

## Objective:

The following task will have two subtasks

1. Is to **create a template based on the designs given.**
2. Is to **create a generator that generates the html template, that which when opened, opens the above template.**

**Make sure to read the whole document before starting the task. The framework of choice used does not matter. The implementation needs to be right.**

## Materials:

PDF Link: <https://mozilla.github.io/pdf.js/web/viewer.html>

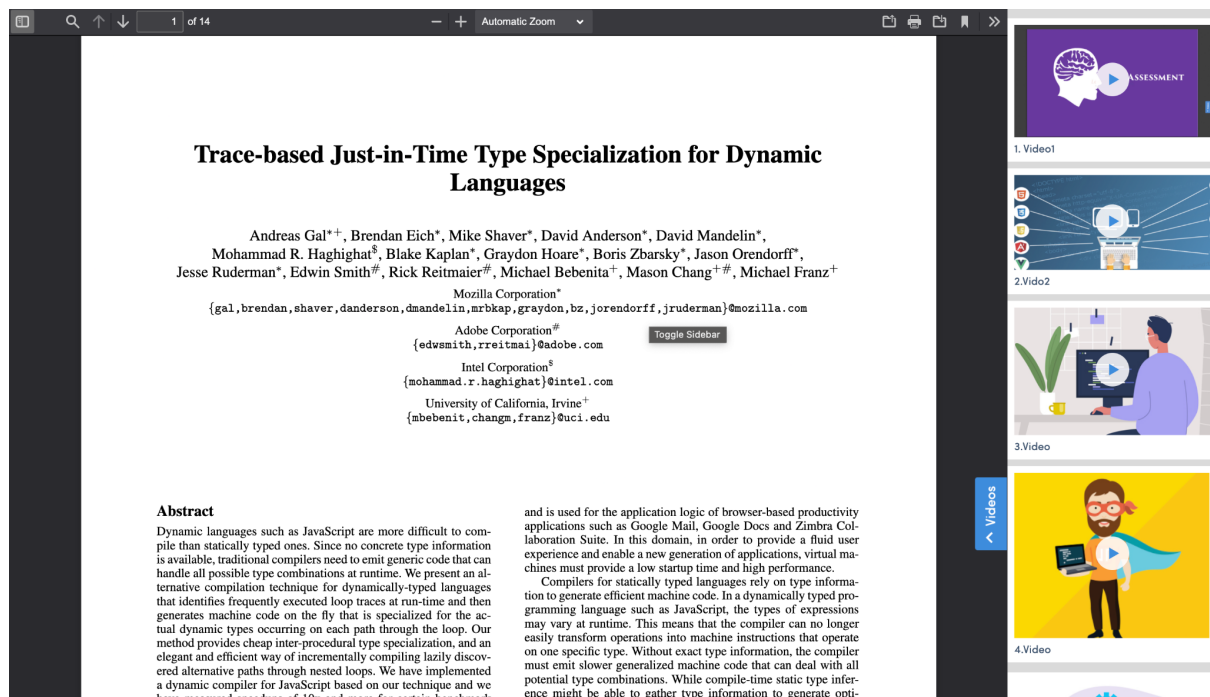
Video Link: <http://techslides.com/demos/sample-videos/small.mp4>

Hosted Image Link: <https://i.ibb.co/h9xhFxN/Screenshot-2021-10-08-at-3-26-26-PM.png>

## Details of Task 1:

The template consists of two parts, the left and the right.

- The left part contains an **iframe which includes the PDF**
- The right part is a **collapsible sidebar** that contains a list of videos , that when clicked opens a Modal , which can play a video.



of 14

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

Andrew Gai<sup>1</sup>, Brendan Eich<sup>2</sup>, Mike Shaver<sup>3</sup>, David Anderson<sup>4</sup>, David Mandelin<sup>5</sup>,  
 Mohammed R. Haghighat<sup>6</sup>, Blake Kaplan<sup>7</sup>, Graydon Hoover<sup>8</sup>, Boris Zharsky<sup>9</sup>, Jason Orendorff<sup>9</sup>,  
 Jesse Ruderman<sup>9</sup>, Edwin Smith<sup>9</sup>, Karl Retzmaier<sup>9</sup>, Michael Beheravi<sup>9</sup>, Maocen Chang<sup>9</sup>, Michael Franz<sup>9</sup>

Mozilla Corporation<sup>1</sup>

{gai,brendas,shaver,anderson,mandelin,retzmaier,graydon,hoover,orendorff,beheravi,kaplan}@mozilla.com

Adobe Corporation<sup>2</sup>

{semitext,rutman}@adobe.com

Intel Corporation<sup>3</sup>

{mohammed\_r\_haghighat}@intel.com

University of California, Irvine<sup>4</sup>

{mohammed,maocen,graydon}@uci.edu

### Abstract

Dynamic languages such as JavaScript are more difficult to compile than statically typed ones. So no concrete type information is available, traditional compilers need explicit 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 trunks 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 highly 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.3.1 [Programming Languages]: Processes — Incremental compilation, code generation.

