Recency Types for JavaScript (Extended Abstract)

Phillip Heidegger Peter Thiemann
Universität Freiburg, Germany
{heidegger,thiemann}@informatik.uni-freiburg.de

April 11, 2009

Scripting languages are popular because their dynamic type systems accelerate the development of short programs. For larger programs, static analysis becomes an important tool for detecting programming errors.

We specify such an analysis as a type system for a call-by-value lambda calculus with objects and prove its type soundness. The calculus models essential features of JavaScript, a widely used dynamically-typed language: objects as property maps, type change during object initialization, and prototypes.

In JavaScript, the type of a property can change with every assignment. As long as a property is not yet assigned, accessing it returns the value `undefined`, which is a regular JavaScript value. Hence, in a flow-insensitive type system, each property type would have to contain the type `undefined` as a subtype (e.g., as component of a union type). As it is a run-time error to dereference `undefined` or to call it as a function, the type system must reject any attempts to use the property in this way. Although JavaScript converts `undefined` to a string, a number, or a boolean as needed, this conversion may not be intended by the programmer. Hence, the imprecise type gives rise to too many false positives.

We tackle this problem in a novel way with *recency types*. Recency types arise in a flow-sensitive type system, which distinguishes between the most recently created object at some program point and all objects previously created at the same program point. For each object creation site, it assigns one type to the most recent object and another type to all remaining objects. The crucial observation is that the type of the most recent object describes exactly one object. Hence, any update to the value of this object can be precisely reflected as a strong update in the type of the object.

Recency types fit the typical initialization pattern in imperative languages where the programmer allocates a number of objects and then initializes these objects. Quite often, no further properties are defined after the initialization and the type of the properties rarely changes. During the initialization phase, objects are most recent and their types can be updated in a flow sensitive manner. Later on, when the shape of the object does not change anymore, it falls back to flow insensitive typing.

The idea of a recency abstraction can be fruitfully transported to a type system. This type system is amenable to analyzing programs in scripting languages, in particular handling object initialization. Its distinguishing feature is the ability to handle type changes and prototypes precisely.

We designed and implemented a type inference algorithm and exercised it with example programs. We are extending the implementation with the goal to apply it to realistic programs.