Programming Language Principles for Distributed Systems

Ankush Das

Programming distributed systems is already very challenging due to the presence of data races and deadlocks; bugs are difficult to detect and reproduce when they only arise in certain thread interleavings. Modern distributed systems such as blockchains and queueing networks have introduced further challenges such as programming in an adversarial setting, inferring execution cost, and accounting for random behavior. Recognizing these challenges, my research goal is to **design novel programming language principles to assist developers in building reliable, efficient, and secure distributed software**.

My unique research approach is to recognize the specific programming challenges in various distributed domains and support developers with logically enriched type systems and complexity analysis tools. As a first step, I designed novel resource-aware session types [Das et al., 2018b ; a] and demonstrated that they serve as a practical foundation for distributed systems with strong static guarantees. Next, I focused on two concrete domains, *blockchains* and *probabilistic systems*, and developed

(i) **Nomos**: a safe smart contract language that expresses and enforces contract protocols, tracks assets via a linear type system, and automatically infers the execution cost of transactions [Das and Hoffmann, 2019; Das et al., 2021a].

(ii) **NomosPro**: a concurrent probabilistic language for the implementation and complexity analysis of randomized distributed algorithms and Markov chains [Das et al., 2021b].

Finally, I recognized that further theoretical innovations were necessary to achieve stronger guarantees in distributed domains. Consequently, I enhanced resource-aware session types with refinements for lightweight verification of distributed programs [Das and Pfenning, 2020a ; b ; c]. Beyond session types, I have designed verification and quantitative analysis tools currently employed at Microsoft Research [Das et al., 2015; Das and Lal, 2017] and Facebook [Das and Qadeer, 2020].

As a long-term goal, I am particularly excited to explore how **programming languages can aid in the design, analysis, and verification of cryptographic protocols**. I believe advanced type systems have the potential to model a large class of adversaries; resource-aware types can further model protocols that rely on computational cryptographic assumptions.

Programming Distributed Systems is Challenging

The design of safe, efficient, and secure software has always been a challenging task. With the proliferation of distributed systems, software development has become even more complex. In addition to the usual challenges, developers must also carefully

- ensure that no bugs manifest in any possible thread interleaving,
- avoid deadlocks and data races, and
- calculate the overall memory and time consumption of the system.

The rise of modern distributed systems such as server farms, cloud computing platforms, and blockchains have further complicated software design by introducing unique challenges. For instance, blockchain is a highly adversarial domain due to its transparency. The state of every decentralized application deployed on the blockchain can be publicly viewed and potentially exploited by malicious attackers. Security vulnerabilities in these applications have caused *losses to the tune of several billions of dollars*.

Advances in programming languages research have always benefited software development, improving its performance, safety, and security. Unfortunately, although analysis tools exist for distributed programs, the most popular and usable tools are still centered around traditional programming languages. With the pervasive usage of distributed systems in software design, there is an **urgent need for formal tools to help with the design, verification, and quantitative analysis of distributed software**.

Unfortunately, the heterogeneous nature of distributed systems has made it extremely difficult to devise a unified computation model to represent and analyze them. A unified model for such systems must account for its numerous components: message buffers, shared memory, critical sections, network delays, packet

loss, system crashes, etc. The domain-specific requirements for each system are also distinct: blockchains require correct handling of assets; server farms expect easy-to-implement efficient parallel computation; cloud computing platforms demand consistent availability. A unified model must also be able to provide these diverse domain-specific guarantees.

Resource-Aware Session Types for Distributed Systems

In response, I have designed novel resource-aware session types that **serve as a sound and practical foundation for distributed systems with strong type-theoretic guarantees** [Das et al., 2018b ; a]. A common theme in a distributed system is *communication between its components*. And often, the type, value, and direction of communication depends on the state of the system. Session types leverage this property by capturing the system state in the type, providing a structured way of prescribing communication protocols. Session types can also guarantee *freedom from deadlocks and data races*. They also possess a confluence property stating that a communicating system converges to the *same final state under all possible thread interleavings*. All these properties greatly benefit programmers and automatically prevent a large class of bugs that can result from communication mismatches.

Unfortunately, simple session types cannot express quantitative properties of a distributed system, such as energy consumption, latency, response time, and throughput. Realizing this, I proposed two extensions to express the *work* [Das et al., 2018b] and *span* [Das et al., 2018a] of parallel computation. Work is defined as the *total number of operations* executed as part of the computation. However, due to parallelism in the system, many of these operations execute simultaneously. Therefore, span is defined as the total *time* of computation taking the parallelism in the system into account. To compute work, my key innovation was that messages and processes both carry an abstract notion of *potential* which is consumed to perform work. To compute span, my key innovation was to introduce operators from temporal logic to capture the timing of message exchanges. Resource-aware session types combine session types with work and span extensions allowing programmers to reason about both qualitative and quantitative aspects of distributed systems.

