BacCaml: The Meta-Hybrid Just-In-Time Compiler

Yusuke Izawa (Tokyo Institute of Technology)

Context: Language Implementation Frameworks

providing an effective way to create a VM with Just-In-Time (JIT) compiler

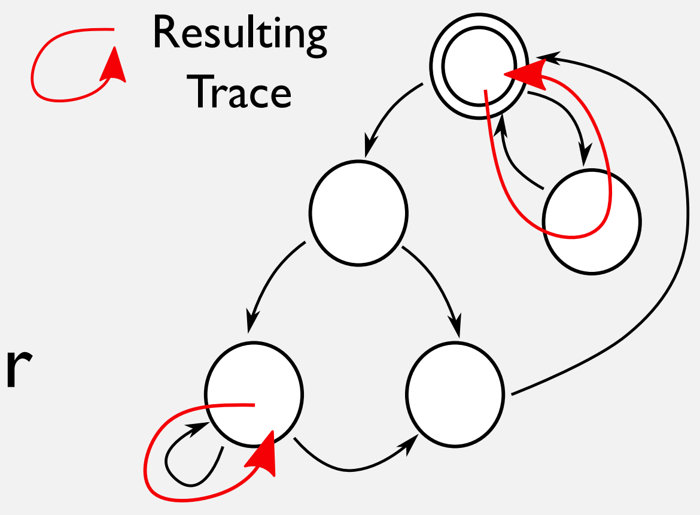
Cmpl. strategy	Focus Trace-based	Method-based
Framework	RPython[1]	Truffle/Graal[2]
Implementaion by framework	pypy	TRUFFLE RUBY
Cmpl. target	Frequently-executed code path (e.g.) loop)	Frequently-called functions

Problem: Path Divergence Problem

Mismatch between tracing and running

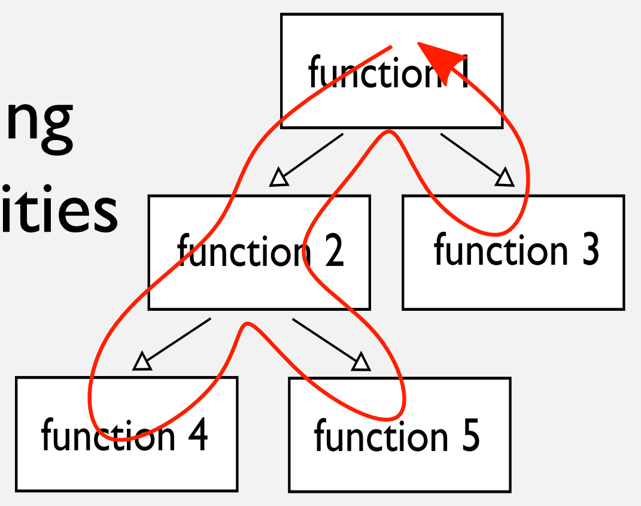
1. Getting a trace of a *varying control-flow*
2. Resulting in not covering all paths
3. Most executions are run in an interpreter

The control-flow of Fibonacci



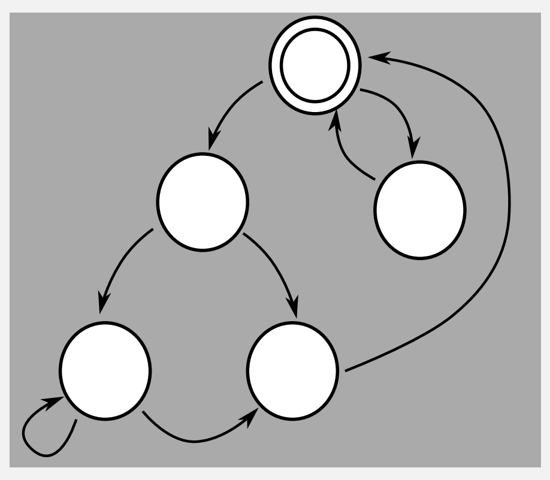
Solution: Apply different compilation strategies for different program parts

Branching Possibilities



Apply Trace-based Cmpl.

Varying Control-flow



Apply Method-based Cmpl.