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

**Keywords:** JavaScript, Just-in-time compilation, trace-based.

### 1. Introduction

Dynamic languages such as JavaScript, Python, and Ruby, are popular since they are expressive, accessible to non-experts, and easy to deployment as well as distributing a small file. They are used for small scripts as well as for complex applications. JavaScript, for example, is the de facto standard for client-side web programming

and is used for applications of all browser-based productivity categories such as Google Mail, Google Docs and Zimbra Collaboration Suite. In the future, it is likely to provide a full 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 dynamically typed programming languages such as JavaScript, the types of expressions may vary at runtime. This means that the compiler can no longer easily transform operations into machine instructions that operate on the specific type. Without such type information, the compiler must either generate 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-model execution approach: the system uses the JavaScript engine in a fast-running byte-code interpreter. As the program runs, the system identifies frequently executed bytecode sequences, records them, and compiles them to fast native code. We call this a sequence of instructions a trace.

Unlike machine-based dynamic compilers, our dynamic compiler operates at the granularity of individual loops. This design choice is based on the expectation that programs spend most of their time in loop trunks. Even in dynamically typed languages, we expect hot loops to be mostly type-stable, meaning that the types of values are invariant. (12) For example, we would expect loop counters that act as integers to remain integers for all iterations. When both of these expectations hold, a trace-based compiler can capture the program execution with a small number of type specializations, or statically compiled traces.

Each compiled trace covers one path through the program with one sequence of values to types. When the VM executes a compiled trace, it cannot guarantee that the same path will be followed on the same types will occur in subsequent loop iterations.

*Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted by ACM for users registered with ACM, provided that the fee of \$5.00 per copy is paid directly to ACM. For those organizations that have been granted a special rate of payment, the fee code must appear with the code at the top of the page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or fee.*  
 Copyright 2006 ACM 978-1-59593-395-9/06/0008 ...\$5.00

Here, recording and compiling a trace replaces what the path and typing will be exactly as they were during recording for subsequent iterations of the loop.

Every compiled trace contains all the *payloads* (behind the required to initialize the operation; i.e. the values of the payload fields of control flow, a difference, or a value of a different type is generated), the trace exits. If an exit becomes hot, the VM can convert it back into a branch exiting the exit to cover the next path. In this way, the VM can have a new trace covering all of the hot paths through the loop.

Second loops can be different. When the inner loop is fast, the VM is a naive implementation, inner loops would become hot first, and the VM would start tracing them. When the inner loop is slow, the VM would detect that a different branch was taken. The VM would be forced to branch twice, but the first time the trace reaches the inner loop header, but the outer loop header. At this point, the VM would start tracing entry code that leads to the inner loop header again. But this requires tracing a copy of the outer loop for every side exit from the inner loop. This is not ideal. Instead, the VM would use a set of unrolled tail loops, which can easily overflow the code space. Alternatively, the VM would employ entry tracing, and give up on outer tracing outer loops.

We solve the nested loop problem by recording nested loop trunks. Our system traces the inner loop trunk by executing nested loop trunks. The inner loop trunk is a trace that starts at the inner loop header. When the outer loop reaches the inner loop header, the system starts to call the trace for the inner loop. If the call succeeds, the VM records the inner loop trunk. If the call fails, the VM records the outer loop trunk. The VM would then start tracing the outer loop trunk.

```
1 for Outer = 2; i < 100; ++i {
2   if (TypeInt(i))
3     continue;
4   for (i = 1; i < 100; ++i) {
5     print(i + false);
6   }
7 }
```

Figure 1. Sample program code of *Exhaustion*, *print* is initialized to an array of 100 false values on entry to this code sample.

Videos

# Trace-based JS-i

Andreas Gal<sup>1\*</sup>, Bernd  
Mohammadi R. Haghighat<sup>2</sup>,  
Jesse Ruderman<sup>3</sup>, Edwin Smith<sup>4</sup>,

{gal, berndma, shaver, andersa}

## 1. Video1

## 2.Video2

## 3.Video

**Abstract**

Dynamic languages such as JavaScript are so popular that virtually every site, from a small business to a large corporation, is available, traditional compilation need to meet handle all possible type combinations at runtime. Iterative compilation technique for dynamic languages frequently executed loop traces generates machine code on the fly that is optimal dynamic types occurring in each path. It method provides cheap static procedural type's elegant and efficient way of incrementally consider alternative paths through nested loops. W a dynamic compiler for JavaScript based on we have measured speedups of 10x and more for programs.

