

# Trace-based Just-in-Time Type Specialization for Dynamic Languages

Andreas Gal<sup>\*+</sup>, Brendan Eich\*, Mike Shaver\*, David Anderson\*, David Mandelin\*,  
Mohammad R. Haghagh†, Blake Kaplan\*, Graydon Hoare\*, Boris Zbarsky\*, Jason Orendorff\*,  
Jesse Ruderman\*, Edwin Smith#, Rick Reitmaier#, Michael Bebenita<sup>†</sup>, Mason Chang<sup>†#</sup>, Michael Franz<sup>†</sup>

Mozilla Corporation<sup>\*</sup>  
{gal,brendan,shaver,danderson,dmandelin,mrbkap,graydon,bz,jorendorff,jruderman}@mozilla.com

Adobe Corporation<sup>#</sup>  
{edwsmith,rreitmai}@adobe.com

Intel Corporation<sup>\$</sup>  
{mohammad.r.haghagh†}@intel.com

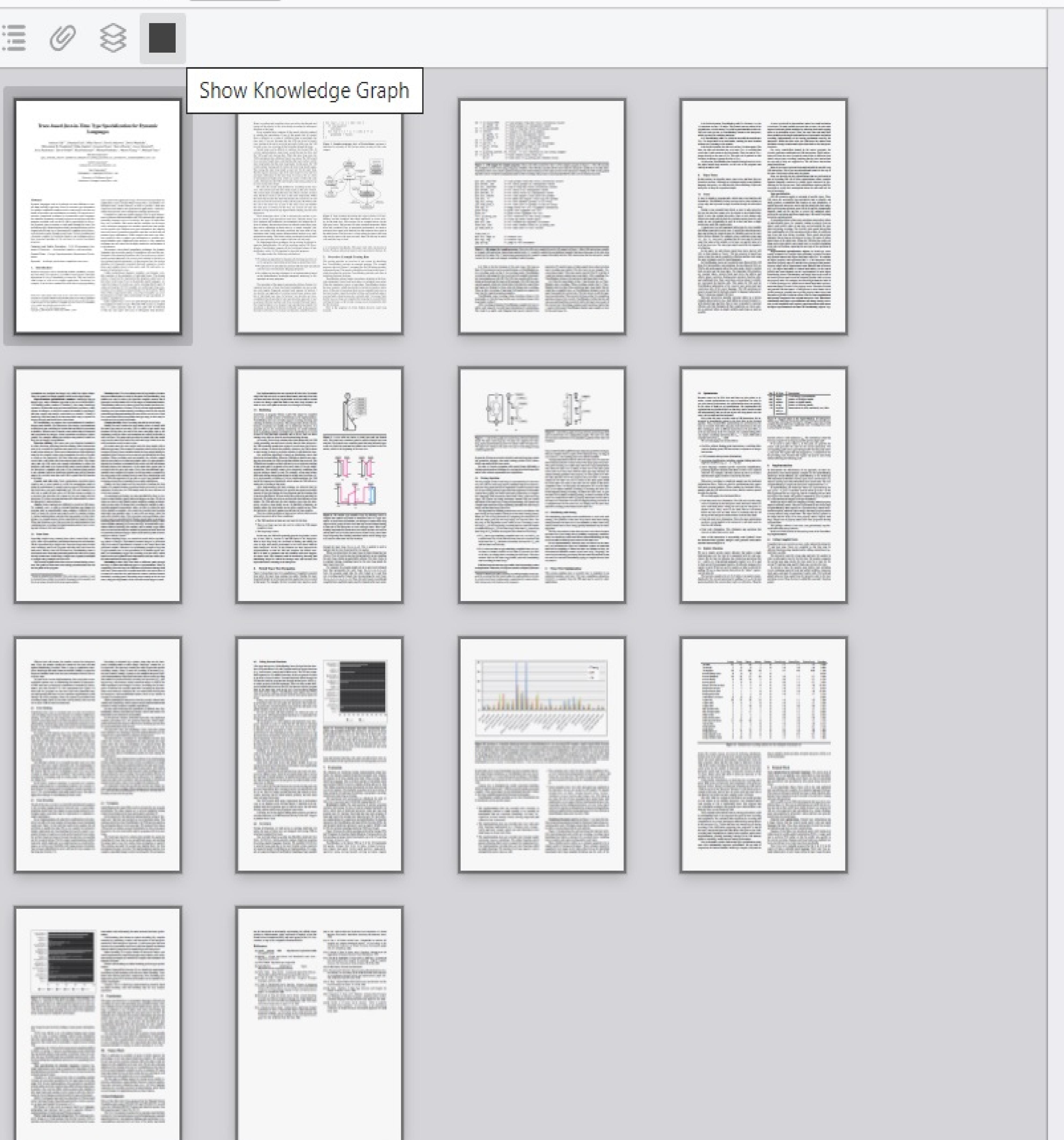
University of California, Irvine<sup>†</sup>  
{mbebenit,changm,franz}@uci.edu