Proposal: Meta-Hybrid Compilation Framework

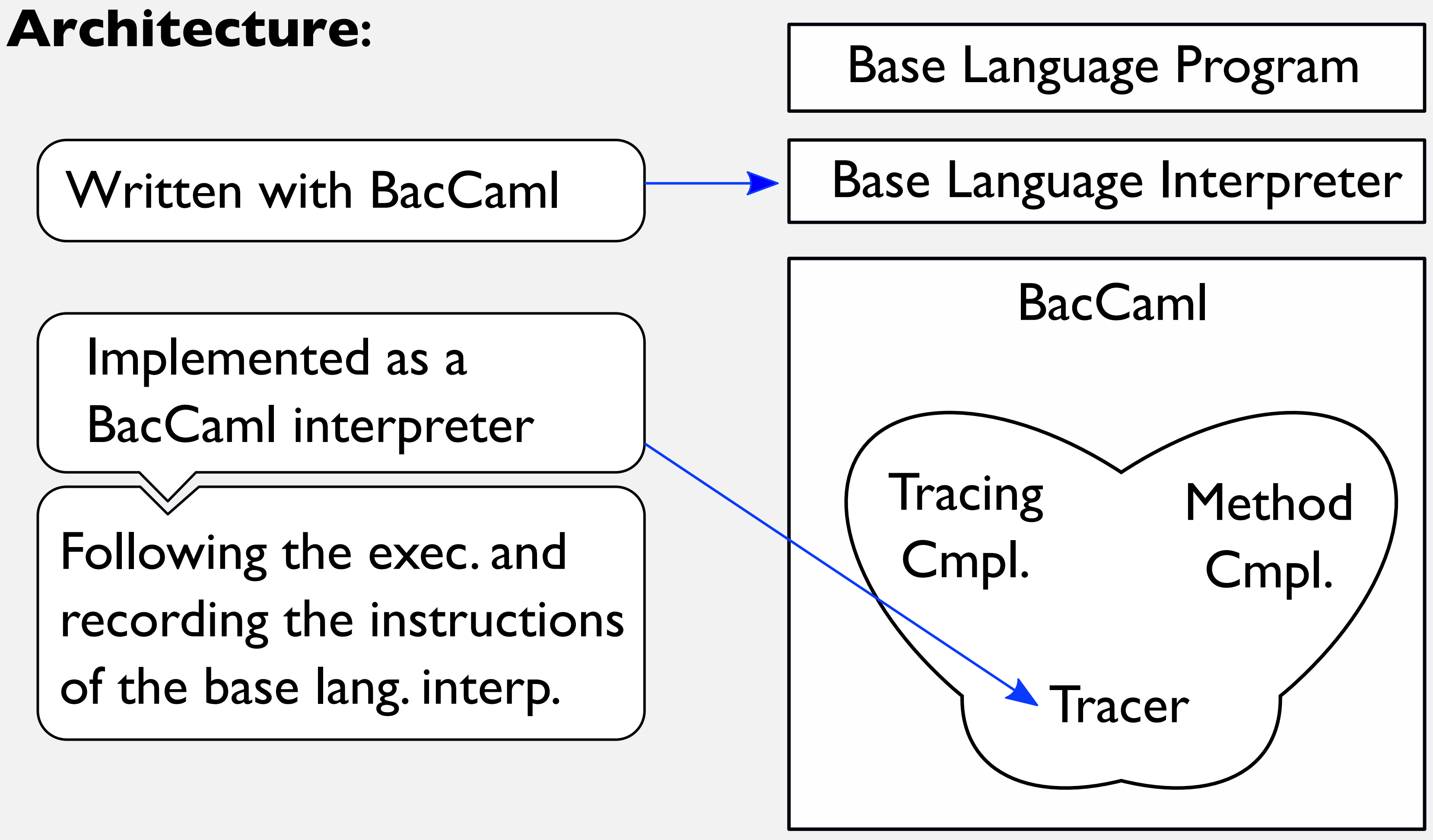
Principle:

- **Re-using** a meta-tracing compilation in a method compilation
- Requiring only a **single** interpreter definition

Proof-of-conpept Implementation (from scratch):

- **BacCaml**, based on MinCaml[3] compiler

Architecture:



