Eduardo Munoz
Magdalene College
em487

Computer Science Tripos Part II Project Proposal `draft`

Multilanguage programming in F# and JavaScript

October 14, 2012

Project Originator: *Eduardo Munoz*

Project Supervisor: *Tomas Petricek*

Signature:

Director of Studies: *Dr John Fawcett*

Signature:

Overseers: *Dr S. Teufel* and *Dr J. Crowcroft*

Signatures:

1 Introduction and description of the work

Selecting the programming language to use when implementing a certain algorithm is crucial in the success of the project. For this reason, large scale software systems tend to be written in several languages. However, the interactions between those components may be an issue. There are several approaches to tackle this:

1. Foreign function interfaces (FFI): mechanism by which a program written in one language may call procedures from a program in a different language.
2. Multilanguage runtimes: Several programming languages target the same architecture, allowing a richer interaction between languages: e.g. inherit classes in one language defined in another. Examples of multilanguage runtimes are Java, Scala, Jython, etc. targeting the Java Virtual Machine; all programs written for the .NET framework target the CLR.
3. Embedded interpreters: implementing an interpreter of the target language in the host language and use a type-indexed embedding and projection algorithm.

In this project, we intend to explore the ideas described in [2], for an ML-like language and JavaScript. This includes producing an implementation of the *lump embedding* and the *natural embedding*. The former concept is related to FFI frameworks (such as the Java Native Interface), where each environment sees foreign values as "lumps" (values with an opaque type). The natural embedding provides a richer interoperability between the two languages, allowing values to be converted across the boundary from one language to the other. The implementation for the natural embedding can be seen as FFI with some properties of multilanguage runtimes.

An example of using the lump embedding (note that the syntax hasn't been decided yet, this is for illustrative purposes):

```
(* implementation missing; types subject to change *)
let eval_js x : JS list->JS = raise notImplemented;
let apply_lump f x = eval_js([f,x]);
let succ:Lump = JS("function(a) {return a+1;}");
in apply_lump(succ, JS(3));

> val it: JS(4) : Lump
```

In this case, the ML-like language can only interact with values of type `Lump` to interoperate with JavaScript (using the type constructor `JS`).

An example of using the natural embedding (note that the syntax hasn't been decided yet, this is for illustrative purposes):

```
let test (x:string->unit) :unit = x("testing");
in test(JS(string, unit, "function(s) {print(s)}"));
> val it: () : unit
```

2 Starting point

This project will involve material from *Types* (deal with language boundaries), *Semantics of Programming Languages* (specify the behavior of the framework), *Compiler Construction* (analyze the runtime of the JavaScript engine to integrate it) and *Foundations of Computer Science* (functional programming, ML).

3 Resources required

The implementation will be completed on my own laptop.

I will make use of a JavaScript engine (e.g. *V8*) and an ML-like functional programming language (OCaml or F#).

4 Substance and structure of the project

The goal of this project is to design and implement a framework for the multilanguage programming paradigm. Some aspects in which two programming languages can differ in are:

1. *Type system*: JavaScript is an untyped[1] language, while F# has strong static typing with type inference.
2. *Evaluation rules*: for instance, JavaScript allows the evaluation of two empty lists [1]; OCaml however forbids this operation because it doesn't type check:

[1]There is some confusion with the terms *dynamically typed* and *untyped*. In the academic literature, the term dynamically typed was introduced much later than untyped to mean the same concept. Perhaps the best categorization for JavaScript is *weakly dynamically typed*.

```
(JavaScript)
js> [] + []

// (empty string)

--------------------
(OCaml)
# [] + [];;
Error: This expression has type 'a list
       but an expression was expected of type int
```

3. *Values*: for instance, JavaScript treats all numbers as floating point numbers:

```
(JavaScript)
js> 1000000000000000000 + 1
1000000000000000000
js> 4611686018427387903 + 1
4611686018427388000
--------------------
(OCaml)
# 1000000000000000000 + 1;;
- : int = 1000000000000000001
# 4611686018427387903 + 1;;
- : int = -4611686018427387904
```

4. *Syntax*: JavaScript's syntax is loosely based on that by C and Java, whereas the syntax of F# is that of the ML-language family.