## Abstract

Dynamic languages such as JavaScript are more difficult to compile than statically typed ones. Since no concrete type information is available, traditional compilers need to emit generic code that can handle all possible type combinations at runtime. We present an alternative compilation technique for dynamically-typed languages that identifies frequently executed loop traces at run-time and then

and is used for the application logic of browser-based productivity applications such as Google Mail, Google Docs and Zimbra Collaboration Suite. In this domain, in order to provide a fluid user experience and enable a new generation of applications, virtual machines must provide a low startup time and high performance.

Compilers for statically typed languages rely on type information to generate efficient machine code. In a dynamically typed programming language such as JavaScript, the types of expressions



Show Knowledge Graph

# Trace-based Just-in-Time Type Specialization for Dynamic Languages

Andreas Gal<sup>\*,†</sup>, Brendan Eich<sup>\*</sup>, Mike Shaver<sup>\*</sup>, David Anderson<sup>\*</sup>, David Mandelin<sup>\*</sup>, Mohammad R. Haghhighat<sup>§</sup>, Blake Kaplan<sup>\*</sup>, Graydon Hoare<sup>\*</sup>, Boris Zbarsky<sup>\*</sup>, Jason Orendorff<sup>\*</sup>, Jesse Ruderman<sup>\*</sup>, Edwin Smith<sup>#</sup>, Rick Reitmaier<sup>#</sup>, Michael Bebenita<sup>†</sup>, Mason Chang<sup>†,‡</sup>, and Franziska Franz<sup>†</sup>

Mozilla Corporation<sup>\*</sup>  
{gal,brendan,shaver,danderson,dmandelin,mrbkap,graydon,bz,jorendorff,jruderman}@mozilla.com

Adobe Corporation<sup>#</sup>  
{edwsmith,rreitmai}@adobe.com

Intel Corporation<sup>§</sup>  
{mohammad.r.haghhighat}@intel.com

University of California, Irvine<sup>†</sup>  
{mbebenit, changm, franz}@uci.edu

## Abstract

Dynamic languages such as JavaScript are more difficult to compile than statically typed ones. Since no concrete type information is available, traditional compilers need to emit generic code that can handle all possible type combinations at runtime. We present an alternative compilation technique for dynamically-typed languages that identifies frequently executed loop traces at run-time and then generates machine code on the fly that is specialized for the actual dynamic types occurring on each path through the loop. Our method provides cheap inter-procedural type specialization, and an elegant and efficient way of incrementally compiling lazily discovered alternative paths through nested loops. We have implemented a dynamic compiler for JavaScript based on our technique and we have measured speedups of 10x and more for certain benchmark programs.

**Categories and Subject Descriptors** D.3.4 [Programming Languages]: Processors — Incremental compilers, code generation.

