

Nagini: A Static Verifier for Python

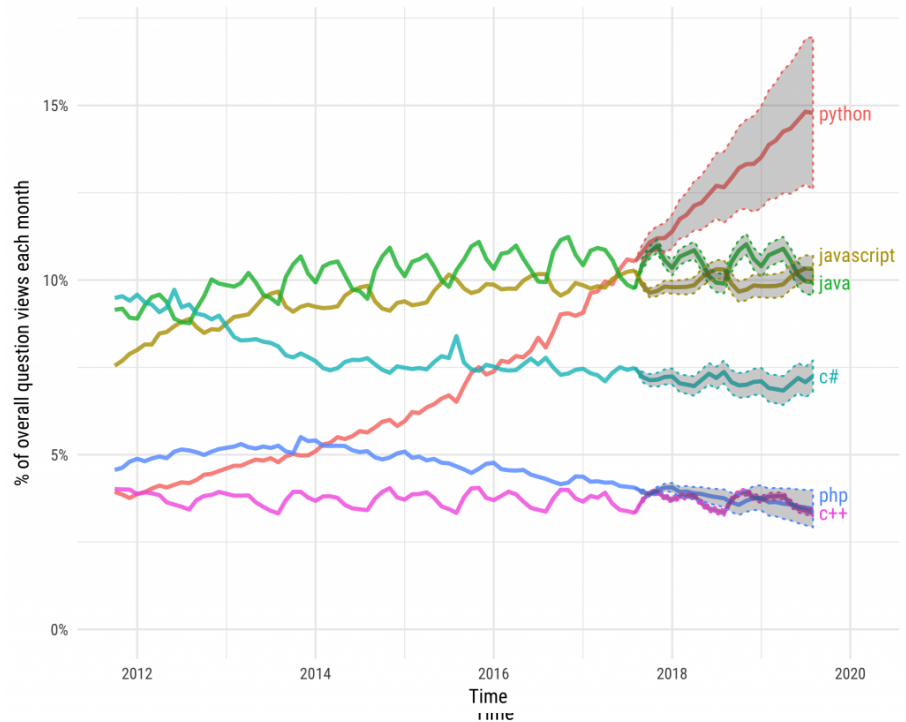
Marco Eilers

Peter Müller



Projections of future traffic for major programming languages

Future traffic is predicted with an STL model, along with an 80% prediction interval.



Nagini: Sound Verification for Concurrent Python

Implicit dynamic frames

Scalability,
verify libraries

Finite blocking

Input/output

High level
abstractions

Goal: **Modularly** specify and verify **complex properties**
of **real-world user code**

Support large
language subset

Inheritance,
exceptions,
...

