictable behavior and performance makes them less suitable for emerging IoT applications. Another crucial disadvantage of these window-based approaches is that they severely restrict the ability to express computations that depend on the sequence of events in the entire stream history. A natural example of such a computation is the “maximum drawdown duration” in financial time series data, i.e. the maximum duration between consecutive all-time high values in the sequence of values.
StreamQRE. This situation drove my research to answer the following question: Can we extend the existing stream-processing languages with new programming abstractions for sequence-aware computation, while offering precise efficiency guarantees? As a response, we proposed the StreamQRE (Streaming Quantitative Regular Expressions) language for high-level programming over streaming data [MRA17]. This language integrates constructs that have been used in streaming extensions of relational database query languages with quantitative extensions of regular expressions. This novel combination allows the programmer to intermingle operations that are relational in nature, such as partitioning the input data by keys and integrating streams from different sources, with sequence-based operations that rely on a regular hierarchical structure of the input stream. The StreamQRE language has a small core of combinators with a clean formal semantics, and is based on the firm theoretical foundations of regular expressions and state-based models of computation. Its expressiveness encompasses several useful streaming transformations, such as selection, projection, join, aggregation, and various forms of windows (e.g., non-overlapping and overlapping/sliding, event-based, sessions).

A key distinction between StreamQRE and other streaming query languages is that a high-level query of StreamQRE can be compiled into a single-pass streaming algorithm with strong space- and time-efficiency guarantees. I showed that for a large subset of StreamQRE the evaluation algorithm has a small computational overhead in terms of memory footprint and per-item processing time, which is independent of the length of the stream and quadratic in the size of the query.

In order to validate the good performance of StreamQRE experimentally, I provided an implementation as a Java library and used it for two widely-used stream processing benchmarks: the Yahoo Streaming Benchmark, which analyzes a clickstream of user interaction with advertisements, and the NEXMark Benchmark, which processes the event stream of an online auction system. The StreamQRE engine compares favorably against three popular open-source streaming engines with Java implementations: RxJava, Esper, and Flink. The theoretical guarantees indeed translate to significantly better performance in practice (measured in terms of throughput): 2–4 times higher than RxJava, 6–75 times higher than Esper, and 10–140 times higher than Flink.

Usability of StreamQRE. The StreamQRE language syntax and semantics was taught to 85 students at the 2016 Marktoberdorf Summer School over the course of two lectures [AM17]. After the lectures, the students participated in two hands-on programming labs, where they had the chance to use the StreamQRE Java library to program several stream-processing tasks. I was happy to receive positive comments from students regarding their experience using StreamQRE. The students’ feedback consistently confirmed the design choices to base the language on the familiar programming abstractions of regular expressions and to include the functional “fold” combinator for aggregation.

Application to cardiac arrhythmia detection. We illustrated the practical usefulness of StreamQRE in the application area of medical devices, specifically for monitoring the cardiac rhythm of patients [AAM17]. In the domain of patient monitoring for cardiac arrhythmia, a small, implantable medical device continuously monitors the patient’s cardiac rhythm and delivers electrical therapy when needed. Given the resource-constrained nature of implantable medical devices, the streaming algorithms executed by them are carefully designed to minimize resource consumption. The implementation is typically done in a low-level language such as C, which is a laborious and costly task. I used the StreamQRE library to give a succinct high-level implementation of an industry-standard algorithm for arrhythmic tachycardia detection by monitoring the digitized electrical heart signal. This significant case study serves as strong evidence for the usefulness of StreamQRE as a domain-specific language for the rapid prototyping of medical monitoring algorithms. The efficiency guarantees that are offered by StreamQRE make it especially appropriate for this application domain, which requires stable and predictable resource consumption. An additional benefit of using StreamQRE for the implementation of the monitor is that it enables the static estimation of energy consumption (i.e., without simulation and measurement) by analyzing the query. This allows for easier exploration of the algorithm design space, and it opens the door for automatic query tuning to minimize energy consumption.

Automata-based models for stream processing. Finite-state automata and related formalisms have been used very effectively for pattern matching, and they naturally give rise to efficient algorithms in the streaming model of computation. We have introduced an automaton model, Streaming Automata, which corresponds expressively to a subset of the StreamQRE language. This is analogous to the correspondence of Nondeterministic Finite Automata to regular expressions [AMS17]. An important implication is that the query evaluation algorithm for this subset of StreamQRE can be expressed in the framework of Streaming Automata, which opens the door for performing query optimization via automata-theoretic state minimization procedures. I have also shown how the notions of Brzozowski and Antimirov derivatives from formal language theory can be extended to StreamQRE [AMU17]. This connection enables the use of syntactic derivatives to obtain an efficient streaming evaluation algorithm.

