Leveraging dynamic typing through static typing

Paola Giannini[1], Daniele Mantovani[2] and Albert Shaqiri[3]

[1]Dipartimento di Informatica, Università del Piemonte Orientale
[2]Algorithmedia s.r.l.[3]Dipartimento di Informatica, Università del Piemonte Orientale, Algorithmedia s.r.l.

2012 - Varese, 19 September 2012

Brief

- F# $\rightarrow$ JavaScript compiler
- Take advantage of both statically and dynamically typed languages allowing this way for type safe (meta)programming in dynamic environments such as those of many web applications

Dynamically typed languages

Pros:

- (Usually) simple
- No type annotations (not always a pro)
- Dynamic type checking and automatic type casting can shorten programming time
- Usually higher level $\rightarrow$ less code
- Quick prototyping and scripting
- Metaprogramming

Cons:

- The absence of type annotations $\rightarrow$ poor documentation
- The absence of a rigorous type checker introduces serious difficulties in developing and maintaining medium to big size applications
- Unexpected application behaviour
- More run-time errors
- Onerous debugging (run-time errors long after the error occured)

Statically typed languages

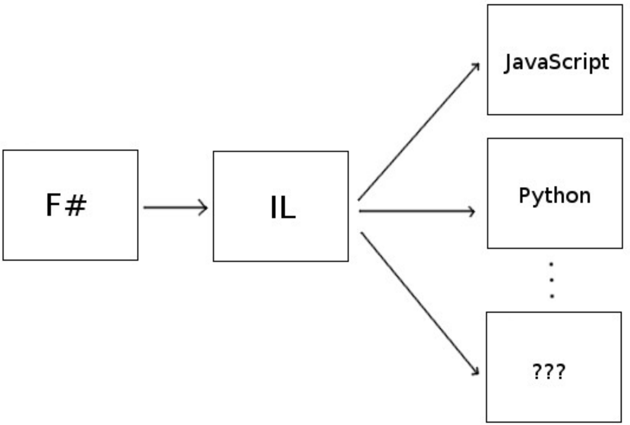
Pros:

- A static type checker guarantees the absence of certain type of errors
- Type annotations $\rightarrow$ documentation
- Performance

Cons:

- Verbose (even though with type inference you can reduce verbosity, e.g. F#, Ocaml, etc.)

Architecture



Supported features

- Currenlty supported target languages: JavaScript, Python
- Easy integration of new target languages
- When possible, we translate by direct mapping or by using target language primitives to obtain semanticaly equivalent behavior
- Namespacing, pattern matching, classes, discriminated unions, etc.
- Distinction between statements and expressions

Core syntax

$$
\begin{array}{llll}
s & ::= & e \mid st; s & \text{seq. statem.} \\
st & ::= & u{:=}e \mid \texttt{let } x{:}t{=}e \mid \texttt{let! } u{:}t{=}e & \text{statements} \\
e & ::= & x \mid u \mid n \mid \texttt{tr} \mid \texttt{fls} \mid e_1 \texttt{+} e_2 \mid \texttt{fun } x{:}t\texttt{->}s \mid e_1\ e_2 & \\
 & & \mid \texttt{stm2exp}(s, \{u_1{:}t_1, \ldots, u_n{:}t_n\}) & \\
 & & \mid (\texttt{int})e \mid (\texttt{bool})e \mid \texttt{exc } e \mid \texttt{if } e \texttt{ then } s_1 \texttt{ else } s_2 & \text{expressions} \\
t & ::= & \texttt{int} \mid \texttt{bool} \mid t_1 \to t_2 \mid \langle u_1{:}t_1, \ldots, u_n{:}t_n \rangle t & \text{types} \\
v & ::= & n \mid \texttt{tr} \mid \texttt{fls} \mid \texttt{fun } x{:}t\texttt{->}s \mid \texttt{stm2exp}(s, \{u_1{:}t_1, \ldots, u_n{:}t_n\}) & \text{values}
\end{array}
$$