With a unified computation model in hand, my research focused on two aspects: *(i)* ***applications***: discovering new distributed domains where developers can benefit from programming language support to provide domain-specific guarantees, and *(ii)* ***theory***: further advancing the analysis tools and techniques to provide stronger guarantees. For the former, I designed Nomos and NomosPro applying resource-aware session types to blockchains and probabilistic systems, respectively. For the latter, I designed Rast to integrate refinements into the type system to enable lightweight verification.

Nomos. To aid blockchain developers, I designed Nomos [Das et al., 2021a], a *smart contract language based on resource-aware session types*. Blockchains typically allow execution of smart contracts: programs that implement transaction protocols between distrusting parties. Smart contract programmers need to

(i) ensure that transactions (e.g. auctions, elections) follow their *predefined protocol*,

(ii) compute the *execution cost* of transactions since users issuing them pay for this cost in advance, and

(iii) ensure that assets such as money, valuables, estates, etc. are *not duplicated or discarded accidentally*.

Resource-aware session types are just the right abstraction for implementing smart contracts. Session types statically express contract protocols. Resource-aware types automatically infer the execution cost of transactions leveraging ideas from automatic amortized resource analysis. The built-in linear type system of session types provides a natural representation for assets like money. The Nomos type checker statically enforces the above requirements: protocols are enforced at runtime, bounds inferred are sound and precise, and assets used are neither duplicated nor discarded. Nomos also *significantly develops the theory of programming languages*: integrating session types with functional programming, linear-time type checking to prevent denial-of-service attacks, and an acquire-release discipline to rule out re-entrancy attacks.

NomosPro. To aid developers in probabilistic programming, I introduced novel *probabilistic session types* and implemented them as a concurrent probabilistic language called NomosPro [Das et al., 2021b]. Many practical distributed systems such as Markov chains and randomized algorithms are inherently probabilistic.

NomosPro models such systems by describing probability distributions over message exchanges. The language is further equipped with resource-aware types to automatically infer *expected cost* of programs. NomosPro has a plethora of applications: it automatically infers bounds on the expected cost of randomized distributed protocols; it verifies the limiting distribution of standard Markov chains such as Google's PageRank algorithm, random walks, and dice programs; it determines win probabilities and expected financial gain/loss for probabilistic games such as lotteries and slot machines.

Rast. On the theoretical side, I designed the Rast language which integrates resource-aware session types with *arithmetic refinements* [Das and Pfenning, 2020b]. Fundamental properties of distributed systems are often arithmetic in nature (e.g. size and value of messages exchanged), which cannot be captured by simple session types. Refinement session types [Das and Pfenning, 2020a] employ linear arithmetic to capture these intrinsic attributes and enable lightweight verification of distributed programs. We used refinements for verifying arithmetic properties of concurrent data structures such as stacks, queues, tries, and lists [Das and Pfenning, 2020c]. In the future, I plan to apply refinements for lightweight verification of smart contracts. Along with faculty at CMU, I am also designing a **course using Rast to teach distributed programming**.

Availability to Developers. *My goal is to also make my research ideas available to all developers.* Therefore, Nomos, NomosPro, and Rast are all available as open-source implementations [Das et al., 2019b ; a]. The Nomos language goes a step further and is also available as a well-designed web interface [Das and Hoffmann, 2019] which simulates a blockchain system where users can implement, type check, and submit transactions. Even at a technical level, I have incorporated several design decisions in Nomos and Rast to simplify programming with intuitive surface syntax and precise error messages anticipating the most common programming mistakes. The implementation of Rast won the **best paper award by a junior researcher at FSCD 2020**. It is also equally important that *programming languages and tools are backed by sound theory.* To that end, Nomos, NomosPro, and Rast are also equipped with a mathematical description with a formal type system, execution semantics and theorems establishing their type safety.

Research beyond Session Types.

I have been involved in numerous other projects that provide programming language support to all developers. At Microsoft Research, we designed an angelic verification tool [Das and Lal, 2017; Das et al., 2015] to identify bugs in production software without inundating developers with false alarms. Minimizing false alarms greatly helped with its adoption, and this tool now ships with the Static Driver Verifier tool in Windows. I also assisted with the design and implementation of Resource-Aware ML [Hoffmann et al., 2017] (RaML), a tool for automatic cost analysis of OCaml programs. I employed RaML to infer execution time of OCaml programs on concrete hardware such as Intel x86 and ARM [Das and Hoffmann, 2017]. I demonstrated that the core innovations of the Nomos language are broadly applicable by using **resource-aware types for execution cost analysis of programs in Facebook's Move language** on the Libra blockchain [Das and Qadeer, 2020].

Future Directions.

