FLOC 2018:
FEDERATED LOGIC CONFERENCE 2018

MLP ON WEDNESDAY, JULY 18TH

09:00-10:30 Session 125G

CHAIR: *Bruno Marnette*

LOCATION: Blavatnik LT1

09:00 *Viktor Kuncak*

Opening and demos

ABSTRACT. Informal opening of the event: a chance to start networking and ask for demos.

09:45 *Earl Barr*

Bimodal Software Engineering

ABSTRACT. Source code is bimodal: it combines a formal algorithmic channel and a natural language channel of identifiers and comments. To date, most work has focused exclusively on a single channel. This is a missed opportunity because the two channels interact: the natural language often explains or summarizes the algorithmic channel, so information in one channel can be used to improve analyses of the other channel. A canonical bimodal fact is identifier named “secret” (NL channel) printed to the console (AL channel). To exploit such bimodal facts, one must overcome two challenges: find cross-channel synchronisation points and handle noise in the form of ambiguity in the NL channel and

imprecision in the AL channel. Thus, bimodality is a natural fit for machine learning. I will present RefiNym, a bimodal analysis that models code with *name-flows*, a dataflow graph augmented to track identifier names. Conceptual types are logically different types that do not always coincide with program types. Passwords and URLs are example conceptual types that can share the program type String. RefiNym is an unsupervised method that mines a lattice of conceptual types from name-flows and reifies those conceptual types into distinct nominal types. For the String type, we show that RefiNym minimises co-occurrence of disparate conceptual types in the same scope by 42%, thereby making it harder for a developer to inadvertently introduce an unintended flow.

10:30-11:00 Coffee Break

11:00-12:30 Session 127G

CHAIR: *Viktor Kuncak*

LOCATION: Blavatnik LT1

11:00 *Jules Villard*

Static Analysis for Developer Efficiency with Infer

ABSTRACT. **Infer** is an open-source static analysis tool for Java, C, C++, and Objective-C. **Infer** has been successfully deployed at Facebook, where it identifies hundreds of potential bugs per month in mobile apps and backend code. **Infer** uses AI (Abstract Interpretation) and ML (more precisely the OCaml implementation) to analyse source code. This talk will present **infer** and attempt to draw bridges between infer and other AI/ML techniques.

11:45 *Liam Atkinson*

Learning to Type

ABSTRACT. JavaScript developers enjoy the freedom of not having to specify types in their source code, but JavaScript is also notoriously harder to refactor or debug than a statically typed language. Type systems such as FlowType or TypeScript have been designed to extend JavaScript but adding type annotations to an existing code base can be a laborious task. In this paper, we present a Deep Learning system able to facilitate this process by accurately suggesting probable type annotations to all the relevant nodes of a program.

We generate our training set using runtime information, observing the types of various expressions during their execution. We represent programs as graphs using abstract syntax trees enriched with string information (e.g. variable names) and over 90 additional types of edges, encoding scope information, data flow, and the relative position of different nodes. We then feed these graphs into a tailored neural network

architecture designed to propagate information efficiently across edges, support multiple edge types, and pay attention to different edges in different contexts. We finally use this attention mechanism to explain back to the user what drove each prediction.

Our experiments show that a good accuracy can be obtained with a relatively small amount of training data, suggesting that applying Deep Learning to the analysis of programs is somewhat easier than classical applications, where large amounts of training data are usually required.

12:30-14:00 Lunch Break

14:00-15:30 Session 128G

CHAIR: *Bruno Marnette*

LOCATION: Blavatnik LT1

14:00 *Rishabh Singh*

Neural Meta Program Synthesis

ABSTRACT. The key to attaining general artificial intelligence is to develop architectures that are capable of learning complex algorithmic behaviors modeled as programs. The ability to learn programs can allow these architectures to learn to compose high-level abstractions that can lead to many benefits: i) enable neural architectures to perform more complex tasks, ii) learn interpretable representations (programs which can be analyzed, debugged, or modified), and iii) better generalization to new inputs (like algorithms). In this talk, I will present some of our recent work in developing neural architectures for learning programs from examples, and also briefly discuss other applications such as program repair and fuzzing that can benefit from such neural program representations.

