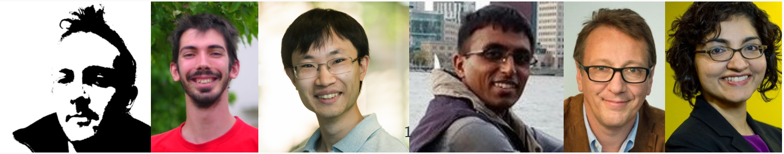
Speculative Optimizations without Fear

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

August 9, 2017

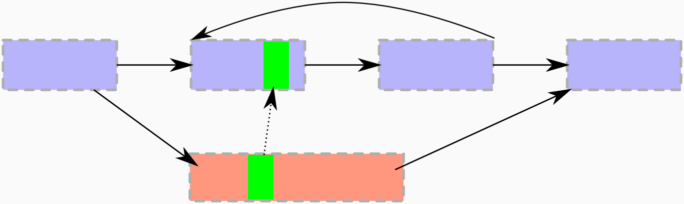
Northeastern University, Boston, USA



Context

Just-in-time compilation: Deoptimization

Baseline version, e.g. AST, bytecode.



Optimized version, e.g. bytecode, native.

Checkpoint : Deoptimization/OSR point

Compiler Correctness?

Ahead-of-time: Relation between input and output program

Here: Relation between differently optimized **versions**

Difficulty: Intra-version control flow

Checklist

Speculations: using assumptions for optimizations

- Model speculative optimization and deoptimization
- Capture interaction between optimization and checkpoints
- Let practitioners reason about correctness

Sourir

Modeling Speculation

- high- and low-level representations
- dynamic code generation
- speculative optimization and bailout

Modeling Speculation

- ~~high- and low-level representations~~ a single bytecode language
- dynamic code generation
- speculative optimization and bailout

Modeling Speculation

- ~~high- and low-level representations~~ a single bytecode language
- ~~dynamic code generation~~ one unrolled multi-version program
- speculative optimization and bailout

Modeling Speculation

- ~~high- and low-level representations~~ a single bytecode language
- ~~dynamic code generation~~ one unrolled multi-version program
- speculative optimization and bailout ✓

Modeling Speculation

- ~~high- and low-level representations~~ a single bytecode language
- ~~dynamic code generation~~ one unrolled multi-version program
- speculative optimization and bailout ✓

(See Myreen [2010] for the first two)

Sourir: instructions

$$
\begin{array}{rcl}
i & ::= & \\
 & | & \textbf{var}\ x = e \\
 & | & \textbf{drop}\ x \\
 & | & x \leftarrow e \\
 & | & \textbf{array}\ x[e] \\
 & | & \textbf{array}\ x = [e^*] \\
 & | & x[e_1] \leftarrow e_2 \\
 & | & \textbf{branch}\ e\ \ L_1\ \ L_2 \\
 & | & \textbf{goto}\ L \\
 & | & \textbf{print}\ e \\
 & | & \textbf{read}\ x \\
 & | & \textbf{call}\ x = e(e^*) \\
 & | & \textbf{return}\ e \\
 & | & \textbf{assume}\ [e^*]\ \textbf{else}\ \xi\ \tilde{\xi}^*
\end{array}
$$

HelloWorld

$$
\begin{aligned}
& F_{fun}(c) \rightarrow \\
& \quad V_{tough} \rightarrow \\
& \qquad L_0 \quad : \quad \textbf{var } o = 1 \\
& \qquad L_1 \quad : \quad \textbf{print}\,(c + o)
\end{aligned}
$$

HelloWorld

$$
\begin{array}{l}
F_{fun}(c) \rightarrow \\
\quad V_{luck} \rightarrow \\
\qquad L_0 \quad : \quad \textbf{assume } [(c = 41)] \textbf{ else } \langle F_{fun}.V_{tough}.L_1 \ [c = c, o = 1] \rangle \\
\qquad L_1 \quad : \quad \textbf{print } 42 \\
\quad V_{tough} \rightarrow \\
\qquad L_0 \quad : \quad \textbf{var } o = 1 \\
\qquad L_1 \quad : \quad \textbf{print } (c + o)
\end{array}
$$