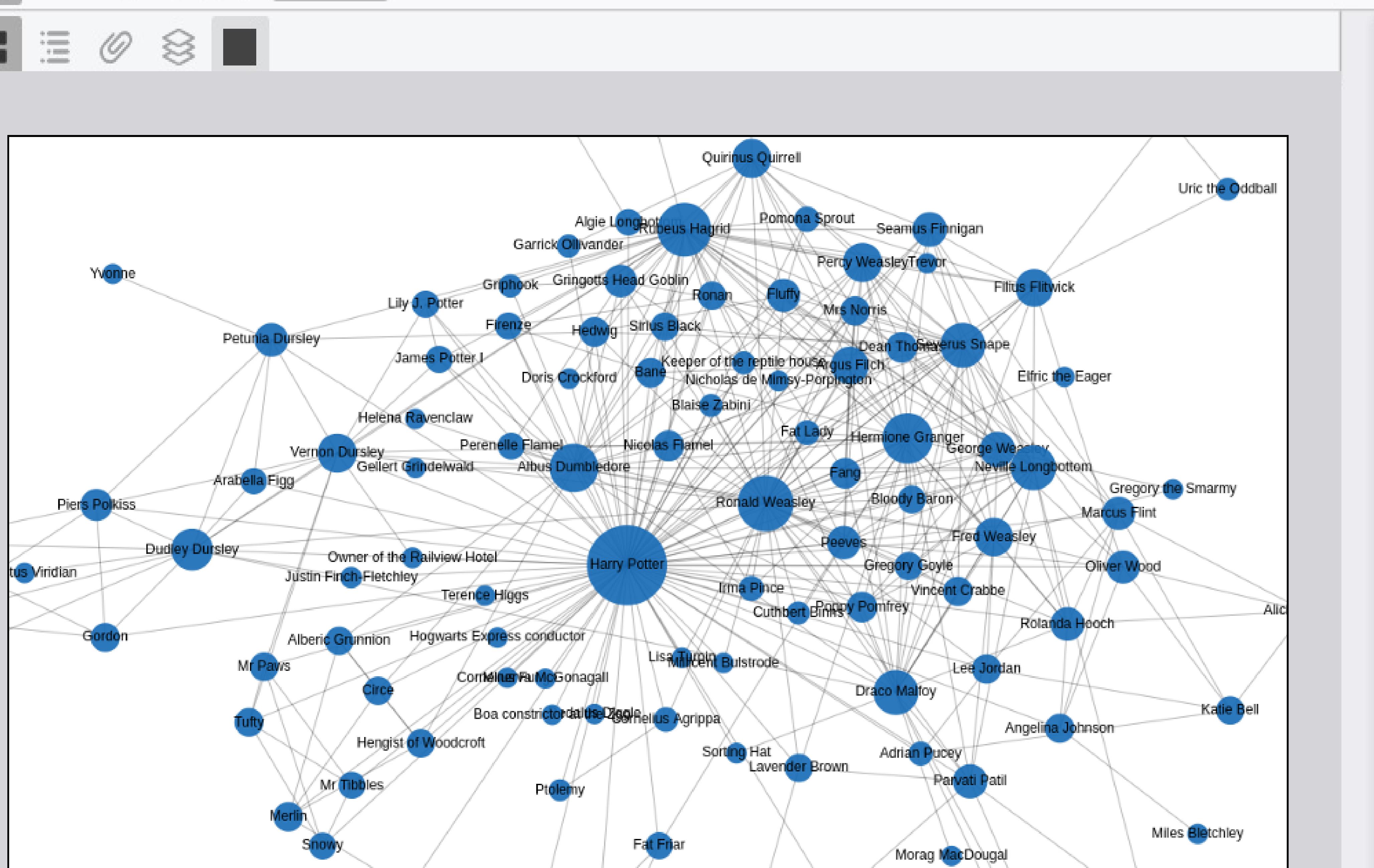
**General Terms** Design, Experimentation, Measurement, Performance.

and is used for the application logic of browser applications such as Google Mail, Google Collaboration Suite. In this domain, in order to provide a good user experience and enable a new generation of web applications, the browser engines must provide a low startup time and fast execution times.

