Biological systems can be seen as reactive systems, namely dynamical systems which interact with their environment by receiving and transmitting signals. Gene regulatory networks (GRNs) are systems for which proteins are produced and led to the environment in response to proteins coming from the environment. Moreover, GRNs are subject to evolution which vary their behaviour.

**How to reason about evolving reactive systems?**

We use parametric model checking to compute the probability of an evolving system to satisfy a property, i.e. its mutational robustness.

![Gene regulatory network](image)

**Syntax: Gene regulatory network**

**Semantics: Markov chain of transition systems**

M. Giacobbe, T. Petrov, and external collaborators

**Computer-aided Concurrent Programming**

Typical concurrency bugs such as data races are caused by subtle synchronization errors. We aim to make concurrent programming easier by automatically synthesizing tricky synchronization primitives (locks). The programmer programs assuming a non-preemptive (cooperative) scheduler that does not interrupt threads. The synthesis procedure makes the program safe for a preemptive scheduler.

**How to automatically infer locks to ensure correctness?**

**How to infer minimal locks required for correctness?**

Our solution is based on a finitary abstraction, an automata-theoretic language inclusion check, trace-generalization, rewrite rules for synchronization inference and constraint-solving.

![Flow-pipe construction](image)

**Flow-pipe construction**

S. Bogomolov, M. Giacobbe, H. Kong, J. Ruess, and external collaborators

**Software Systems**

We develop automatic techniques to verify and synthesize hybrid systems. Hybrid systems consist of discrete software controllers interacting with continuous physical environments.

**How to automatically compute a system abstraction that provides enough information to efficiently check a property?**

In our work, we build system abstractions using flow-pipe construction, barrier-certificates and moment closures.

![Barrier certificate](image)

**Barrier certificate**

S. Bogomolov, M. Giacobbe, H. Kong, J. Ruess, and external collaborators

**Performance-Aware Software Synthesis**

Software systems are built using components such as counters, pools, and queues. Components with strong guarantees (e.g., queues) are easier to reason about, while those with relaxed guarantees (e.g., pools) provide better performance.

**How to ease the programming burden by automatically optimizing the performance of component-based programs?**

We allow the programmer to write a program \( P \) using components \( \{A_i\} \), and automatically synthesize glue code that dynamically redirects calls to more efficient components \( \{B_i\} \) while ensuring that \( P(\{A_i\}) \) is observationally equivalent to \( P(\{B_i\}) \).

![Synthesis](image)

**Synthesis**

B. Kragl, A. Kupriyanov, and R. Samanta

**Probabilistic Systems**

Statistical model checking (SMC) is an approach to verification of probabilistic systems, where we sample from the system, and apply statistics to infer conclusions. We propose an SMC algorithm for unbounded properties, e.g.:

**What is the probability we reach state \( u \) once?**

**How about infinitely many times?**

Our key idea is to detect recurrent states by observing simulations.

![Recurrence](image)

S. Bogomolov, M. Giacobbe, H. Kong, J. Ruess, and external collaborators

http://ist.ac.at/