# A Formal Foundation for Trace-Based JIT Compilation

Vandercammen, Maarten; Nicolay, Jens; De Roover, Coen

# A Formal Foundation for Trace-Based JIT Compilation

Maarten Vandercammen  Jens Nicolay  Coen De Roover

Software Languages Lab, Vrije Universiteit Brussel, Belgium

## Problem: design formal framework for trace-based JIT compilation

### Trace-based JIT Compilation

```
(define (fac n)
  (if (< n 2)
      1
      (* n
         (fac (- n 1)))))
```

Guo and Palsberg (2011)
Disegna et al. (2014)

Tracing tied to specific execution model

```
(label 'fac-loop)
(literal-value 2)
(save-val)
(lookup-var 'n)
(save-val)
(lookup-var '<)
(apply-native 2)
(guard-false)
(literal-value 1)
(save-val)
(lookup-var 'n)
(save-val)
(lookup-var '-)
(apply-native 2)
. . .
(goto 'fac-loop)
```

### Research goal

Language semantics (interpreter) → Traced language semantics (TJIT Compiler)

Recording/Executing traces

Handling guard failures

Dynamic analysis hook

From

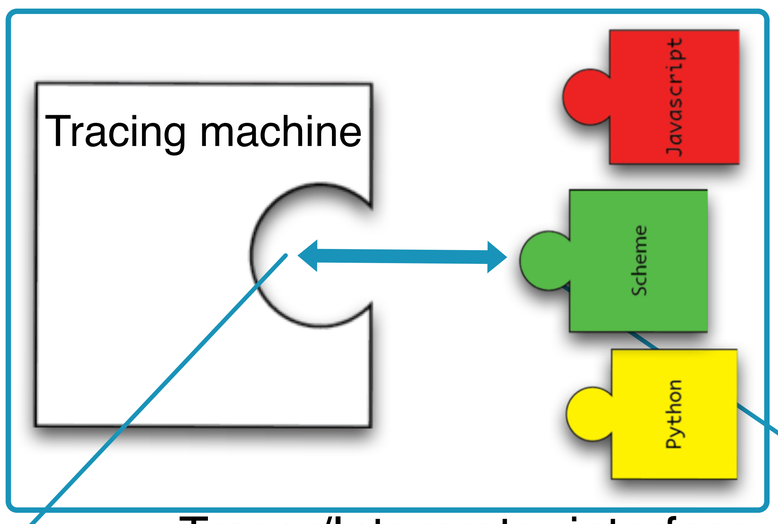
Local optimisations on traces

Common framework for trace and non-trace execution

Towards

Global optimisations between traces

## Approach: separate tracer and interpreter by creating an explicit interface

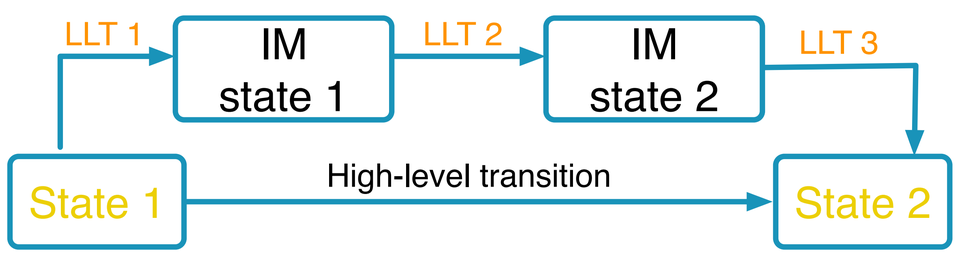



Tracer/Interpreter interface

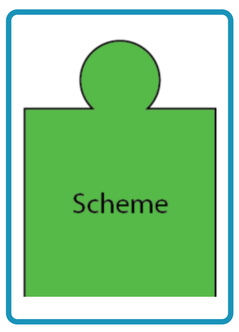
Interpreter as state machine

```
optimize: Trace -> Trace

restart: RestartPoint X ProgramState
         -> ProgramState

step: ProgramState -> InterpreterReturn
```