One day, I hope to make distributed programming as intuitive as ordinary programming for all developers! In particular, I am excited to work in 3 concrete directions to assist programmers in:

(i) design of optimal scheduling policies based on execution cost (short-term)

(ii) automatic synthesis of distributed programs from their specification (medium-term)

(iii) implementation and analysis of cryptographic systems (long-term)

I also plan to develop dependent type systems for verification and quantitative analysis of distributed algorithms such as consensus, commitment, and leader election. Looking beyond session types, I also want to explore other models of distributed computation, such as actor models, Petri nets, and pi-calculus. I would like to examine the tradeoffs of each model, and then decide which model is best suited to what domain, particularly from the viewpoint of verification and quantitative analysis.

Cost Analysis for Optimal Scheduling. Cost analysis can assist developers in designing optimal scheduling policies for their applications. The sequential complexity bounds from Rast can be used to determine whether a new computation needs to be executed in the current or a freshly spawned thread. The parallel complexity bounds from Rast implicitly determine data dependency between threads and can be used to decide the order of thread execution. I would like to implement and evaluate complexity-driven scheduling policies in Rast and evaluate their impact on performance.

Synthesis of Distributed Programs. One effective way of assisting developers is by writing programs for them! Refinement session types can naturally express program specifications. Programs can then be synthesized from their specifications by applying data-driven deep learning techniques. Refinements can also be utilized for semi-automatic synthesis of smart contracts. I believe refinement session types can carry program synthesis to distributed systems.

Type Systems for Cryptographic Protocols. In a recent collaboration, I observed that refinement type systems can neatly represent security protocols. With the recent growth in the complexity of such protocols, developers can greatly benefit from language support while building secure systems. We can also employ type systems to model ill-behaved adversaries and formally verify cryptographic protocols. Furthermore, resource-aware types can provide computational security by modeling adversaries who are capable of only polynomial-time computation. We can also use programming language techniques to demonstrate *universal composability*, which entails that security properties of cryptographic protocols are preserved even when arbitrarily composed with other protocols.

Modern distributed systems are now prevalent in all programming domains. In response, I want to work with researchers from all areas of computer science and beyond to understand their domain-specific requirements and design the best tools and languages to support them. I envision a future where safe, secure, and efficient computer programming is accessible to everyone!

References

Ankush Das and Jan Hoffmann. ML for ML: Learning Cost Semantics by Experiment. In *Tools and Algorithms for the Construction and Analysis of Systems*, 2017.

Ankush Das and Jan Hoffmann. Nomos Web Interface. https://nomos-lang.org, 2019. Accessed: 2020-09-13.

Ankush Das and Akash Lal. Precise Null Pointer Analysis Through Global Value Numbering. In *Automated Technology for Verification and Analysis*, 2017.

Ankush Das and Frank Pfenning. Session Types with Arithmetic Refinements. In *31st International Conference on Concurrency Theory (CONCUR 2020)*, Dagstuhl, Germany, 2020a.

Ankush Das and Frank Pfenning. Rast: Resource-Aware Session Types with Arithmetic Refinements (System Description). In *5th International Conference on Formal Structures for Computation and Deduction (FSCD 2020)*, Dagstuhl, Germany, 2020b.

Ankush Das and Frank Pfenning. Verified Linear Session-Typed Concurrent Programming. In *22nd International Symposium on Principles and Practice of Declarative Programming*, PPDP '20, 2020c.

Ankush Das and Shaz Qadeer. Exact and Linear-Time Gas-Cost Analysis. In *Static Analysis - 27th International Symposium, SAS 2020, Proceedings*, Lecture Notes in Computer Science. Springer, 2020.

Ankush Das, Shuvendu K. Lahiri, Akash Lal, and Yi Li. Angelic Verification: Precise Verification Modulo Unknowns. In *Computer Aided Verification*, 2015.

Ankush Das, Jan Hoffmann, and Frank Pfenning. Parallel Complexity Analysis with Temporal Session Types. *Proc. ACM Program. Lang.*, 2(ICFP), July 2018a.

Ankush Das, Jan Hoffmann, and Frank Pfenning. Work Analysis with Resource-Aware Session Types. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science*, LICS '18, 2018b.

Ankush Das, Farzaneh Derakhshan, and Frank Pfenning. Rast Implementation. https://bitbucket.org/fpfenning/rast/src/master/, 2019a. Accessed: 2019-11-11.

Ankush Das, Jan Hoffmann, Ishani Santurkar, and Stephen McIntosh. Nomos Implementation. https://github.com/ankushdas/Nomos, 2019b. Accessed: 2019-11-11.

Ankush Das, Stephanie Balzer, Jan Hoffmann, Frank Pfenning, and Ishani Santurkar. Resource-Aware Session Types for Digital Contracts. In *34th IEEE Computer Security Foundations Symposium, CSF*, 2021a.

Ankush Das, Di Wang, and Jan Hoffmann. Probabilistic Session Types, 2021b.

Jan Hoffmann, Ankush Das, and Shu-Chun Weng. Towards Automatic Resource Bound Analysis for OCaml. In *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages, POPL*, 2017.