**Categories and Subject Descriptors** D.3.4 [general]: Processors — Incremental compilers

**General Terms** Design, Experimentation, S. name.

**Keywords** JavaScript, just-in-time compilation

### 1. Introduction

Dynamic languages such as JavaScript, Python also since they are expressive, accessible to deployment as easy as distributing a source file small scripts as well as for complex application example, in the de facto standard for client-side

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made for commercial sale and the copies bear this notice and the full copyright notice and address to the publisher. For more information, contact the publisher or the copyright owner.

PLDI'09, June 15–20, 2009, Beijing, China.  
Copyright 2009 ACM 978-1-60558-162-9/09/06...\$5.00

### Videos

Here, recording and copying will be exactly a iteration of the loop.

Every compiled trace consists at the point to validate the speculation. If one of the flow is different, or a value of a different type trace exits. It is an exit because the VM would start at the exit to cover the new path records a trace type covering all the loop paths.

Nested loops can be difficult to optimize a naive implementation, inner loop would the VM would start tracing them. When the VM would detect that a different branch was to try to record a branch trace, and find that the inner loop head, but the outer loop head, it could continue tracing until it reached the inner loop tracing the outer loop inside that trace. It like this requires tracing a copy of the outer loop and type combination in the inner loop. In as of unintended tail duplication, which can cause cache. Alternatively, the VM could simply stop on every tracing outer loops.

We solve the nested loop problem by no trace. Our system traces the inner loop exactly The system stops extending the inner trace who loop, but then it starts a new trace at the outer loop trace reaches the inner loop head, if the trace trace for the inner loop. If the call the new trace to the outer loop.

The image is a screenshot of a presentation slide. At the top, there is a dark navigation bar with icons for back, forward, and search, and a status bar showing '1 of 14'. The slide title is 'Trace-based Just-in-Time Type Specialization for Dynamic Languages'. Below the title, the authors are listed: Andreas Gal<sup>\*,†</sup>, Brendan Eich<sup>\*</sup>, Mike Shaver<sup>\*</sup>, David Anderson<sup>\*</sup>, David Mandelin<sup>\*</sup>, Mohammad R. Haghighat<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>, Michael Franz<sup>+</sup>. The affiliations are listed below: Mozilla Corporation<sup>\*</sup> (with email addresses {gal, brendan, shaver, danderson, dmandelin, mrbkap, graydon, bz, jorendorff, jruderman}@mozilla.com), Adobe Corporation<sup>#</sup> (with email address {edwsmith, rreitmai}@adobe.com), Intel Corporation<sup>‡</sup> (with email address {mohammad.r.haghighat}@intel.com), and University of California, Irvine<sup>+</sup> (with email address {mbebenit, changm, franz}@uci.edu). The slide is divided into two main sections: 'Abstract' and 'and is used for the application logic of browser-based productivity applications...'. The 'Abstract' section describes the challenges of dynamic languages and the proposed technique. The second section, which is partially cut off, describes how the technique is used for application logic.

## 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. Haghighat<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>, 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.haghighat}@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

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 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 outli-





## Details of Task 2:

The aim of the following task , is to create a HTML Template Generator that takes in some amount of data i.e required to create the above template.

Use Cases that needs to be implemented:

1. A place to **input the PDF Link**.
2. A provision to **Add Multiple Video data**.
3. A provision to **Remove already added Video**.
4. A provision of a button that on click **downloads the HTML File**.
5. If the HTML file is opened it should **open the above template with the data given**.

PDF Link\*

Add your pdf link here



Generate HTML

PDF Link\*

Add your pdf link here

+

Add video title here

Add video thumbnail URL

Add video URL



Generate HTML

PDF Link\*

Add your pdf link here

! Please fill in this field.

+

Add video title here

Add video thumbnail URL

Add video URL

+

Add video title here

Add video thumbnail URL

Add video URL



Generate HTML

## Backend Task Details

- You need to make database schema for same
- Also you need to show list of template that is generate from generate HTML
- You need to show below mention data in list in latest order
- Created\_at time format should be human diffraction pattern (5 minutes ago, 1 hour ago, 2 days ago)
- Using Download Button user can download that template
- Using preview button he can see that template preview

No	Template Id	Created A	Preview	Download
01	100.qrtzfxer	1 Hour ago	LINK	Download