LIGHTWEIGHT MULTILINGUAL SOFTWARE ANALYSIS

Damian M. Lyons[1], Anne Marie Bogar[1] and David Baird[2]

[1]Department of Computer & Information Science, Fordham University, New York NY USA

[2]Bloomberg L.P., New York NY USA

{dlyons,abogar}@fordham.edu, dbaird16@bloomberg.net

The authors acknowledge the contributions of Bruno Vieira, Sunand Raghupathi and Nicholas Estelami in building MLSA Tools

The authors are partially supported by grant DL-47359-15016 from Bloomberg L.P.

Motivation

Companies with a large software base:

- have to manage software architectures and libraries that
  - *Are written by different developers (in & out house)*
  - *Are developed over many decades*
  - *Are often in different languages*
- In order to enforce
  - *Security*
  - *Efficiency*
  - *Quality Metrics*

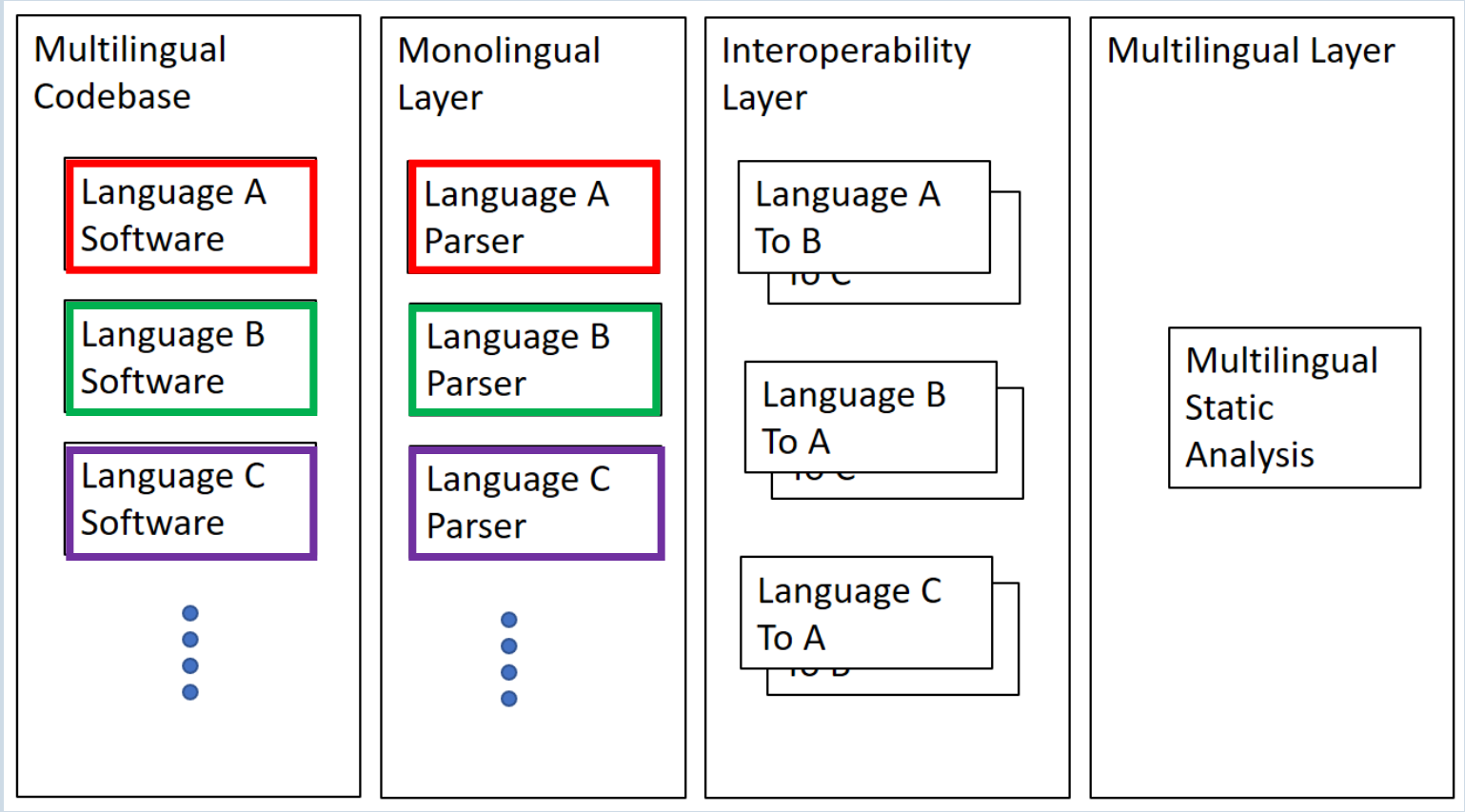
Why is Multilingual Software a Problem?