Dynamic
field addition

Focus on
typical features

Restrict metaclasses,
decorators, magic
methods

Built-in support
for common
libraries

Nagini: Sound Verification for Concurrent Python

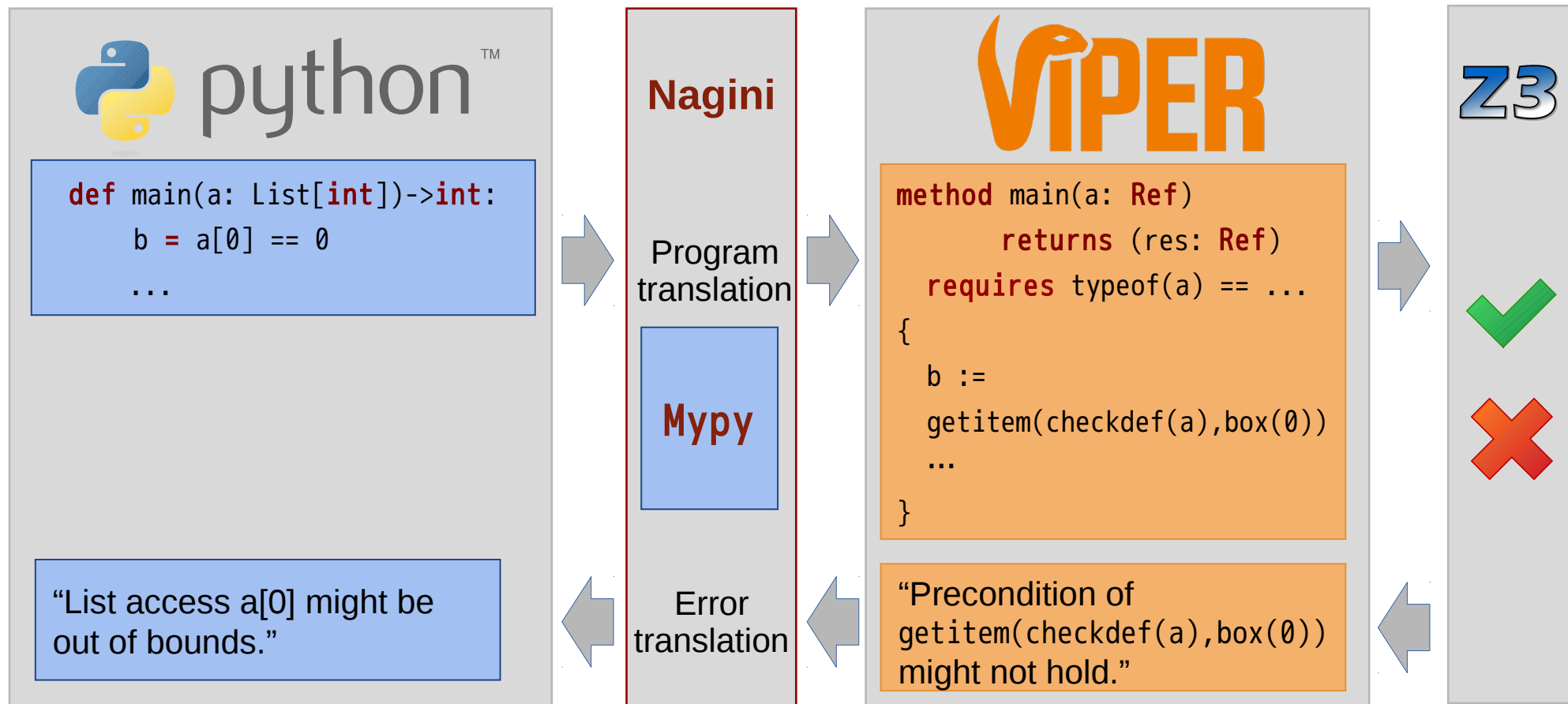
```
from nagini_contracts.contracts import *

def indexOf(l: List[int], i: int, val: int) -> int:
    Requires(i >= 0 and i < len(l))
    Requires(list_pred(l))
    Requires(MustTerminate(len(l) - i))
    Ensures(list_pred(l))
    Ensures(Implies(Result() != -1,
        Forall(range(i, Result()), lambda j: l[j] != val) and
        l[Result()] == val))
    if l[i] == val:
        return i
    if i >= len(l) - 1:
        return -1
    return indexOf(l, i + 1, val)
```

- Memory Safety
 - No runtime errors
 - No data races
 - ...
- Functional properties
- Termination

- Finite blocking
 - Deadlock freedom
 - ...
- Input/Output behavior
 - Allowed I/O
 - Liveness

Implementation



Conclusion

- In the paper: Experimental evaluation
- Available open source
 - www.github.com/marcoeilers/nagini
 - Online version available:
viper.ethz.ch/nagini-examples

Nagini Examples

Binary Search Tree

Binary search tree implementation from interactivepython.org

```
1 from typing import Optional
2 from nagini_contracts.contracts import *
3
4
5 class TreeNode:
6     def __init__(self, key: int, val: str, left: 'TreeNode'=None,
7                 right: 'TreeNode'=None, parent: 'TreeNode'=None) -> None:
8         self.key = key
9         self.payload = val
10        self.leftChild = left
11        self.rightChild = right
12        self.parent = parent
13        Ensures(Acc(self.key) and self.key is key and
14               Acc(self.payload) and self.payload is val and
15               Acc(self.leftChild) and self.leftChild is left and
16               Acc(self.rightChild) and self.rightChild is right and
17               Acc(self.parent) and self.parent is parent)
18
19
20 @Pure
21 def hasLeftChild(self) -> Optional['TreeNode']:
22     Requires(Acc(self.leftChild))
23     return self.leftChild
24
25 @Pure
26 def hasRightChild(self) -> Optional['TreeNode']:
```

- Currently used to verify SCION internet architecture implementation
- Future work: Higher order functions, secure information flow

Try it out!

www.github.com/marcoeilers/nagini

vipser.ethz.ch/nagini-examples/