Figure : Syntax of core intermediate language

Sequence of expressions to sequence of statements

Suppose we have to calculate the 7th Fibonacci number and store the information if the number is even or odd. We could write it this way:

F#

```
let mutable even = false
let x =
    let rec fib x =
        if x < 3 then 1
        else fib(x - 1) + fib(x - 2)
    let temp = fib 7
    even <- (temp % 2 = 0)
    temp
x
```

Intermediate language

```
let! even = false;
let y = stm2exp(
    let fib = fun x:int ->
            if x < 3 then return 1
            else return (fib (x-1) + fib (x-2));
    let temp = fib 7;
    even := temp % 2 = 0;
    return temp;,
  {even:bool});
let x = exc y
return x;
```

Translation of F# sequence of expressions in the intermediate language

Sequence of expressions to sequence of statements (JS)

Direct mapping to JavaScript	Correct translation to JavaScript

<pre>var even = false;
var x =
  var fib = function (x) {
    if (x < 3)
      return 1;
    else
      return fib(x - 1) + fib(x - 2);
  };
  var temp = fib(7);
  even = (temp % 2) == 0;
  temp;
return x;</pre>
	<pre>(function() {
	  var even = false;
	  var x = (function () {
	    var fib = function (x) {
	      if (x < 3)
	        return 1;
	      else
	        return fib(x - 1) + fib(x - 2);
	    };
	    var temp = fib(7);
	    even = (temp % 2) == 0;
	    return temp;
	  })();
	  return x;
	})();</pre>

Wrong and Correct JavaScript translations

Sequence of expressions to sequence of statements (Py)

- stm2exp is mapped into a top-level function
- We call the function where stm2exp was in the IL
- Now even is out of the scope and thus is undefined!

```
def temp1():
  def temp2(fib, x):
    if (x < 3):
      return 1
    else:
      return fib(x - 1) + fib(x - 2)

  fib = lambda x: temp2(fib, x)
  temp = fib(7)
  # ERROR!!! even is undefined
  even = ((temp % 2) == 0)
  return temp

def __main__():
  even = false;
  x = temp1()
  return x

__main__();
```

Wrong translation in Python

Sequence of expressions to sequence of statements (Py)

- We pass even (by reference) to the temporary function
- Wrapping into ByRef and unwrapping are done automatically by the compiler

```
def temp1(even):
  def temp2(even, fib, x):
    if (x < 3):
      return 1
    else:
      return fib(x - 1) + fib(x - 2)

  fib = lambda x: temp2(even, fib, x)
  temp = fib(7)
  even.value = ((temp % 2) == 0)
  return temp

def __main__():
  even = false;
  wrapper1 = ByRef(even)
  x = temp1(wrapper1)
  even = wrapper1.value
  return x

__main__();
```

Correct translation in Python

Dynamic type checking

F#

```
let add x y = x + y

val add : int -> int -> int
```

Intermediate language

```
let add = fun x:int ->
    return fun y:int ->
        return (int)x+(int)y;
```

JavaScript without type casting

```
function add(x) {
  return function(y) {
    return x + y;
  }
}

add(4)("foo"); // "4foo"
```

JavaScript with forced type casting

```
var add = function (x) {
    return function(y) {
      return toInt(x) + toInt(y);
    }
}

add(4)("foo"); // Exception!
```

Similar projects

- **Pit** (well documented, translates only to JavaScript, no IL)
- F# **Web Tools** (tries to solve "the heterogeneous nature of execution, the discontinuity between client and server parts of execution and the lack of type-checked execution on the client side", no IL)
- **Websharper** (professional web and mobile development framework, extensions for ExtJs, jQuery, Google Maps, WebGL and many more)

What we have

- Working translations from F# to intermediate language, IL, and from IL to both JavaScript and Python
- Formalization of core IL:
    - syntax,
    - operational semantics,
    - type system, and
    - soundness result.

Planned work

- Formal description of core `F#`, JavaScript and Python
- Formal description of translations: `F#` to IL, and IL to JavaScript and Python
- Proofs that translations preserve the operational semantics (and for `F#` to IL also the well-typing).