Checkpoints

$$
\begin{aligned}
&F_{fun}(c) \rightarrow \\
&\quad V_{luck} \rightarrow \\
&\qquad L_0 \;:\; \textbf{assume } [(c\ =\ 41)]\ \textbf{else}\ \langle F_{fun}.V_{tough}.L_1\ [c = c, o = 1]\rangle \\
&\qquad L_1 \;:\; \textbf{print}\ 42 \\
&\quad V_{tough} \rightarrow \ \ldots
\end{aligned}
$$

$$\textbf{assume}\ [e^*]\ \textbf{else}\ \langle F_{aFun}.V_{aVers}.L_{aLabel}\ [x_1 = e_1, \ldots, x_n = e_n]\rangle$$

Guards: list of boolean conditions e^*

Bailout data:

where $F_{aFunction}.V_{aVersion}.L_{aLabel}$ (unique location)

how $[x_1 = e_1, \ldots, x_n = e_n]$ (frame at bailout target)

Versions

Multiple versions per function and multiple assumptions per version

Assumptions undoable with version granularity

Only assume explicitly refers to versions

Transformations applied to one version do not consider others

Results

Optimizations: Constant Propagation

$$
\begin{aligned}
L_1 &: \textbf{var } x = 1 \\
L_3 &: \textbf{var } z = (x + y) \\
L_2 &: \textbf{assume } [\,] \textbf{ else } \langle F.V.L\ [x = x, y = y, z = z] \rangle \\
L_4 &: \textbf{print } z
\end{aligned}
$$

Optimizations: Constant Propagation

$$
\begin{aligned}
L_3 &: \ \textbf{var}\ z = (1 + y) \\
L_2 &: \ \textbf{assume}\ [\,]\ \textbf{else}\ \langle F.V.L\ [x = 1, y = y, z = z]\rangle \\
L_4 &: \ \textbf{print}\ z
\end{aligned}
$$

Optimizations: Constant Propagation

$$\begin{array}{lcl} L_2 & : & \textbf{assume}\ [\,]\ \textbf{else}\ \langle F.V.L\ [x = 1, y = y, z = (1\ +\ y)]\rangle \\ L_4 & : & \textbf{print}\,(1\ +\ y) \end{array}$$

Optimizations: Constant Propagation

$$
\begin{aligned}
L_2 &\quad : \quad \textbf{assume } [(y\ =\ 2)] \textbf{ else } \langle F.V.L\ [x = 1, y = y, z = (1\ +\ y)]\rangle \\
L_4 &\quad : \quad \textbf{print } 3
\end{aligned}
$$

Inlining

$$
\begin{array}{l}
F_{main}(\,) \rightarrow \\
\quad V_{inlined} \rightarrow \\
\qquad \begin{array}{lcl}
L_0 & : & \textbf{array}\ vec = [1,2,3,4] \\
L_2 & : & \textbf{var}\ size = \textbf{nil} \\
L_3 & : & \textbf{var}\ obj = vec \\
L_{cp_1} & : & \textbf{assume}\ [(obj \neq \textbf{nil})]\ \textbf{else}\ \ldots \\
L_5 & : & \textbf{var}\ len = \textbf{length}(obj) \\
L_6 & : & size \leftarrow (len * 4) \\
L_7 & : & \textbf{drop}\ len \\
L_8 & : & \textbf{drop}\ obj \\
L_9 & : & \textbf{goto}\ L_{ret} \\
L_{ret} & : & \textbf{print}\ size
\end{array} \\
\quad V_{base} \rightarrow \ \ldots
\end{array}
$$