<ProgramState, LLT's, Loop signal>

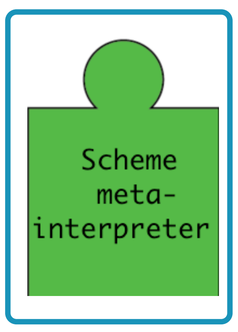


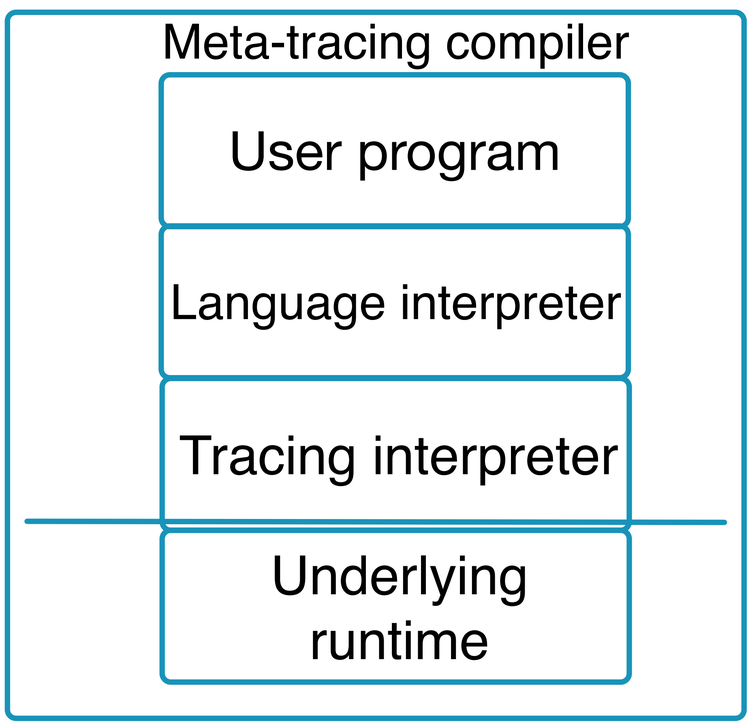

## Experiment: apply transformation on existing interpreters




```
(match program-state
  . . .
  ((ifk condition consequent alternative) ρ σ κ))
    (if condition
        (interpreter-return
                    program-state
                    (ev consequent)
                    #f
                    (restore-env)
                    (pop-continuation)
                    (guard-true alternative))
        (interpreter-return
                    program-state
                    (ev alternative)
                    #f
                    (restore-env)
                    (pop-continuation)
                    (guard-false consequent))))
  . . .)
```

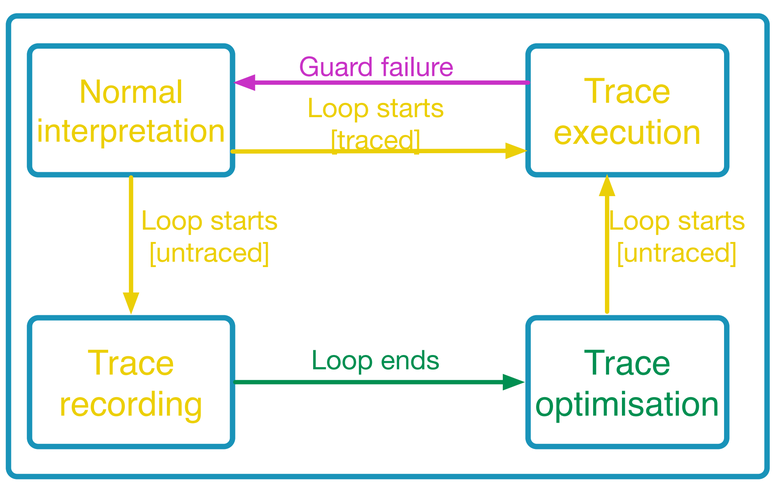
Minimal changes to implementation required




## Implementation: formalism + implementation

### State machine




### Formal semantics

$$\mathbf{ts}(TE,\ tc,\ \varsigma,\ \mathbf{tn}(lbl,\ \iota : \tau)) \rightarrow \mathbf{ts}(TE,\ tc,\ \varsigma',\ \mathbf{tn}(lbl,\ \tau)) \quad \text{if } \iota(\varsigma) = \mathbf{traceStep}(\varsigma')$$

$$\mathbf{ts}(TE,\ tc,\ \varsigma,\ \mathbf{tn}(lbl,\ \iota : \tau)) \rightarrow \mathbf{ts}(NI,\ tc,\ \varsigma',\ \mathit{False}) \quad \text{if } \iota(\varsigma) = \mathbf{guardFailed}(rp) \text{ and where } \varsigma' = \mathbf{restart}(rp,\ \varsigma)$$

$$\mathbf{ts}(TE,\ tc,\ \varsigma,\ \mathbf{tn}(lbl,\ `())) \rightarrow \mathbf{ts}(TE,\ tc,\ \varsigma,\ \mathbf{tn}(lbl,\ \tau)) \quad \text{and where } \tau \text{ is the trace that has already been recorded for } lbl$$

### Implementation

```
(define (do-trace-executing-step)
  (let* ((trace-node (evaluator-state-trace-executing
                      evaluator-state))
         (trace (trace-node-trace trace-node))
         (label (trace-key-label (trace-node-trace-key
                                  trace-node))))
    (if (null? trace)
        (let* ((old-trace-node (evaluator-state-trace-executing
                                evaluator-state))
               (old-trace-key (trace-node-trace-key old-trace-node))
               (trace-key (make-label-trace-key
                           (trace-key-label old-trace-key)
                           (trace-key-debug-info old-trace-key)))
               (new-trace-node (get-trace tracer-context trace-key)))
          (evaluator-state-copy evaluator-state
                                (trace-executing new-trace-node)))
        (let* ((instruction (car trace))
               (program-state
                (evaluator-state-program-state evaluator-state)))
          (handle-response-executing (instruction program-state))))))
```

Maarten.Vandercammen@vub.ac.be
@mvdcamme

https://github.com/mvdcamme/woda15/

A Formal Foundation for Trace-Based JIT Compilers
Maarten Vandercammen, Jens Nicolay, Stefan Marr, Joeri De Koster, Theo D'Hondt, Coen De Roover
WODA2015

Vrije Universiteit Brussel