Colloquium de Jim Larus

23 octobre 2018

`http://www.lip6.fr/colloquium/`

Programme

Master classes **de 14:00 à 16:00 (Salle 24-25/405)**

14:00–14:30	Darius Mercadier (WHISPER, LIP6–INRIA) *Usuba, optimizing and trustworthy bitslicing compiler*
14:30–15:00	Ismail Lahkim Bennani (PARKAS, ENS–INRIA) *QuickCheck testing of hybrid systems*
15:00–15:30	Ilyas Toumlilt (DELYS, LIP6–INRIA) *Geo-replication all the way to the client machine*
15:30–16:00	Raphaël Monat (APR, LIP6) *Static analysis by abstract interpretation of dynamic programming languages*

Cocktail à 17:15 (au pied de l'Amphi 25)

Colloquium de Jim Larus à 18:00 (Amphi 25)

Programming Non-Volatile Memory

Master classes – Résumés

- **Usuba, optimizing and trustworthy bitslicing compiler**

  Darius Mercadier (WHISPER, LIP6–INRIA)

  Bitslicing is a standard technique to improve the performance of certain cryptographic algorithms by exploiting data parallelism while making them, de facto, resilient to cache-timing attacks. To do so, bitsliced implementations turn lookup tables (such as S-boxes) into boolean functions, leading to a significant code blow-up and making it hard to write, debug and optimize manually. Usuba is a synchronous dataflow programming language we designed (based on an original idea from X. Leroy) to both specify and implement bitsliced algorithms. Usuba programs compile into bitsliced and optimized C codes, exploiting platform-specific SIMD extensions such as Intel's SSE and AVX, ARM's Neon or IBM's AltiVec.

  – Darius Mercadier, Pierre-Évariste Dagand, Lionel Lacassagne, and Gilles Muller. 2018. Usuba, Optimizing & Trustworthy Bitslicing Compiler. In Workshop on Programming Models for SIMD/Vector Processing, Feb. 2018, Vienna, Austria. `https://hal.archives-ouvertes.fr/hal-01657259`

- **QuickCheck testing of hybrid systems**

  Ismail Lahkim Bennani (PARKAS, ENS–INRIA)

  The main goal of our work is to make a testing tool for hybrid systems.

  We are exploring an approach called property-based testing. It is inspired by QuickCheck [CH00], a Haskell library for random testing. Using this library, you can write an executable specification (a property) of the program to be tested as a boolean function, and then test that specification with automatically generated entries.

  The problem becomes more complicated in the context of hybrid systems: the input and output spaces of the programs are dense. Several tools such as Breach [Don10] and S-TaLiro [ALFS11] tackle this problem for Simulink systems[1]; they use piecewise constant functions as inputs and Metric Interval Temporal Logic (MITL) formulas as properties. However, the use of temporal logics requires an offline verification and a discrete approximation of the computed signals.

  We want to create an online testing tool that uses formally defined properties and is expressive enough to be used in real-life use-cases.

  During a previous internship, I developed a first prototype written in the synchronous language Zélus [BP13], a language used to program and simulate hybrid systems. It consists

[1] `https://www.mathworks.com/products/simulink.html`

of a library of input generators inspired by the Lutin [RRJ08] programming language and a library of synchronous observer to monitor the outputs.

This prototype has been compared to state-of-the-art tools on an automatic transmission controller model[2] (a demo example of Simulink, reimplemented in Zélus). On this particular example, our random testing tool ran faster than the others.

ALFS11 Yashwanth Annpureddy, Che Liu, Georgios Fainekos, and Sriram Sankaranarayanan. S-taliro: A tool for temporal logic falsification for hybrid systems. In Parosh Aziz Abdulla and K. Rustan M. Leino, editors, Tools and Algorithms for the Construction and Analysis of Systems, pages 254–257, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

BP13 Timothy Bourke and Marc Pouzet. Zélus: A Synchronous Language with ODEs. In Calin Belta and Franjo Ivančić, editors, HSCC - 16th International Conference on Hybrid systems: computation and con- trol, Proceedings of the 16th International Conference on Hybrid systems: computation and control, pages 113–118, Philadelphia, United States, April 2013. Calin Belta and Franjo Ivančić, ACM.

CH00 Koen Claessen and John Hughes. Quickcheck: A lightweight tool for random testing of haskell programs. 46, 01 2000.

Don10 Alexandre Donzé. Breach, a toolbox for verification and parameter synthesis of hybrid systems. In Tayssir Touili, Byron Cook, and Paul Jackson, editors, Computer Aided Verification, pages 167–170, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

RRJ08 Pascal Raymond, Yvan Roux, and Erwan Jahier. Lutin: a language for specifying and executing reactive scenarios. EURASIP Journal on Embedded Systems, 2008, 2008. `http://jes.eurasipjournals.com/content/2008/1/753821`

- **Geo-replication all the way to the client machine**

  Ilyas Toumlilt (DELYS, LIP6–INRIA)

  Cloud-scale services improve availability and latency by geo-replicating data in several data centers (DC) across the world. Nevertheless, the closest DC is often still too far away for an optimal user experience. To remain available at all times, client-side applications need to cache data at client machines, caching data at client machines can improve availability and latency for many applications, and also allow for temporary disconnection. This approach is used in many recent cloud services, like Google Drive RT API or Mobius, where developers implement caching and buffering at application level, but it doesn't ensure system-wide consistency guarantees. Our system, EdgeAnt, brings geo-replication consistency and availability guarantees all the way to the edge client machine, offers a simple protocol buffer interface for developers, and support rich data types semantics.

[2] `https://www.mathworks.com/help/simulink/examples/modeling-automatic-transmission-controller.html`

- **Static analysis by abstract interpretation of dynamic programming languages**

  Raphaël Monat (APR, LIP6)

  Dynamic programming languages are increasingly popular, due to their powerful, high-level syntax, their flexibility and the presence of numerous libraries. Examples of dynamic programming languages include JavaScript and Python, both standing in the top 3 languages most used on Github. When analyzing programs statically (i.e, without executing the real program) to find bugs however, it is more difficult to work on such programming languages: a lot of information (such as types of values) is implicit, and only discovered at runtime. The goal of my PhD is to develop such analyses for Python programs using the framework of abstract interpretation. My work is currently focused on discovering type information for Python programs. Future work aims at developing new approaches to analyze real-world Python programs, by performing modular interprocedural analyses, inferring library calls orders, generating stubs automatically and performing multilingual analyses. This talk includes a brief introduction to Python from a PL perspective, and will focus on ongoing and future work.