Compilers for statically typed languages generate machine code that is specialized for a specific type. In contrast, compilers for dynamically-typed languages such as JavaScript, the type of variables may vary at runtime. This means that the compiler can no longer generate machine code that is specialized for a specific type. Instead, it must generate machine code that can handle all possible type combinations. While this is a challenging task, it is also a key advantage of dynamically-typed languages. By generating machine code on the fly, the compiler can quickly adapt to changes in the program's state, making it easier to implement complex features like closures and lexical scoping.

We present a trace-based compilation technique for dynamic languages that reconciles speed of compilation with excellent performance of the generated machine code. Our system uses a mixed-

- ↶ Go to First Page
- ↷ Go to Last Page
- ↻ Rotate Clockwise
- ↶ Rotate Counterclockwise
- 🖱 Text Selection Tool
- 🖐 Hand Tool
- ➡ Page Scrolling
- ⬇ Vertical Scrolling
- ↔ Horizontal Scrolling
- ⤻Wrapped Scrolling
- ▬ No Spreads
- Odd Spreads
- Even Spreads
- >Show Cross Page References
- >Show Knowledge Graph
- Generate Summaries
- Highlight Commonalities
- ⓘ Document Properties...



# Summary

# Semantic Scholar:

This work presents an alternative compilation technique for dynamically-typed languages that identifies frequently executed loop traces at run-time and then generates machine code on the fly that is specialized for the actual dynamic types occurring on each path through the loop.

# Huggingface (bart-large-cnn):

Trace-based Just-in-Time Type Specialization for Dynamic languages. We present an alternative compilation technique for dynamically-typed languages that identifies frequently executed loop traces at run-time and then generates machine code on the fly. We have implemented a dynamic compiler for JavaScript based on our technique and we have measured speedups of 10x and more.

# Trace-based Just-in-Time Type Specialization for LLVM Languages

Andreas Gal\*,+ , Brendan Eich\*, Mike Shaver\*, David Anderson\*, David Mandelbaum\*, Mohammad R. Haghigiat\$, Blake Kaplan\*, Graydon Hoare\*, Boris Zbarsky\*, Jason Clegg\*, Jesse Ruderman\*, Edwin Smith#, Rick Reitmaier#, Michael Bebenita+, Mason Chang++#,

Mozilla Corporation\*  
{gal,brendan,shaver,danderson,dmandelin,mrbkap,graydon,bz,jorendorff,jruderman}@

Adobe Corporation<sup>#</sup>  
{edwsmith,rreitmai}@adobe.com

Intel Corporation<sup>§</sup>  
{mohammad.r.haghigraph}@intel.com

University of California, Irvine<sup>+</sup>  
`{mbebenit, changm, ffranz}@uci.edu`

### Abstract

Dynamic languages such as JavaScript are more difficult to compile than statically typed ones. Since no concrete type information is available, traditional compilers need to emit generic code that can handle all possible type combinations at runtime. We present an alternative compilation technique for dynamically-typed languages that identifies frequently executed loop traces at run-time and then generates machine code on the fly that is specialized for the actual dynamic types occurring on each path through the loop. Our method provides cheap inter-procedural type specialization, and an elegant and efficient way of incrementally compiling lazily discovered alternative paths through nested loops. We have implemented a dynamic compiler for JavaScript based on our technique and we have measured speedups of 10x and more for certain benchmark programs.

**Categories and Subject Descriptors** D.3.4 [Programming Languages]: Processors — *Incremental compilers, code generation.*

*General Terms* Design, Experimentation, Measurement, Performance, Test.