- A multilingual codebase gives rise to many software engineering issues, including
  - *Redundancy, e.g., procedures in several different language libraries for the same functionality, necessitating refactoring*
  - *Debugging complexity as languages interact with each other in unexpected ways*
  - *Security issues relating to what information is exposed when one language procedure is called from another*
    - *Problem of opacity when one language calls another*

Approaches to the problem

- Development tools tend to be language specific, with some cross-platform functionality. (e.g., *Checkmarx* offers static analysis)
- One approach is to instead use a versatile monolingual environment
- ‘Reverse engineering’ approach: leverage a metalanguage, e.g., Rascal
- Proposed Approach: Multilingual will always be with us. Develop lightweight tools to process multilingual codebase into **unified common representations** such as callgraphs, flowgraphs, etc.

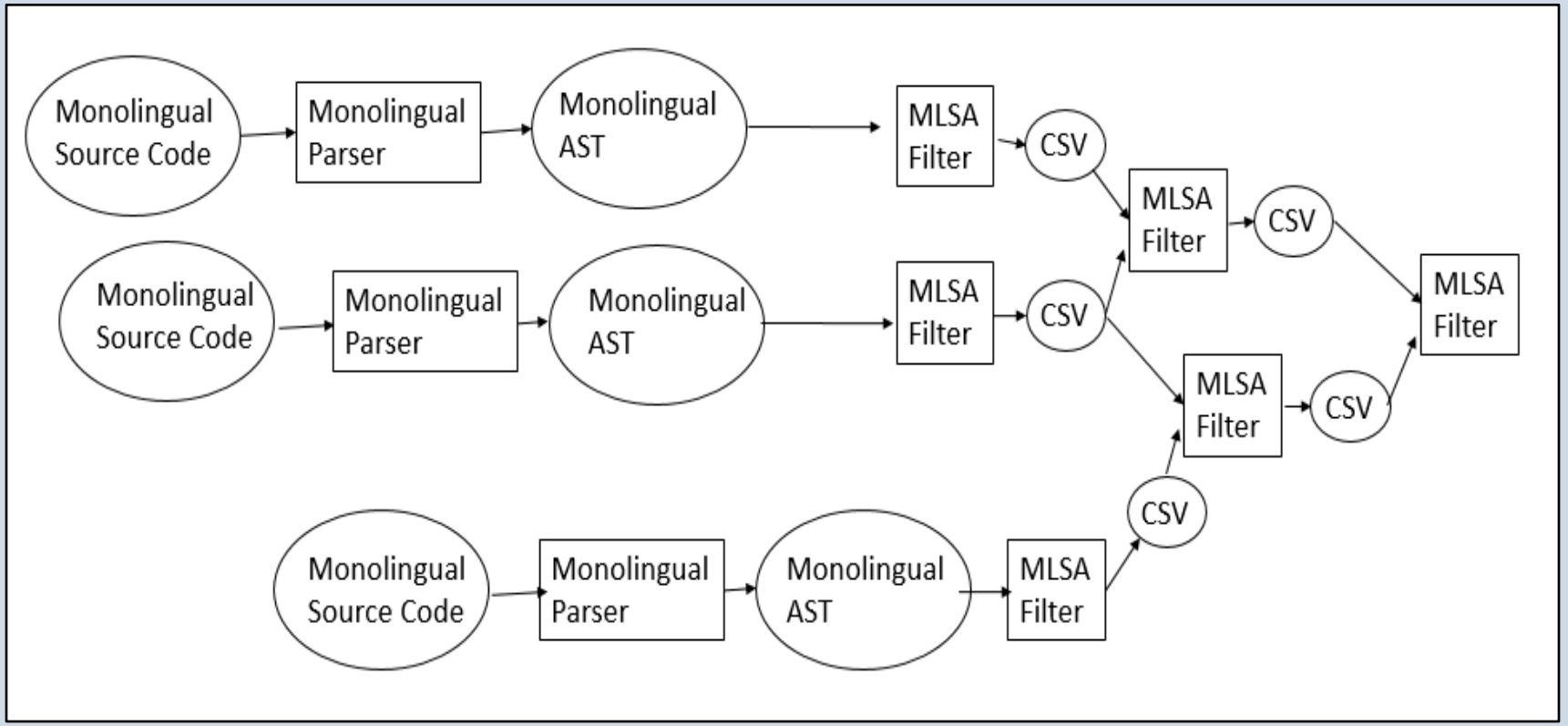
MultiLingual Static Analysis (MLSA) Architecture



