9-1-2013

Better Inferred Types for JavaScript

John Sarracino
Harvey Mudd College

Better Inferred Types for JavaScript

John Sarracino

Types Prevent Crashes

Statically typed languages guarantee that some run-time errors cannot occur. For example, if the user submits 0 to Figure 1, an integer gets treated as an object. A type system would eliminate this possible error.

```
var y, z;
y = input('1 or 0?');

if (y == 1) {
  z = {}; // z is an object
  z.s1 = 5;
  z.s2 = 7;
  z.s3 = 9;
} else {
    z = 2; // z is an int
}

// type error if int
write(z.s1 + z.s2 + z.s3);
```

Figure 1: Possible type-error in JavaScript

JavaScript Lacks Types

JavaScript is a *dynamically typed* language, so variable types are checked at run-time. For example, the type of variable z in Figure 2 is determined to be either object or int during the program's execution.

```
var y, z;
y = input('1 or 0?');

if (y == 1) {
  z = {}; // runtime object
  z.s1 = 5;
  z.s2 = 7;
  z.s3 = 9;
} else {
  z = 2; // runtime int
}

write(z.s1 + z.s2 + z.s3);
```

Figure 2: Dynamically typed variable in JavaScript

Added vs. Inferred Types

There are two approaches to the benefits of static types to a dynamically typed language:

Explicit types are annotations added by the programmer, identical to a static type system. They require much less work from the compiler, but require a completely new language specification.

Inferred types are calculated by the compiler. They require no extra work by the programmer and operate on the same base language, but are less powerful than explicit types. For example, in Figure 3, the compiler can infer the type of y to be String.

Our work focuses on inferred types, but both techniques are active areas of research.

```
var y, z;
y = input('1 or 0?');
 // input always returns string types

if (y == 1) {
  z = {};
  z.s1 = 5;
  z.s2 = 7;
  z.s3 = 9;
} else {
  z = 2;
}

write(z.s1 + z.s2 + z.s3);
```

Figure 3: Inferred String Type

Costs of Inferred Types

Unfortunately, due to undecidability, inferred type systems often can't assign a single type to a variable. For example, an inferred type system would determine z's possible type in Figure 4 to be both object and int.

```
var y, z;
y = input('1 or 0?');

if (y == 1) {
  z = {}; // z could be object...
  z.s1 = 5;
  z.s2 = 7;
  z.s3 = 9;
} else {
  z = 2; // and could be int...
}

// so z could be
// either object or int
if (typeof z == "object") {
  write(z.s1 + z.s2 + z.s3);
} else {
  write(z);
}
```

Figure 4: Imprecise inferred types

Our Improvement

Our work takes advantage of information in conditional checks to make an inferred type system more precise [1]. For example, in Figure 5, our work precisely determines the type of z to be object in the true branches and int in the false branches.

```
var y, z;
y = input('1 or 0?');

if (y == 1) {
  z = {};
  z.s1 = 5;
  z.s2 = 7;
  z.s3 = 9;
} else {
  z = 2;
}

// typeof check forces z's type
// no inferred errors!
if (typeof z == 'object') {
  write(z.s1 + z.s2 + z.s3);
} else {
  write(z);
}
```

Figure 5: Precise inferred types

Impact

Since our inferred types are more accurate, we present fewer possible errors to the programmer. For example, a naïve type inference algorithm reports possible errors in Figure 6, while our system certifies it as error-free.

```
var y, z;
y = input('1 or 0?');

if (y == 1) {
  z = {};
  z.s1 = 5;
  z.s2 = 7;
  z.s3 = 9;
} else {
  z = 2;
}

// inferred warnings
if (typeof z == 'object') {
  write(z.s1 + z.s2 + z.s3);
} else {
  write(z);
}
```

Figure 6: Unneccesary warnings due to imprecise types

Acknowledgements & References

In addition to the authors, the Programming Languages Lab at UCSB was essential to this work. All figures are intellectual property of Ben Wiedermann and John Sarracino. This work was supported by NSF CCF-1117165.

[1] Vineeth Kashyap, **John Sarracino**, John Wagner, Ben Wiedermann, and Ben Hardekopf. Type Refinement for Static Analysis of JavaScript. In Proceedings of the 9th Symposium on Dynamic Languages, DLS '13, pages 17-26, New York, NY, USA, 2013. ACM. Located at http://doi.acm.org/10.1145/2508168.2508175.

Advisors: Ben Wiedermann(HMC) , Ben Hardekopf(UCSB)