- ↖ Go to First Page
  - ↘ Go to Last Page
  - ⟳ Rotate Clockwise
  - ⟲ Rotate Counterclockwise
  - 👉 Text Selection Tool
  - 👉 Hand Tool
  - ➡ Page Scrolling
  - ➡ Vertical Scrolling
  - ➡ Horizontal Scrolling
  - ➡ Wrapped Scrolling
  - ➡ No Spreads
  - ➡ Odd Spreads
  - ➡ Even Spreads
  - ➡ Show Cross Page References
  - ➡ Show Knowledge Graph
  - ➡ Generate Summaries
  - ➡ Highlight Commonalities
  - ⓘ Document Properties...

ompiler can no longer  
structions that operate  
mation, the compiler  
that can deal with all  
ime static type infer-  
on to generate optimi-  
sis is very expensive  
active environment of

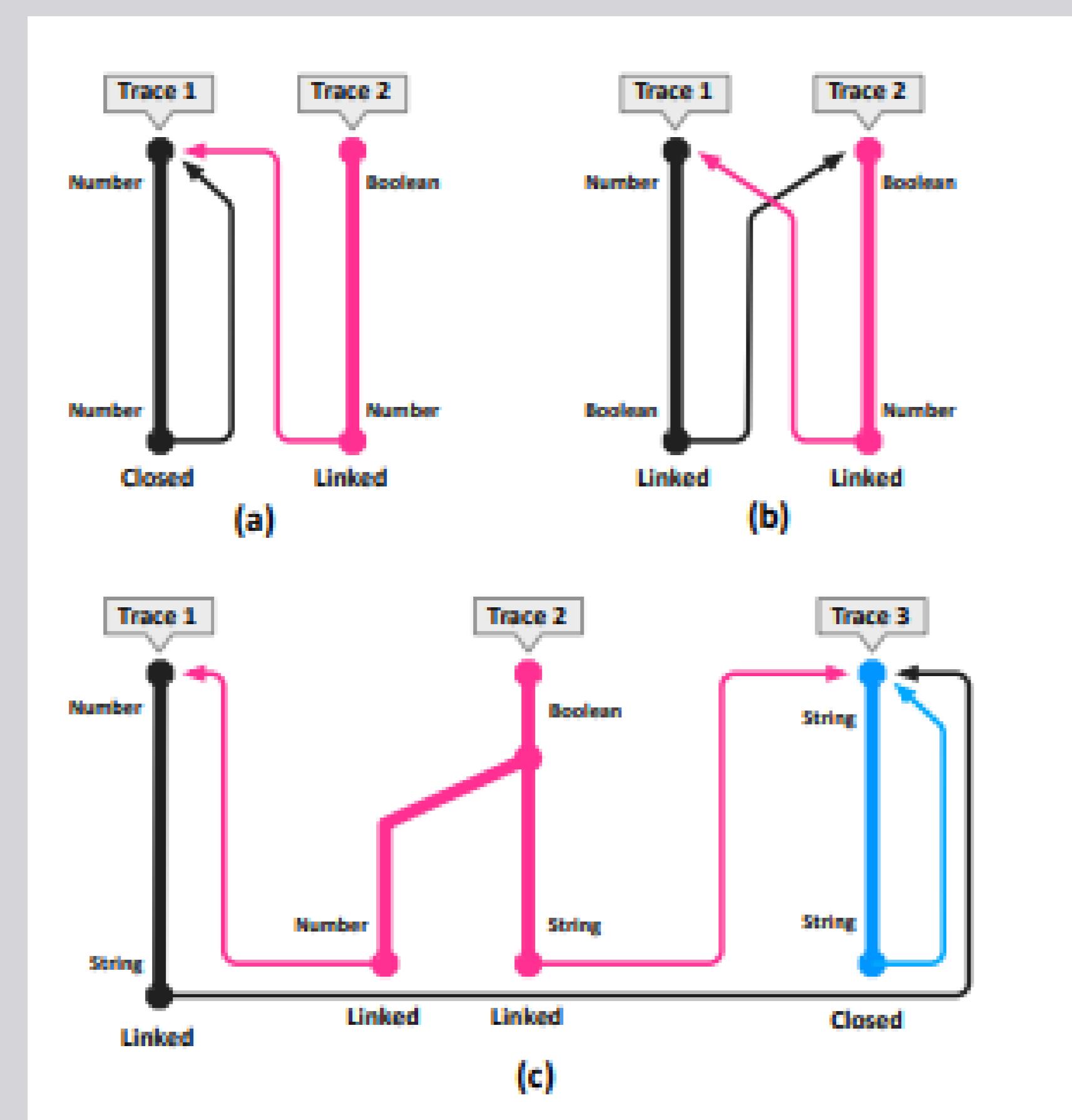
**Figure 1.** Sample program: sieve of Eratosthenes. primes is initialized to an array of 100 false values on entry to this code...

