8:00AM-10:00 AM	Registration and Breakfast		
Cubical Type Theory	10:00AM - 10:30AM	An Overview of Computational Cubical Type Theory	Bob Harper
	Interest in higher type theory emerged from the observation that the very limited ITT formalism can be interpreted into structures in which the “identity type” is populated with elements, called identifications, beyond just reflexivity, but nevertheless validating the universal property that maps out of it are determined by their behavior on reflexivity. The first example of this kind was the groupoid model in which types are interpreted as equivalence relations with evidence, with laws such as associativity of composition holding exactly. The univalence axiom proposed by Voevodsky states that identifications of universe elements (types) are given by equivalences, which may be considered as “weak isomorphisms” in which the inverse laws hold only up to further identification. To account for this requires an infinite-dimensional generalization of groupoids to weak ω-groupoids, which provide just such structure. The extension of ITT with the univalence axiom provides a formal axiomatic theory of types as spaces identified up to equivalence, a formalism know as homotopy type theory, or HoTT. From one viewpoint ITT is to be regarded as analogous to first-order logic in that it is merely an axiomatic theory that can be equipped with whatever axioms you may wish to consider. From another viewpoint, including that of the speaker, type theory is rather a theory of computation that provides a theory of mathematical truth grounded in the innate human ability to understand algorithms, a perspective famously introduced by Brouwer long before there were computers. The central question raised by the univalence axiom is whether it can be understood computationally. In this talk I will describe a meaning explanation of type theory of the kind pioneered in the NuPRL Project that extends type theory to higher dimensions by regarding types, and their elements, as cubes presented by cubical programs in terms of a Cartesian coordinate system, with the crucial canonicity property that programs at any dimension that compute a Boolean point will evaluate to true or false. Moreover, the theory has recently been extended to give a (substructural) cubical account of Reynolds’ concept of parametricity, without requiring impredicativity, and admitting an analogue of the univalence principle, called relativity, in which the bridges between types are given by binary relations. The RedTT implementation of these ideas is based on a formal extensional type theory using another cubical structure that, unlike the HoTT formalism, enjoys the property of computational adequacy, meaning that the formalism properly validates both the canonicity property and function extensionality, the only one of its kind in the higher-dimensional context.		
	10:30AM - 11:00AM	A Reynolds' Iceberg	Kristina Sojakova
	We introduce a generalization of cubical sets, which we call cubical categories, and use it to develop a framework for higher-dimensional parametricity, all the way up to and including infinity. Our framework has the crucial property that if a model is p-parametric according to our definition, then it is l-parametric for every l < p, which is a significant generalization of existing definitions of parametricity. We illustrate our framework by giving a corresponding p-parametric model for System F when p <= 3.		

11:00AM - 11:30 AM	Coffee Break 1		
Short Talks	11:30AM - 11:45 AM	Adaptivity Depth Analysis in Adaptive Data Analysis	Jiawen Liu
	An adaptive data analysis is based on multiple queries over a data set, in which some queries rely on the results of some other queries. The error of each query is usually controllable and bound independently, but the error can propagate through the chain of different queries and bring to low generalization error. To address this issue, data analysts are adopting different mechanisms in their algorithms, such as Gaussian mechanism, etc. To utilize these mechanisms in the best way one needs to understand the depth of chain of queries that one can generate in a data analyses. In this work, we define a programming language which can provide, through its type system, an upper bound on the adaptivity depth (the length of the longest chain of queries) of a program implementing an adaptive data analysis. We show how these language can help to analyze the generalization error of two data analyses with different adaptivity structure.		
	11:45AM - 12:00 PM	Formalizing Program Languages Using the coq Proof Assistant	Nathaniel Zogby
	The coq proof assistant is an effective tool for formally verifying the correctness of programming languages. Our research focuses on formalizing PICCO, a system designed to convert a program written in an extension of C into its secure distributed implementation to be run in a distributed environment. It allows programs to label data either public or private using an annotated grammar which is then transformed into a new program using a translated grammar that can be compiled by computation parties into executables. We are using coq to model these two languages along with a function to model the rules by which an input program is translated. Using coq's built in proof capabilities, our goal is to formally prove the soundness of the translation using the reliability of the coq proof assistant.		
	12:00 PM - 12:15 PM	A unified language abstraction for versatile, structured control flow	Yizhou Zhang
	It is currently in vogue for programming languages to acquire support for promises, async–await, and coroutines. These built-in language features allow programmers to organize asynchronous, event-driven code in a more structured way than is possible with just callback functions. However, despite this recent trend in language design, it remains challenging to program control-flow-rich applications: the type systems prevent the programmer from expressing interesting control-flow patterns, and do not enforce enough static checking to help the programmer reason about the complex control flow. We present a new language design that addresses these challenges. Through a novel generalization of algebraic-effect handlers to allow bidirectional effect propagation, the new design offers a unified approach to expressing versatile control flow in a both type-safe and abstraction-safe way.		