A specific computational problem that is widely used in the context of stream monitoring is the detection of a regular pattern over a span of the most recent events, i.e. over a sliding window. In recent work [GHK18], I have identified the computational complexity of finding the space resource requirements for regular pattern matching over a sliding window, assuming the pattern is specified by a nondeterministic finite automaton (NFA).
Distributed Stream Processing with Typed Abstractions

The current practice of distributed stream processing in popular systems —such as Apache Storm, Heron, and Apache Samza— is to express the overall computation in some variant of the distributed dataflow model of computation. A drawback of these systems is that they allow nondeterminism in the computation that results in unpredictable output that cannot be reproduced. In order to enable the precise functional specification of system behavior, we are advocating in [AMST17] the viewpoint that in a distributed setting the real-time events should be logically viewed as being partially ordered. This idea gives rise to a semantic framework where the finite prefixes of streams are modeled as data traces, which are partially ordered finite sets of events with a representation that is inspired from Mazurkiewicz traces. A streaming transformation is then formally modeled as a data-trace transduction, a monotone deterministic function from input data traces to output data traces. This generalizes both acyclic Kahn process networks and relational query processors, and can specify computations over data streams with a rich variety of partial ordering and synchronization characteristics.

In order to enforce predictability in the results of the computation, the processing elements of a distributed stream processing system should respect the logical partial order of events in the sense of not depending on some spurious interleaving of the events that is imposed by the communication network or the particularities of an implementation. For this goal, we have developed in [MSA+17] a rich input/output type discipline for classifying streams according to their partial ordering characteristics using data-trace types. These types are used to annotate the communication links in the dataflow graph that describes a streaming computation. Each vertex of this typed dataflow graph is programmed using a pre-defined set of templates, so as to ensure that (1) the code respects the types of the input/output channels, and (2) data parallelism can be introduced while preserving the semantics. I have provided an implementation of this typed framework in Java and an automatic compilation procedure for deployment on Apache Storm. I have shown experimentally that our framework can produce efficient implementations comparable to hand-coded solutions. Moreover, it be can used to express complex computations that are required in IoT streaming applications. This has been illustrated in [MSA+17] with a significant case study (inspired from the DEBS 2014 Grand Challenge) for the online prediction of power consumption using a machine-learned regression model, where the input stream consists of power sensor readings that are generated by “smart plugs” deployed at residences.

Semantics and Logics for Programs

While my more recent work is about programming language techniques for data processing applications, my prior Ph.D. work focuses on the foundational topics of program semantics and program logics for abstract compositional reasoning. The philosophy behind my Ph.D. work is to tame the undecidability of program verification tasks by suppressing the sources of hardness and focusing on the program abstractions that admit (1) deductive complete logical calculi, (2) decision procedures of manageable computational complexity, and (3) efficient procedures for constructing proofs that can serve as certificates of the desired program properties.

Equational theories based on Kleene algebra. My dissertation work on theories of program equivalence is based on the framework of Kleene Algebra (KA), the algebra of regular expressions, and its extension with Booleans called Kleene Algebra with Tests (KAT). KAT enables a propositional style of algebraic reasoning that abstracts away the details of the domain of computation and focuses on the control structure of the program. In my dissertation I showed how to enrich KAT with extra equational assumptions about the domain of computation without losing decidability or deductive completeness. The proposed framework of KA with equations [KM14] is powerful enough to unify several previous constructions, including the recent result on the completeness of the equational theory of NetKAT (Anderson et. al., POPL 2014), a programming language for software-defined networks. In order to enable program transformations that require additional finite state, such as the classical Böhm-Jacopini theorem of program schematology and the folklore result that every while program can be written with a single while loop, we proposed a general construction for extending KAT conservatively with extra mutable tests [GKM14]. My work also includes extensions of KA with types and products for capturing the theories of mathematical structures, such as CPOs and powerdomains, that arise frequently in the denotational semantics of programming languages [KM13], algebraic nominal theories of unique name generation and binding [KMPST15,KMS15,KMST17], as well as complete equational theories of nonlocal control flow and exception throwing/handling [Mam17]. We have also proposed a probabilistic network programming language, called Probabilistic NetKAT [FKM+16], which extends NetKAT conservatively with new primitives for expressing probabilistic behavior. Probabilistic NetKAT can be used to model and analyze expected delivery in the presence of failures, expected congestion with randomized routing schemes, and expected convergence with gossip protocols.