$$
\begin{array}{l}
F_{main}(\,) \rightarrow \\
\quad V_{base} \rightarrow \\
\qquad \begin{array}{lcl}
L_0 & : & \textbf{array}\ vec = [1,2,3,4] \\
L_2 & : & \textbf{call}\ size = \text{`}F_{size}(vec) \\
L_{ret} & : & \textbf{print}\ size
\end{array} \\
,\ F_{size}(obj) \rightarrow \\
\quad V_{opt} \rightarrow \\
\qquad \begin{array}{lcl}
L_{cp_1} & : & \textbf{assume}\ [(obj \neq \textbf{nil})]\ \textbf{else}\ \ldots \\
L_{vec} & : & \textbf{var}\ len = \textbf{length}(obj) \\
L_3 & : & \textbf{return}\,(len * 4)
\end{array} \\
\quad V_{base} \rightarrow \ \ldots
\end{array}
$$

Need for an extra frame in the inlined version:

L_{cp1} : **assume** $[(obj \neq \textbf{nil})]$ **else** $\langle F_{size}.V_{base}.L_1\ [\ldots]\rangle\ \langle F_{main}.V_{base}.L_{ret}\ size\ [\ldots]\rangle$

Step-By-Step

$$
\begin{array}{lll}
V_{base} \rightarrow & & \\
\quad L_1 & : & \textbf{branch}\,(tag\ =\ INT)\ L_{int}\ L_{nonint} \\
\quad L_{int} & : & \textbf{return}\,(x + 1) \\
\quad L_{nonint} & : & \ldots
\end{array}
$$

Step-By-Step

$$
\begin{aligned}
&V_{base} \rightarrow \\
&\quad L_1 &:& \quad \textbf{branch}\,(tag\ =\ INT)\ L_{int}\ L_{nonint} \\
&\quad L_{int} &:& \quad \textbf{return}\,(x\ +\ 1) \\
&\quad L_{nonint} &:& \quad \ldots
\end{aligned}
$$

1. Create new Version

$$
\begin{aligned}
&V_{opt} \rightarrow \\
&\quad L_0 &:& \quad \textbf{assume}\ [\,]\ \textbf{else}\ \langle F.V_{base}.L_1\ [tag = tag, x = x]\rangle \\
&\quad L_1 &:& \quad \textbf{branch}\,(tag\ =\ INT)\ L_{int}\ L_{nonint} \\
&\quad L_{int} &:& \quad \textbf{return}\,(x\ +\ 1) \\
&\quad L_{nonint} &:& \quad \ldots
\end{aligned}
$$

Step-By-Step

$$
\begin{array}{lcl}
V_{base} \rightarrow & & \\
\quad L_1 & : & \textbf{branch}\,(tag\ =\ INT)\ L_{int}\ L_{nonint} \\
\quad L_{int} & : & \textbf{return}\,(x\ +\ 1) \\
\quad L_{nonint} & : & \ldots
\end{array}
$$

2. A new Assumption

$$
\begin{array}{lcl}
V_{opt} \rightarrow & & \\
\quad L_0 & : & \textbf{assume}\ [(tag\ =\ INT)]\ \textbf{else}\ \langle F.V_{base}.L_1\ [tag = tag, x = x]\rangle \\
\quad L_1 & : & \textbf{branch}\,(tag\ =\ INT)\ L_{int}\ L_{nonint} \\
\quad L_{int} & : & \textbf{return}\,(x\ +\ 1) \\
\quad L_{nonint} & : & \ldots
\end{array}
$$

Step-By-Step

$$
\begin{array}{lll}
V_{base} \rightarrow & & \\
\quad L_1 & : & \textbf{branch}\,(tag\ =\ INT)\ L_{int}\ L_{nonint} \\
\quad L_{int} & : & \textbf{return}\,(x\ +\ 1) \\
\quad L_{nonint} & : & \ldots
\end{array}
$$

3. Optimize: Remove Unreachable Code

$$
\begin{array}{lll}
V_{opt} \rightarrow & & \\
\quad L_0 & : & \textbf{assume}\ [(tag\ =\ INT)]\ \textbf{else}\ \langle F.V_{base}.L_1\ [tag = tag, x = x]\rangle \\
\quad L_{int} & : & \textbf{return}\,(x\ +\ 1)
\end{array}
$$