12:15 PM - 1:30 PM	Lunch		
Program Analysis	**1:30PM - 2:00PM**	**A Theory of Locality and its Applications**	**Chen Ding**
	Locality is increasingly the primary objective in the organization of a computing system. For software, locality is the quality of its data usage that determines the effectiveness of caching. A new characterization of this relation is the higher-order theory of locality (HOTL). This talk will introduce the theory and demonstrate its use in modeling and optimizing cache performance.		
	2:00PM - 2:30PM	**Relational Cost Analysis for Functional-Imperative Programs**	**Weihao Qu**
	The difference in the evaluation costs of two programs are called relative cost. Relational cost analysis aims at formally establishing bounds on the relative cost of two programs. As a particular case, relational cost analysis can be used to establish bounds on the difference in the evaluation cost of the same program on two different inputs. One way to perform relational cost analysis is to use a relational type-and-effect system that supports reasoning about relations between two executions of two programs. Building on this basic idea, we present a type-and-effect system, called ARel, for reasoning about the relative cost of array-manipulating, higher-order functional-imperative programs. The key ingredient of our approach is a new lightweight type refinement discipline that we use to track relations (differences) between two mutable arrays. This discipline combined with Hoare-style triples built into the types allows us to express and establish precise relative costs of several interesting programs which imperatively update their data. We have implemented ARel using ideas from bidirectional type checking.		

2:30PM - 3:00PM	**Coffee Break 2**		
Concurrency	**3:00 PM - 3:30PM**	**A Tour of Gallifrey, a Language for Geodistributed Programming**	**Matthew Milano**
	Programming efficient distributed, concurrent systems requires new abstractions that go beyond traditional sequential programming. But programmers already have trouble getting sequential code right, so simplicity is essential. The core problem is that low-latency, high-availability access to data requires replication of mutable state. Keeping replicas fully consistent is expensive, so the question is how to expose asynchronously replicated objects to programmers in a way that allows them to reason simply about their code. We propose an answer to this question in our ongoing work designing a new language, Gallifrey, which provides orthogonal replication through restrictions with merge strategies, contingencies for conflicts arising from concurrency, and branches, a novel concurrency control construct inspired by version control, to contain provisional behavior.		
	3:30PM - 4:00 PM	**Manifest Deadlock-Freedom for SharedSession Types**	**Stephanie Balzer**
	Shared session types generalize the Curry-Howard correspondence between intuitionistic linear logic and the session-typed pi-calculus with adjoint modalities that mediate between linear and shared session types, giving rise to a programming model where shared channels must be used according to a locking discipline of acquire-release. While this generalization greatly increases the range of programs that can be written, the gain in expressiveness comes at the cost of deadlock-freedom, a property which holds for many linear session type systems. In this talk I develop a type system for logically-shared sessions in which types capture not only the interactive behavior of processes but also constrain the order of resources (i.e., shared processes) they may acquire. This type-level information is then used to rule out cyclic dependencies among acquires and synchronization points, resulting in a system that ensures deadlock-free communication for well-typed processes in the presence of shared sessions, higher-order channel passing, and recursive processes. I illustrate the approach on a series of examples, showing that it rules out deadlocks in circular networks of both shared and linear recursive processes, while still being permissive enough to type concurrent implementations of shared imperative data structures as processes.		

4:00 PM - 4:30PM	**Coffee Break 3**		
Applications	**4:30PM - 4:45PM**	**Probabilistic Programming Languages for Autonomous Systems under Uncertainty**	**Sayed Mahdi Shamsi**
	Robotics and autonomy has emerged as an interdisciplinary field of research in science and technology. However, many of the tools and approaches utilized are adopted from other fields and not customized to the needs of modern robotics. Specifically in the context of robot programming, roboticists have primarily solved problems by proposing direct solutions composed of procedures that generate outputs from inputs, i.e. using imperative programming. Alternatively, we propose using tools and abstractions that address the problem by modeling the domain, i.e. the variables and how they interact, and using probabilistic inference executed by a runtime environment or compiler. In these approaches, a system model is defined as a program and specific problems are defined as subsets of variables in the domain to be inferred. This allows for 1) solving a range of problems in the same domain with trivial changes to the code, e.g. localization, mapping, exploration, etc; 2) black-boxing the inference algorithm and updating seamlessly to different methods. We present a prototype system for programming robotic systems using probabilistic programming languages (PPL). First, we show how a large number of robotic problems are naturally expressed as probabilistic inference problems. Second, we show how we could achieve additional benefits such as automatic error detection and parameter calibration in robotic systems simply by defining an inference profile in the domain. Lastly, we show how motion/path planning could be achieved by using off-the-shelf inference tools embedded in most PPLs. This approach provides a programming framework for building complex autonomous systems that have to operate under uncertainty. By directly modeling the interconnection of uncertain components, decoupled from the inference engine, the design benefits from robustness, re-usability, upgradability, and ease of specification.		
	4:45PM - 5:15PM	**Making Federated, Verified Software-Defined Networks with Proof-Carrying Network Code**	**Joshua Robbins**
	Proof-Carrying Network Code (PCNC) is a novel solution for creating formally verified, federated SDNs, currently in development by a joint team of the Rochester Institute of Technology and the University of Vermont. It makes use of the Frenetic SDN controller (developed at Cornell!), its NetKAT language, and proof-carrying code to allow us to make computer networks that are provably problem-free.		