Correctness of Speculative Optimizations with Dynamic Deoptimization

Olivier Flückiger, Gabriel Scherer, Ming-Ho Yee, Aviral Goel, Amal Ahmed, Jan Vitek

November 27, 2017

Northeastern University, Boston, USA
Context
Just-in-time compilation: Deoptimization

Source program, e.g. AST or bytecode
Just-in-time compilation: Deoptimization

Critical section, hot loop
Just-in-time compilation: Deoptimization

Critical section, hot loop.

Optimized version, e.g. native
Just-in-time compilation: Deoptimization

Speculation
Just-in-time compilation: Deoptimization
Outline

Case study: V8 and speculation

Sourir: modeling deoptimization

Optimizations in sourir

Formalization
yield
// Considers only the first element

function eq(x) {
    return x[0] === 42
}


// Considers only the first element
function eq(x) {
    return x[0] === 42
}

// Array of length 3
var x = [42, 1, .2]
// Considers only the first element
function eq(x) {
    return x[0] === 42
}

// Array of length 3
var x = [42, 1, .2]

// Sparse array of length 3 with element 1 undefined
var x = [42]; x[2] = .2
yield
Compiler Correctness?

Multiple versions need to be considered.

Speculation requires keeping deoptimization metadata.

Difficulty: intra-version optimizations in the presence of inter-version controlflow.

Research Question: Interaction between deoptimization points and compiler optimizations.
yield
Sourir
What does a JIT entail?

- High- and low-level representations
- Dynamic code generation
- Deoptimization metadata and supporting optimizations
Nothing left to remove

What does a JIT entail?

- High- and low-level representations
- Dynamic code generation
- Deoptimization metadata and supporting optimizations

One single language
What does a JIT entail?

- High- and low-level representations
- Dynamic code generation
- Deoptimization metadata and supporting optimizations
What does a JIT entail?

- High-and-low-level representations One single language
- Dynamic code generation One unrolled multi-version program
- Deoptimization metadata and supporting optimizations ✓
fun(c)

V luck

\textbf{assume} \ c = 41 \ \textbf{else} \ \text{fun.Vtough.L1} \ [c = c, o = 1]

\textbf{print} 42

V tough

\textbf{var} \ o = 1

L1 \ \textbf{print} c + o
fun(c)

$V_{\text{luck}}$

L0  \begin{cases} 
  \text{assume } c = 41 & \text{else } \text{fun.}V_{\text{tough}}.L1 \ [c = c, o = 1] \\
  \text{print } 42
\end{cases}

V_{\text{tough}} \ldots

assume e^* else fun.\text{Vver}.L \ [x_1 = e_1, \ldots, x_n = e_n]

**Predicates**: list of boolean conditions $e^*$

**Metadata**:

- **where** fun.\text{Vver}.L (unique location)
- **how** $[x_1 = e_1, \ldots, x_n = e_n]$ (frame at bailout target)
1)

```
var o = 1
assume c = 41 else F.V.L [c = c, o = o]
print c + o
```
Optimization: Constant Propagation

1)

var o = 1
assume c = 41 else F.V.L [c = c, o = o]
print c + o

2)

assume c = 41 else F.V.L [c = c, o = 1]
print c + 1
Optimization: Constant Propagation

1) 

```plaintext
var o = 1
assume c = 41 else F.V.L [c = c, o = o]
print c + o
```

2) 

```plaintext
assume c = 41 else F.V.L [c = c, o = 1]
print c + 1
```

3) 

```plaintext
assume c = 41 else F.V.L [c = c, o = 1]
print 42
```
yield
Baseline Version
Establish Invariants

Copy Version: Assumes are trivial
Speculation Pipeline

Preserve Invariants

Optimizations

![Diagram showing preservation of invariants through optimizations. 1:1 ratio between stages.](image)
Finally

Most Optimized & Baseline Version
Finally

Equivalence result: Most Optimized & Baseline Version
Results

Explicit instruction for deoptimization

Invariants between versions

Optimizations are easy to adapt
Formalization
Execution: Operational semantics

Configurations:

\[ C ::= \langle P \mid L \mid K^* \mid M \mid E \rangle \]

Actions:

\[ A ::= \text{read lit} \mid \text{print lit} \]

\[ A_\tau ::= A \mid \tau \]

\[ T ::= A^* \]