Hoisting Assumptions

$$
\begin{array}{lcl}
L_1 & : & \textbf{assume } [\,] \textbf{ else } \langle F.V.L\ [\,]\rangle \\
L_{loop} & : & \ldots \\
L_3 & : & \textbf{assume } [(y\ =\ 0)] \textbf{ else } \langle F.V.L\ [y = y, x = x]\rangle \\
L_4 & : & \textbf{branch } e\ L_{loop}\ L_5 \\
L_5 & : & \ldots
\end{array}
$$

Hoisting Assumptions

$$\begin{array}{lcl}
L_1 & : & \textbf{assume } [(y\ =\ 0)] \textbf{ else } \langle F.V.L\ [\,]\rangle \\
L_{loop} & : & \ldots \\
L_3 & : & \textbf{assume } [\,] \textbf{ else } \langle F.V.L\ [y = y, x = x]\rangle \\
L_4 & : & \textbf{branch}\, e\ L_{loop}\ L_5 \\
L_5 & : & \ldots
\end{array}$$

Hoisting Assumptions

$$
\begin{array}{lll}
L_1 & : & \textbf{assume } [(y = 0)] \textbf{ else } \langle F.V.L\ [\,]\rangle \\
L_{loop} & : & \ldots \\
L_4 & : & \textbf{branch } e\ L_{loop}\ L_5 \\
L_5 & : & \ldots
\end{array}
$$

Composing Checkpoints

$$
\begin{array}{l}
F_{undo}(\,) \rightarrow \\
\quad V_{spec123} \rightarrow \\
\qquad L_0 \;:\; \textbf{assume } [e_1, e_2, e_3] \textbf{ else } \langle F_{undo}.V_{spec12}.L_0 \; \delta_3 \rangle \\
\quad V_{spec12} \rightarrow \\
\qquad L_0 \;:\; \textbf{assume } [e_1, e_2] \textbf{ else } \langle F_{undo}.V_{spec1}.L_0 \; \delta_2 \rangle \\
\quad V_{spec1} \rightarrow \\
\qquad L_0 \;:\; \textbf{assume } [e_1] \textbf{ else } \langle F_{undo}.V_{base}.L_0 \; \delta_1 \rangle \\
\quad V_{base} \rightarrow \; \ldots
\end{array}
$$

Checkpoints compose

Composing Checkpoints

$$
\begin{array}{l}
F_{undo}(\,) \rightarrow \\
\quad V_{spec123} \rightarrow \\
\quad\quad L_0 \quad : \quad \textbf{assume } [e_1, e_2, e_3] \textbf{ else } \langle F_{undo}.V_{spec1}.L_0\ \delta_{23} \rangle \\
\quad V_{spec1} \rightarrow \\
\quad\quad L_0 \quad : \quad \textbf{assume } [e_1] \textbf{ else } \langle F_{undo}.V_{base}.L_0\ \delta_1 \rangle \\
\quad V_{base} \rightarrow \quad \ldots
\end{array}
$$

Intermediate versions can be removed after the fact

Formalization

Bailout Invariants

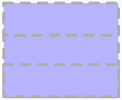
All **versions** of a function are observationally **equivalent**

Bailing out **more than necessary** is correct

The first invariant is our correctness result, the second allows adding more assumptions.