Data flows left to right

MLSA

Example



All intermediate data store in the form of CSV “table” files

MLSA Architecture

- Modularity:
  - *Adding extra languages just requires adding new monolingual filters for the language.*
  - *Modifying analyses just requires reconfiguration of the analysis pipeline.*
  - *New analysis filters can build on existing filters (and their tabular CSV output)*
- Computational efficiency:
  - *The policy of the pipelines was chosen to make parallelism and dependency explicit to allow for mapping to a cluster/cloud.*

Call Graphs

- Call graph analysis (CGA) is a useful software engineering tool
- In particular, for multilingual code, the call graph can be used to investigate the boundary line between languages, a boundary that is opaque in many tools.
- Example: a C program PC may call a Python PP procedure in addition to many C procedures.
  - *All of PC's calls are reviewed and safe.*
  - *However PP may opaquely call procedures from PC; these may or may not be safe.*

MLSA

Interoperability API

- How calls from one language are made from another
- Consider the C/Python boundary API (Python.h). Python code can be called from C non-interactively in the following ways (each of which have several variants and may require setup code):
  - *PyRun_SimpleString(pyCodeString)*
  - *PyRun_SimpleFile(filePtr, fileName)*
  - *PyObject_CallObject(pFunc,pArgs)*

Terminology

- Program S_c is a set of (ℓ, b) basic block b with line number ℓ.
- The set B of elementary statements includes a procedure call statement, and for (ℓ, b), $b \in B$ procedure call, we define:
  - *target(b): name of the called procedure*
  - *arg(b)=a_0,...,a_n: arguments of the call*
  - *RDA(p,X, ℓ) = {(x, ℓ')|x$\in$X} is the line number ℓ' of the last assignment in procedure p for each variable x.*

The PyRun_SimpleFile API call

If (ℓ, b), $b \in B$, $target(b)$=PyRun_SimpleFile

For $(x, \ell') \in RDA(p, \{a_0\}, \ell)$, with $arg(b)=a_0,\ldots,a_n$

Calculate $y=Eval(x, \ell')$, and if $y \neq \varnothing$,

Add $(y, \varnothing)$ to V_C and $((p, \alpha),(y, \varnothing))$ to E_C

The PyObject_CallObject API

If (ℓ, b), $b \in B$, $target(b)$=PyObject_CallObject

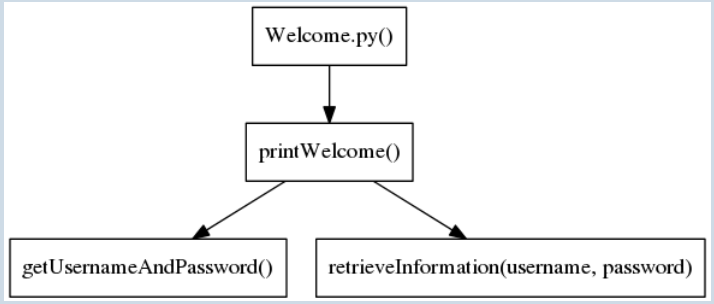
For $(x_i, \ell') \in RDA(p, \{a_0\}, \ell)$, with $arg(b)=a_i$ $i=0..n$

Calculate $y_i = Eval(x_i, \ell')$, and if all $y_i \neq \varnothing$,

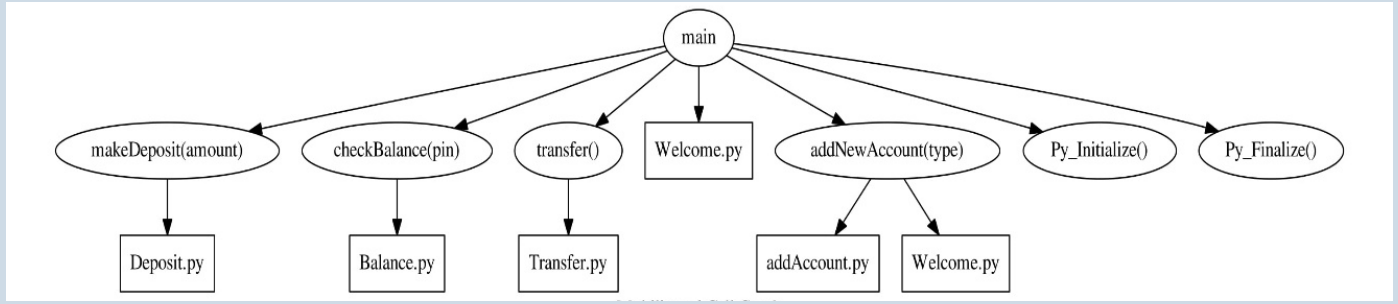
Add $(y_0, y_1, \ldots, y_n)$ to V_C , $((p, \alpha), (y_0, y_1, \ldots, y_n))$ to E_C