Theories of partial correctness. Already in an abstract schematic setting, reasoning equationally about mutually recursive programs presents fundamental barriers, as the equational theory of context-free programs is not recursively
enumerable and therefore cannot be axiomatized. I showed in [Mam14, Mam16a] that the Hoare-style partial correctness theory of monadic recursion schemes admits a simple and elegant axiomatization and I identified its computational complexity. Motivated by program synthesis applications, I studied an extension of Hoare logic with dual nondeterminism, that is, the combination of demonic and angelic nondeterminism [Mam15, Mam16b]. I developed a compositional fully abstract denotational semantics for two-player games and used this to obtain the first unconditionally complete Hoare-style calculi for dual nondeterminism and to identify the complexity of the theories. The problem of obtaining complete logical calculi for computational models that are games has been considered for decades, since the classical work of Rohit Parikh on Game Logic. For this contribution I was awarded the EATCS Best Theory Paper award at the ETAPS conference in 2015.

**Future Research**

My future research goal is to provide programming language support for building robust large-scale systems. This includes domain-specific formalisms that ease the development of complex applications and systems, as well as language-based techniques for guaranteeing correctness and efficiency properties. In what follows, I outline some research directions I plan to pursue in the next few years.

**Monitoring of cyber-physical systems (CPS).** The proliferation of cyber-physical systems—such as autonomous vehicles, robotic assistants, and mobile healthcare devices—raises the need for powerful and expressive techniques for the monitoring of runtime functional correctness and performance properties. In order to facilitate the description and synthesis of runtime monitors that continuously check that execution trace of the system, I advocate an approach where the trace is viewed as a stream of data and the monitor is programmed using a streaming domain-specific language. Many existing approaches that are based on quantitative extensions of temporal logic are severely limited in their ability to express complex summary statistics and typically give rise to infeasible synthesis problems. Given the success of using StreamQRE in the context of cardiac arrhythmia detection, I plan to investigate more broadly the question of how to provide programming language support for the development of CPS monitors. A particularly interesting direction concerns mobile sensor devices for healthcare and wellness applications, which present unique challenges regarding energy efficiency and propagation of provenance metadata.

**Optimizable query languages for real-time IoT applications.** The current practice of data processing for analytics is largely based on the relational data abstraction. A key technology that has enabled the success of data processing systems is automatic query optimization, which bridges the gap between a high-level declarative description of the computation and a concrete efficient execution plan. Real-time IoT applications, however, demand optimizing transformations that go beyond operations on static tables, because of the intricate interaction with the temporal dimension. For example, a continuous patient-monitoring query that asks whether a specific sequence of physiological events has occurred during the last ten minutes cannot be optimized using traditional database techniques. A truly principled approach to the problem of optimization for real-time queries is still missing, as the equational theories of such classes of queries are not well understood. My research goal is to develop the algebraic theory of real-time queries by drawing on my Ph.D. work on Kleene algebra, the state-of-the-art equational approach for dealing with sequential programming constructs and the notion of temporal ordering. I believe that a promising approach to the optimization of real-time queries will require the synergy of (1) traditional database methods based on workload statistics, and (2) formal techniques for dealing with the temporal ordering that rely on algebraic reasoning and automata-theoretic minimization algorithms.

**Language support for data science applications.** Data-driven decision making applications are typically enabled by a multitude of tools and systems including statistical software, machine learning libraries, relational databases, and big data systems and query languages. Learning statistical models from past data and using these machine-learned models for inference on new data are two key algorithmic components, which are often incorporated as “black boxes” within larger workflows or high-level queries. This approach limits the opportunities for aggressive query optimization, as well as the ability to explore the accuracy-efficiency tradeoff (which is acceptable for many applications) in a global fashion. I plan to explore the opportunities presented by a deeper integration of machine-learning abstractions within domain-specific languages for data processing. For example, by exposing linear regression as a primitive construct, a query optimizer can turn its application over a sliding-window into an efficient streaming algorithm that requires constant time per element (i.e., independent of the size of the window). In a real-time setting, an adaptive execution engine could automatically lower the accuracy of a model inference procedure to cope with a spike in system load. Such graceful accuracy degradation is often desirable, but requires a richer interface that does not view the inference algorithm merely as a black box.
References