Proof: Speculation Pipeline

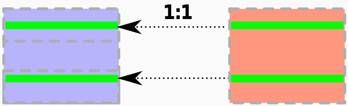
Baseline Version



Proof: Speculation Pipeline

Establish Invariants

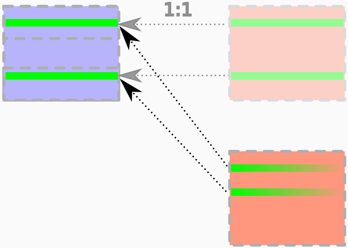
Copy Version: Checkpoints are trivial



Proof: Speculation Pipeline

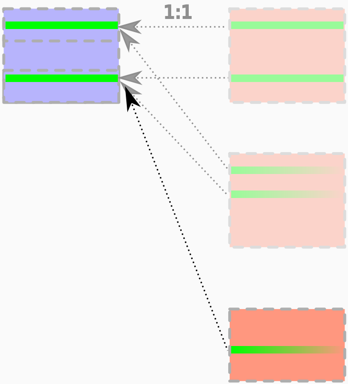
Preserve Invariants

Optimizations



Finally

Most Optimized & Baseline Version



Execution: Operational semantics

$$C ::= \langle P\ I\ L\ K^*\ M\ E \rangle$$

configuration

P		running program
I		current instruction stream
L		next instruction label
K^*	$::= (K_1, .., K_n)$	call stack
M		array **memory**
E		lexical **environment**

Actions:

$$A ::= \textbf{read}\ \mathit{lit} \mid \textbf{print}\ \mathit{lit} \qquad\qquad A_\tau ::= A \mid \tau$$

Reduction:

$$C_1 \xrightarrow{A_\tau}{}^{*}\ C_2$$

Execution: a Peek

$$\frac{I(L) = \ \textbf{branch}\, e\ L_1\ L_2 \quad M\ E\ e \rightarrow \textbf{true}}{\langle P\, I\, L\, K^*\ M\, E\rangle \xrightarrow{\tau} \langle P\, I\, L_1\, K^*\ M\, E\rangle}\ [\textsc{BranchT}]$$

Execution: a Peek

$$\frac{I(L) = \ \textbf{branch}\ e\ L_1\ L_2 \quad M\ E\ e \rightarrow \textbf{true}}{\langle P\, I\, L\, K^*\ M\, E\rangle \xrightarrow{\tau} \langle P\, I\, L_1\, K^*\ M\, E\rangle}\ \text{[BranchT]}$$

$$\frac{I(L) = \ \textbf{print}\ e \quad M\ E\ e \rightarrow \mathit{lit}}{\langle P\, I\, L\, K^*\ M\, E\rangle \xrightarrow{\textbf{print}\ \mathit{lit}} \langle P\, I\, (L{+}1)\, K^*\ M\, E\rangle}\ \text{[Print]}$$

Execution: a Peek

[BranchT]

$$\frac{I(L) = \textbf{branch}\ e\ L_1\ L_2 \quad M\ E\ e \to \textbf{true}}{\langle P\ I\ L\ K^*\ M\ E \rangle \xrightarrow{\tau} \langle P\ I\ L_1\ K^*\ M\ E \rangle}$$

[Print]

$$\frac{I(L) = \textbf{print}\ e \quad M\ E\ e \to lit}{\langle P\ I\ L\ K^*\ M\ E \rangle \xrightarrow{\textbf{print}\ lit} \langle P\ I\ (L{+}1)\ K^*\ M\ E \rangle}$$

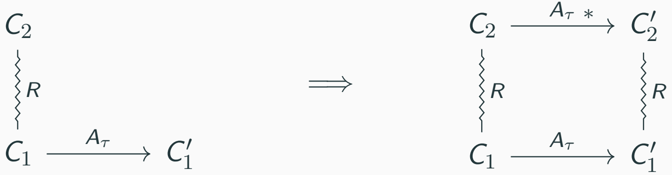
[Update]

$$\frac{I(L) = x \leftarrow e \quad x \in dom(E) \quad M\ E\ e \to v}{\langle P\ I\ L\ K^*\ M\ E \rangle \xrightarrow{\tau} \langle P\ I\ (L{+}1)\ K^*\ M\ E[x \leftarrow v] \rangle}$$

Equivalence: (weak) bisimulation

Relation R between the configurations over P_1 and P_2.

R is a weak **simulation** if:



R is a weak **bisimulation** if R and R^{-1} are simulations.