14:45 *Martin Vechev*

Learning to Analyze Programs at Scale

ABSTRACT. I will present two new results on machine learning-based program analysis. The first direction involves learning static analyzers from a given dataset of programs and is based on counter-example guided synthesis, decision tree learning and adversarial perturbations. The second direction involves learning rules that pinpoint program issues (e.g., security violations), and is based on learning from large datasets of program changes by using semantic abstractions and hierarchical clustering. In both cases, I will show the methods successfully found issues missed by state-of-the-art, manually crafted systems.

15:30-16:00 Coffee Break

16:00-18:00 Session 130F

CHAIR: *Bruno Marnette*

LOCATION: Blavatnik LT1

16:00 *Michael Pradel*

DeepBugs: A Learning Approach to Name-based Bug Detection

ABSTRACT. Natural language elements in source code, e.g., the names of variables and functions, convey useful information. However, most existing bug detection tools ignore this information and therefore miss some classes of bugs. The few existing name-based bug detection approaches reason about names on a syntactic level and rely on manually designed and tuned algorithms to detect bugs. This talk presents DeepBugs, a learning approach to name-based bug detection, which reasons about names based on a semantic representation and which automatically learns bug detectors instead of manually writing them. We formulate bug detection as a binary classification problem and train a classifier that distinguishes correct from incorrect code. To address the challenge that effectively learning a bug detector requires examples of both correct and incorrect code, we create likely incorrect code examples from an existing corpus of code through simple code transformations. A novel insight learned from our work is that learning from artificially seeded bugs yields bug detectors that are effective at finding bugs in real-world code. We implement our idea into a framework for learning-based and name-based bug detection. Three bug detectors built on top of the framework detect accidentally swapped function arguments, incorrect binary operators, and incorrect operands in binary operations. Applying the approach to a corpus of 150,000 JavaScript files yields bug detectors that have a high accuracy (between 89% and 95%), are very efficient (less than 20 milliseconds per analyzed file), and reveal 102 programming mistakes (with 68% true positive rate) in real-world code.

16:45 *Ian Wright*, *Jean Helie* and *Albert Ziegler*
Measuring software development productivity: a machine learning approach
SPEAKER: *Ian Wright*

ABSTRACT. We apply machine learning to version control data to measure software development productivity. Our models measure both the quantity and quality of produced code. Quantity is defined by a model that predicts the labor hours supplied by the `standard coder' to make any code change, and quality is defined by a model that predicts the distribution of different kinds of problems identified by a static code analysis tool.

17:15 *Ezra Winston*, *Bhuwan Dhingra*, *Kathryn Mazaitis*, *Graham Neubig* and *William Cohen*
Answering Cloze-style Software Questions Using Stack Overflow
SPEAKER: *Ezra Winston*

ABSTRACT. Modern Question Answering (QA)

systems rely on both knowledge bases (KBs) and unstructured text corpora as sources for their answers. KBs, when available, generally offer more precise answers than unstructured text. However, in specialized domains such as software engineering, QA requires deep domain expertise and KBs are often lacking. In this paper we tackle such specialized QA by using both text and semi-structured knowledge, in the form of a corpus of entity-labeled documents. We propose CASE, a hybrid of an RNN language model and an entity co-occurrence model, where the entity co-occurrence model is learned from the entity-labeled corpus. On QUASAR-S, a dataset derived from Stack Overflow consisting of Cloze (fill-in-the-blank) software questions and a corpus of tagged posts, CASE shows large accuracy gains over strong baselines.

19:15-21:30 Workshops dinner at Magdalen College

Workshops dinner at Magdalen College. Drinks reception from 7.15pm, to be seated by 7:45 (pre-booking via FLoC registration system required; guests welcome).

LOCATION: Magdalen College