Base Language Program

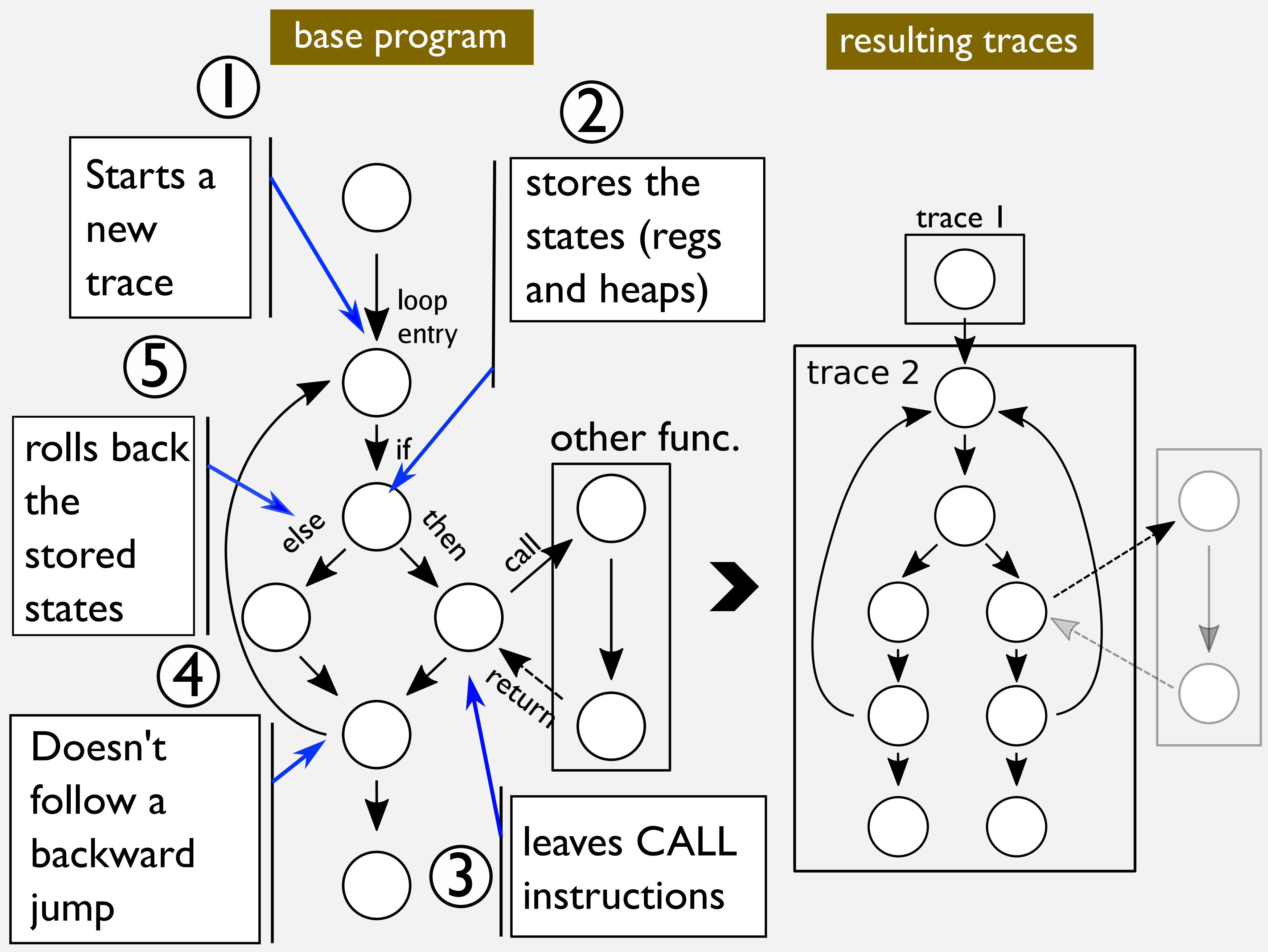
Written with BacCaml → Base Language Interpreter

Implemented as a BacCaml interpreter

Following the exec. and recording the instructions of the base lang. interp.

BacCaml: Tracing Cmpl. / Method Cmpl. / Tracer

How does the BacCaml interpreter trace a function? Tracing all possible paths in the function



① Starts a new trace

② stores the states (regs and heaps)

③ leaves CALL instructions

④ Doesn't follow a backward jump

⑤ rolls back the stored states

Results: Preliminary Benchmarking Test

What we have done:

- Run both compilation strategies separately
- Run the tracer by manyally specifying entry/exit points

What we have not done:

- Cooperating both compilation strategies
- Profiling and dynamic linking
- and more...