Reduction:

\[ C_1 \xrightarrow{A_\tau} C_2 \]

\[ C_1 \xrightarrow{T^*} C_2 \]
 Execution: A Peek

\[ [\text{BranchT}] \]

\[
I(L) = \text{branch } e \ L_1 \ L_2 \quad M \ E \ e \rightarrow \text{true} \\
\langle P \ I \ L \ K^* \ M \ E \rangle \xrightarrow{\tau} \langle P \ I \ L_1 \ K^* \ M \ E \rangle
\]
**Execution: A Peek**

**[BranchT]**

\[ I(L) = \text{branch } e \, L_1 \, L_2 \quad M \, E \, e \rightarrow \text{true} \]

\[ \langle P \, I \, L \, K^* \, M \, E \rangle \xrightarrow{\tau} \langle P \, I \, L_1 \, K^* \, M \, E \rangle \]

**[Print]**

\[ I(L) = \text{print } e \quad M \, E \, e \rightarrow \text{lit} \]

\[ \langle P \, I \, L \, K^* \, M \, E \rangle \xrightarrow{\text{print lit}} \langle P \, I \, (L+1) \, K^* \, M \, E \rangle \]
Equivalence: (weak) bisimulation

Relation $R$ between the configurations over $P_1$ and $P_2$.

$R$ is a weak simulation if:

$\begin{align*}
  C_1 \xrightarrow{A_\tau} C'_1 \\
  C_2 \xrightarrow{R} C_2
\end{align*}$

$\begin{align*}
  C_1 \xrightarrow{A_\tau} C'_1 \\
  C_2 \xrightarrow{A_\tau} C'_2
\end{align*}$

$R$ is a weak bisimulation if $R$ and $R^{-1}$ are simulations.
Version equivalence All versions of a function are equivalent.
(Necessary to replace the active version)

Assumption transparence Bailing out more than necessary is correct.
(Necessary to add new assumptions)
yield
Optimization Pipeline: Create a new Version

... \[ A_T \]

\[ \rightarrow \text{print } \]
Optimization Pipeline: Create a new Version

... $A_{\tau}$ \rightarrow \text{\texttt{print}} x

... $A_{\tau}$ \rightarrow \text{\texttt{print}} x
... \xrightarrow{A_\tau} \text{assume true else } F.V.L_1 \ldots \xrightarrow{\tau} \text{print } x

... \xrightarrow{A_\tau} \text{print } x
Optimization Pipeline: Create a new Version

\[ \ldots \xrightarrow{A_\tau} \text{assume true else } F.V.L_1 \ldots \xrightarrow{\tau} \text{print x} \]

\[ \ldots \xrightarrow{A_\tau} \]
All you need for speculation: versions + checkpoints
Correctness of Speculative Optimizations with Dynamic Deoptimization (POPL’ 18)
https://arxiv.org/abs/1711.03050
https://www.o1o.ch/talk-sourir-rmod.pdf
Advanced Topics
Adding more assumptions

Is deoptimizing in P2 correct, even if P1 does not deoptimize?
Yes, because of assumption transparency in P1.
Adding more assumptions

Is deoptimizing in P2 correct, even if P1 does not deoptimize?
Is deoptimizing in P2 correct, even if P1 does not deoptimize?
Yes, because of assumption transparency in P1.
How many deoptimization points are necessary?

Deoptimization points are expensive. How many are necessary?

Should assume be split into framestate and guard instructions? (unrestricted deoptimization)
Unrestricted deoptimization is just a transformation

before:

```plaintext
assume true else size.Vb.L0 [x = x]
branch x = nil L2 L1
L1 x ← x[0]
return x * el
L2 ...
```

after:

```plaintext
var x0 = x
branch x = nil L2 L1
L4 x ← x[0]
assume x = 1 else size.Vb.L0 [x = x0]
return 1 * el
L3 ...
```
We can inline with deoptimization points

```plaintext
main( )

Vinlined
  array vec = [1, 2, 3, 4]
  var size = nil
  var obj = vec
  assume obj ≠ nil else
    size.Vbase.L1 [...]
    main.Vbase.Lret size [...]

  var len = length(obj)
  size ← len * 4
  drop len
  drop obj
  goto Lret
Lret print size

Vbase ...
```

```plaintext
main( )

Vbase
  array vec = [1, 2, 3, 4]
  call size = size(vec)
Lret
  print size

size(obj)
  Vopt
    assume obj ≠ nil else ...
    var len = length(obj)
    return len * 4
Vbase ...
```

Need for an extra frame in the inlined version
Future Work

experimental validation

bidirectional transformations