Optimization Pipeline: Create a new Version

explained in terms of the simulation relation

$$\ldots \xrightarrow{\qquad\qquad\qquad A_\tau \qquad\qquad\qquad} L_1 : \textbf{print}\, x$$

Optimization Pipeline: Create a new Version

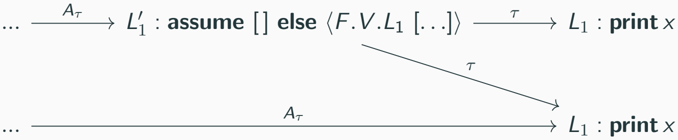
explained in terms of the simulation relation

$$\ldots \xrightarrow{\quad A_\tau \quad} L_1 : \textbf{print}\, x$$

$$\ldots \xrightarrow{\quad A_\tau \quad} L_1 : \textbf{print}\, x$$

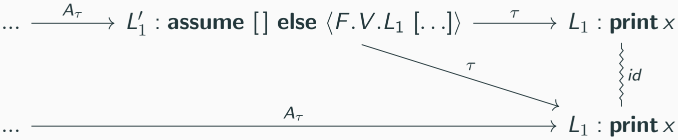
Optimization Pipeline: Create a new Version

explained in terms of the simulation relation



Optimization Pipeline: Create a new Version

explained in terms of the simulation relation



Optimization Pipeline: Constant Propagation

$$\begin{array}{lcl} L_0 & : & \textbf{var}\ x = 1 \\ L_1' & : & \textbf{assume}\ [\,]\ \textbf{else}\ \langle F.V.L_1\ [x = x] \rangle \\ L_1 & : & \textbf{print}\ x \end{array}$$

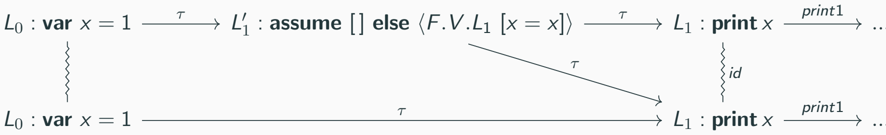
Optimization Pipeline: Constant Propagation

$$\begin{array}{lcl} L_0 & : & \textbf{var}\ x = 1 \\ L_1' & : & \textbf{assume}\ [\,]\ \textbf{else}\ \langle F.V.L_1\ [x = x] \rangle \\ L_1 & : & \textbf{print}\ x \end{array}$$

$$L_0 : \textbf{var}\ x = 1 \xrightarrow{\quad\quad\quad \tau \quad\quad\quad} L_1 : \textbf{print}\ x \xrightarrow{print 1} \ldots$$

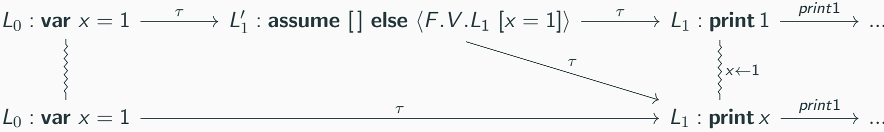
Optimization Pipeline: Constant Propagation

L_0 : **var** $x = 1$
L_1' : **assume** [ ] **else** $\langle F.V.L_1\ [x = 1] \rangle$
L_1 : **print** 1



Optimization Pipeline: Constant Propagation

$$\begin{array}{lcl} L_0 & : & \textbf{var } x = 1 \\ L_1' & : & \textbf{assume } [\,] \textbf{ else } \langle F.V.L_1\ [x = 1] \rangle \\ L_1 & : & \textbf{print } 1 \end{array}$$



Conclusion

All you need for speculation: versions + checkpoints

Future work:

experimental validation

bidirectional transformations

https://www.o1o.ch/sourir.pdf
https://www.o1o.ch/sourir-talk.pdf

References

Magnus O. Myreen. Verified just-in-time compiler on x86. In **Proceedings of the 37th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages**, POPL '10, pages 107–118, New York, NY, USA, 2010. ACM. ISBN 978-1-60558-479-9. doi: 10.1145/1706299.1706313.