5. *Purpose*: JavaScript is the language of the web, used mainly to enhance user interfaces and dynamic websites. F# is a general purpose language.

Work will have to be carried out in the following areas:

1. *Type system*: handle types at the boundaries of the programming languages, making sure the type soundness of F# is preserved (possible use of contracts).

2. *Evaluation rules*: generate glue / wrapper code that allow treating a JavaScript function as F# function (which might need contracts).

3. *Values*: access JavaScript values from the F# runtime using the JavaScript engine.

4. *Syntax*: define a clear interface with which JavaScript code will be embedded in F# source files (either inline or loading a file), both for the lump and the natural embedding.

5 Possible extensions

1. Syntax-check the JavaScript syntax using F# type providers.

6 Evaluation strategy

Quantitative:

- Compare the performance difference between the system implementing the lump embedding and the natural embedding.
- Estimate the relationship between the number of foreign boundary crossings and the execution time.
- Compare the performance difference with other systems that allow some interoperability between F# and JavaScript.

Qualitative:

- Show the expressiveness of the system.
- Proof of correctness, by executing some tests.

7 Backup strategy

All source files (code and LaTeX) are in my local machine in a *git* repository, which is hosted at *Bitbucket* and also replicated to an external hard drive. The git repository will be useful when writing the dissertation as it will be used as a work log.

8 Success criteria

1. The resulting framework should not take a significant amount of time than executing the respective monolingual runtimes.
2. The lump embedding implementation should be able to pass values from JavaScript to F# and then pass them back.
3. The natural embedding should be able to pass a function from JavaScript to F# (and the other way round) and invoke it on the other side of the boundary

4. A convenient syntax for multilanguage programming has been designed.
5. Tests of correctness have been passed.

9 Timetable and milestones

Initial preparation: 18.10.2012 - 31.10.2012

Install F# on my laptop (using the Mono framework for .NET); install Windows as a fallback. Revise ML and become familiar with the differences in F#. Keep reading research papers and articles about the subject. Research JavaScript engines and decide which one offers the best API to access JavaScript values.
Milestone:

01.10.2012 - 14.11.2012

Decide on a specific syntax for embedding JavaScript inside F#.
Milestone:

15.11.2012 - 28.11.2012

A
Milestone:

29.11.2012 - 12.12.2012

B
Milestone:

13.12.2012 - 26.12.2012

C
Milestone:

27.12.2012 - 09.01.2012

D
Milestone:

10.01.2012 - 23.01.2012

E
Milestone:

24.01.2012 - 06.02.2012

Write the introduction, preparation and implementation sections.
Milestone: the dissertation document has been started.

07.02.2012 - 20.02.2012

Finish off the following sections: preparation and implementation. Take evaluation metrics.
Milestone: preparation and implementation are finished. Evaluation data is gathered.

21.02.2012 - 06.03.2012

Write the evaluation and conclusion sections of the dissertation.
Milestone: a draft of the whole dissertation is now complete.

07.03.2012 - 20.03.2012

Make formatting changes to the document. Send final draft to my supervisor and make small adjustments if suggested.
Milestone: the documents formatting is final.

21.03.2012 - 03.04.2012

Make any last-minute minor changes to the dissertation / code if suggested by my supervisor or DoS.
Milestone: have the dissertation approved, hand it in.

References

[1] ECMA. *ECMA-262: ECMAScript Language Specification.* ECMA (European Association for Standardizing Information and Communication Systems), pub-ECMA:adr, third edition, December 1999.

[2] Jacob Matthews and Robert Bruce Findler. Operational semantics for multi-language programs. *ACM Trans. Program. Lang. Syst*, 31(3), 2009.