Took following benchmarks:

- fib: suitable for method-based compilation
- sum: suitable for trace-based compilation

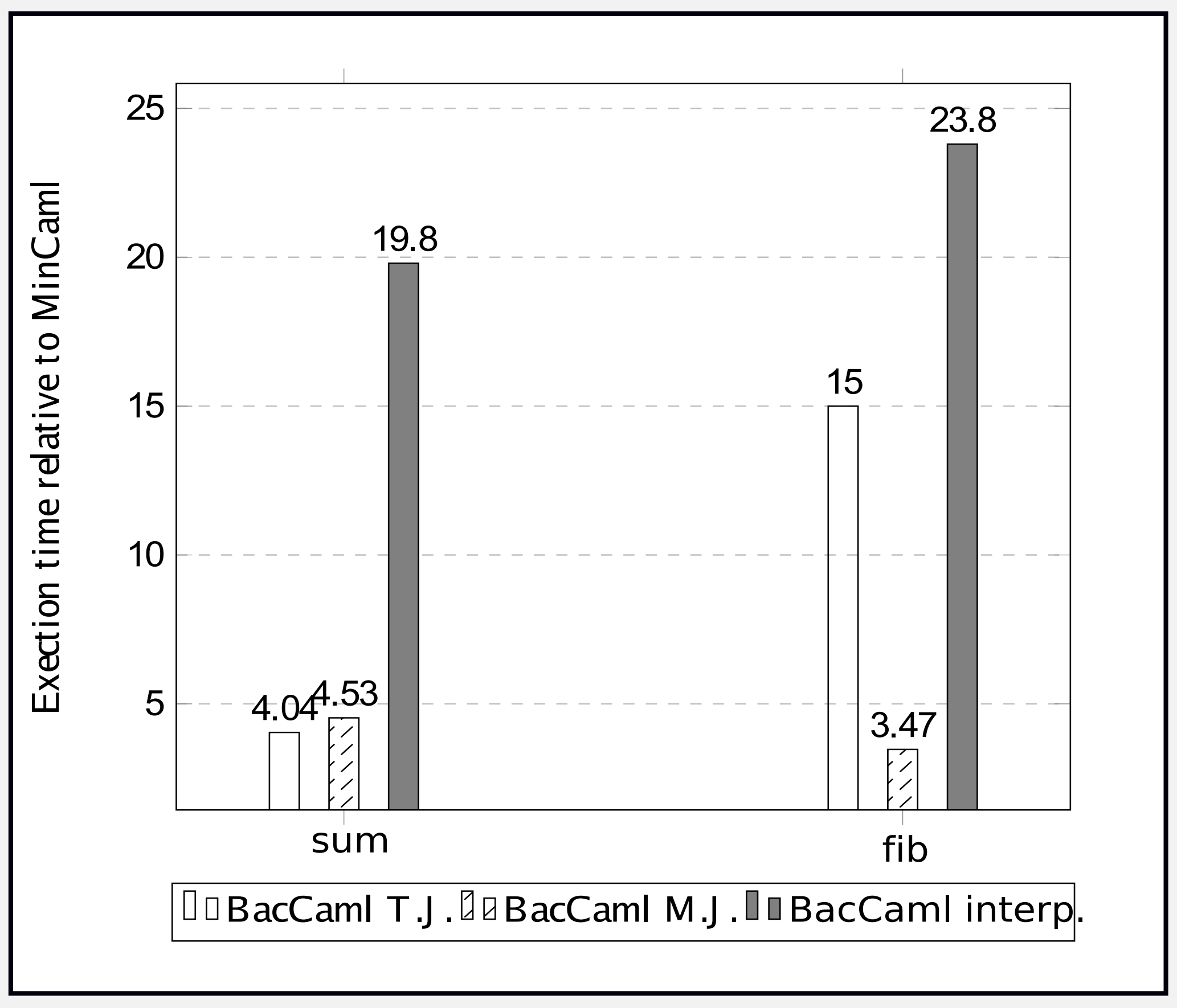
From results:

Our compiler works well

- In sum, the tracing compilation is 4x faster than the method

Our method compilation works well in fib causing Path Divervence Problem

Preliminary Benchimarking Results



Future work:

- Implement many runtime optimizations
- Investigate a strategy of switching compilations
- Apply this approach to RPython

References:

[1] Carl Friedrich Bolz, et al., "Tracing the meta-level: PyPy's tracing JIT compiler", Proceedings of the 4th workshop on the Implementation, Compilation, Optimization of Object-Oriented Languages and Programming Systems - ICOOLPS '09, pp. 18-25

[2] Thomas Wurthinger, et al., "Self-optimizing AST interpreters", In Proceedings of the 8th symposium on Dynamic languages (DLS '12), pp. 73-82

[3] Eijiro Sumii. 2005. MinCaml: a simple and efficient compiler for a minimal functional language. In Proceedings of the 2005 workshop on Functional and declarative programming in education (FDPE '05). ACM, New York, NY, USA, pp. 27-38.