**Figure 2.** State machine describing the major activities of TraceMonkey and the conditions that cause transitions to a new activity. In the dark box...

**Figure 3.** LIR snippet for sample program. This is the LIR recorded for line 5 of the sample program in Figure 1. The LIR encodes...

**Figure 4.** x86 snippet for sample program. This is the x86 code compiled from the LIR snippet in Figure 3. Most LIR instructions compile...

**Figure 5.** A tree with two traces, a trunk trace and one branch trace. The trunk trace contains a guard to which a branch trace was...



**Figure 6.** We handle type-unstable loops by allowing traces to compile that cannot loop back to themselves due to a type mismatch. As...

**Figure 7.** Control flow graph of a nested loop with an if statement inside the inner most loop (a). An inner tree...

**Figure 8.** Control flow graph of a loop with two nested loops (left) and its nested trace tree configuration (right). The outer tree calls...

## Trace-based Just-in-Time Type Specialization for Languages

Andreas Gal<sup>\*,†</sup>, Brendan Eich\*, Mike Shaver\*, David Anderson\*, David Mandelblit\*, Mohammad R. Haghaghian<sup>\$</sup>, Blake Kaplan\*, Graydon Hoare\*, Boris Zbarsky\*, Jason O’Riordan\*, Jesse Ruderman\*, Edwin Smith#, Rick Reitmaier#, Michael Bebenita<sup>†</sup>, Mason Chang<sup>†,‡</sup>, Mozilla Corporation\*

{gal,brendan,shaver,danderson,dmandelin,mrbkap,graydon,bz,jorendorff,jruderman}@mozilla.com

Adobe Corporation<sup>#</sup>  
{edwsmith,rreitmai}@adobe.com

Intel Corporation<sup>\$</sup>  
{mohammad.r.haghaghian}@intel.com

University of California, Irvine<sup>†</sup>  
{mbebenit,changm,franz}@uci.edu

### Abstract

Dynamic languages such as JavaScript are more difficult to compile than statically typed ones. Since no concrete type information is available, traditional compilers need to emit generic code that can handle all possible type combinations at runtime. We present an alternative compilation technique for dynamically-typed languages that identifies frequently executed loop traces at run-time and then generates machine code on the fly that is specialized for the actual dynamic types occurring on each path through the loop. Our method provides cheap inter-procedural type specialization, and an elegant and efficient way of incrementally compiling lazily discovered alternative paths through nested loops. We have implemented a dynamic compiler for JavaScript based on our technique and we have measured speedups of 10x and more for certain benchmark programs.

**Categories and Subject Descriptors** D.3.4 [Programming Languages]: Processors — Incremental compilers, code generation.

**General Terms** Design, Experimentation, Measurement, Performance

and is used for the application logic of browser applications such as Google Mail, Google Collaboration Suite. In this domain, in order to provide a good user experience and enable a new generation of applications, the browser engines must provide a low startup time and fast execution times.

Compilers for statically typed languages generate efficient machine code. In contrast, for dynamically-typed languages such as JavaScript, the type information may vary at runtime. This means that the compiler can no longer easily transform operations into machine instructions that operate on one specific type. Without exact type information, the compiler must emit slower generalized machine code that can deal with all potential type combinations. While compile-time static type inference might be able to gather type information to generate optimized machine code, traditional static analysis is very expensive and hence not well suited for the highly interactive environment of a web browser.

We present a trace-based compilation technique for dynamic languages that reconciles speed of compilation with excellent performance of the generated machine code. Our system uses a mixed-