Example

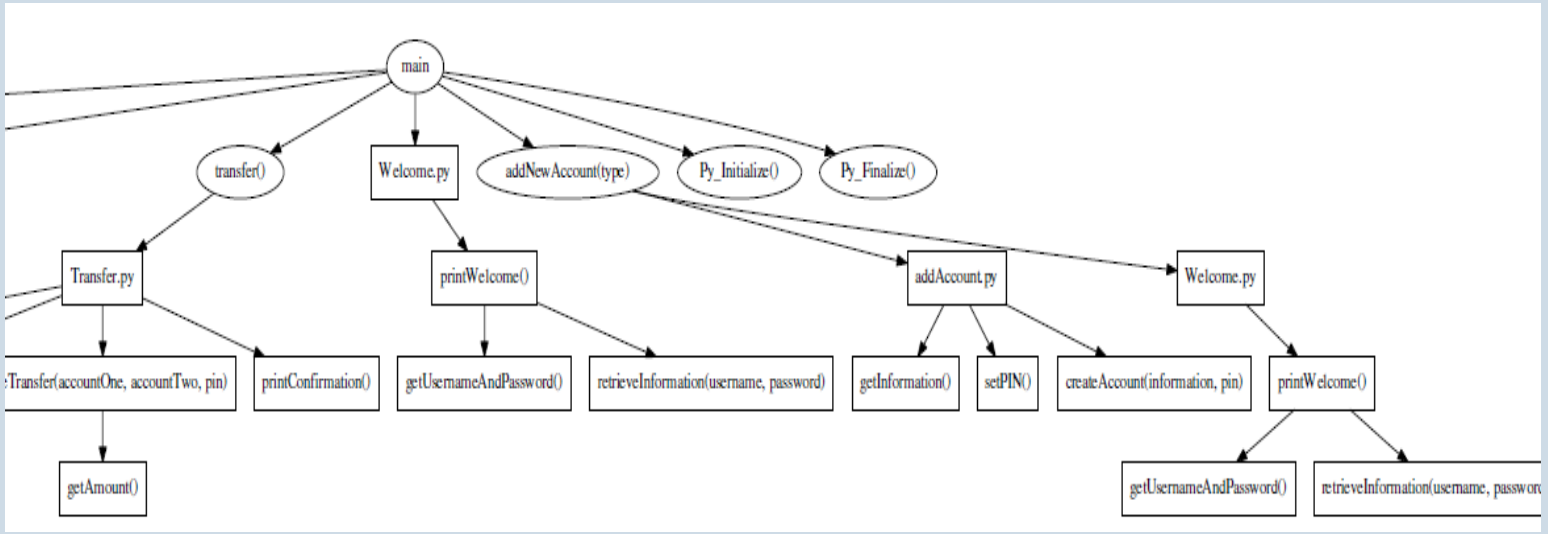
Python library



C Codebase calling python



Multilingual Call Graph



Status

- Filter modules
  - *Call Graph Generators for C/C++, Python, and JavaScript*
    Status: Non OO calls only, but OO calls statically identified
  - *Interoperability filters: PyViaC/C++, PyViaJs, JsViaPy*
    Status: One or two API calls implemented for each; none complete.
  - *Flow Graph Generators for C/C++*
    Status: Partial implementation.
  - *Assignment Collector/Static Evaluator for C*
    Status: Only evaluates literal assignments
  - *Reaching Definitions Module*
    Status: Tested on C but is language independent

Collaboration

- MLSA is open-source and freely available at
  - *TWIKI – https://goo.gl/6mvyNS*
  - *GitHUB – https://git.io/MLSA*
  - *Email – multilingualsa@gmail.com*
- We are actively seeking open source collaborators and participants
- There are videos available at https://goo.gl/g8ra15 showing
  - *How to download and install MLSA*
  - *How to test on the test folders provided*
  - *How to add to MLSA*

Conclusion & Future Work

- We propose an architecture comprised of monolingual filter programs that analyse single language AST and identify the cross-language boundary. The filters generate language independent information in CSV format. Additional multilingual filters operate on the CSV files in pipelines. This architecture has advantages of modularity and efficiency and is open-source friendly.
- Future work includes
  - *extension of the RDA analysis for more complex interoperability APIs, including use of DATALOG for analysis.*
  - *More extensive testing and comparisons of the CGs generated with those for existing CG tools*

MLSA