SML Toys — Compiling Standard ML to JavaScript

Martin Elsman

IT University of Copenhagen

Lunch Talk, Sep 17, 2007

Motivation

Type safe programming for the Web client:

- Program the Web with your favorite programming language...
- Can the use of "**advanced type tricks**", such as *phantom types* and *type indexed values*, provide safety guarantees to developers of AJAX applications?

Outline of the Talk:

- Features of SMLtoJs
- SMLtoJs in action — **a demo**
- The inner workings of SMLtoJs
- Applications
- Related projects

Features of SMLtoJs

Supports all Browsers:

- SMLtoJs compiles Standard ML programs to JavaScript for execution in all main Internet browsers.

Compiles all of Standard ML:

- SMLtoJs compiles **all of SML**, including higher-order functions, pattern matching, generative exceptions, and modules.

Library Support:

- SMLtoJs supports most of the Standard ML Basis Library, including:

  `Array2 ArraySlice Array Bool Byte Char CharArray CharArraySlice CharVector CharVectorSlice Date General Int Int31 Int32 IntInf LargeWord ListPair List Math Option OS.Path Pack32Big Pack32Little Random Real StringCvt String Substring Text Time Timer Vector VectorSlice Word Word31 Word32 Word8 Word8Array Word8ArraySlice Word8Vector Word8VectorSlice`

Features of SMLtoJs — continued

JavaScript Integration:

- ML code may **call** JavaScript functions and **execute** JavaScript statements.

DOM Access:

- SMLtoJs has support for **simple DOM access** and for **installing ML functions** as DOM event handlers and timer call back functions.

Optimizing Compilation:

- All ML module language constructs, including functors, functor applications, and signature constraints, are **eliminated** by SMLtoJs at compile time.
- Further optimizations include **function inlining** and **specialization** of higher-order recursive functions, such as map and foldl.
- As a result, SMLtoJs generates **fairly efficient** JavaScript code.

SMLtoJs in Action — a Demonstration

Example: Compiling the Fibonacci Function (`fib.sml`):

```
fun fib n = if n < 2 then 1 else fib(n-1) + fib(n-2)
val _ = print(Int.toString(fib 23))
```

Resulting JavaScript Code:

```
var fib$45 =
 function fib$45(n$48){
  if (n$48<2) { return 1; }
  else { return SmlPrims.chk_ovf_i32(
                  fib$45( SmlPrims.chk_ovf_i32(n$48-1) ) +     // Overflow
                  fib$45( SmlPrims.chk_ovf_i32(n$48-2) )       // checking
                ); };
 };
var __dummy =
 document.write(basis$Int32$.toString$447(fib$45(23)));        // Printing
```

Notice:

Compilation of an **sml-file** or an **mlb-file** (a project) results in an **html-file** mentioning a series of **js-scripts**.

Library for Manipulating the DOM and Element Events

```
signature JS = sig
  type doc type elem                                (* dom *)
  val document       : doc
  val getElementById : doc -> string -> elem option
  val value          : elem -> string
  val innerHTML      : elem -> string -> unit

  datatype eventType = onclick | onchange    (* events *)
  val installEventHandler : elem -> eventType
                            -> (unit->bool) -> unit
  type intervalId
  val setInterval   : int -> (unit->unit) -> intervalId
  val clearInterval : intervalId -> unit
end
```

Example: Temperature Conversion — temp.sml

```
val _ = print ("<html><body><h1>Temperature Conversion</h1>" ^
                "<table border='1'>" ^
                  "<tr><th align='left'>Temp in Celcius:</th>" ^
                      "<td><input type='text' id='tC'></td></tr>" ^
                  "<tr><th align='left'>Temp in Fahrenheit:</th>" ^
                      "<td><div id='tF'>?</div></td></tr>" ^
                "</table></body></html>")

fun get id = case Js.getElementById Js.document id of
      SOME e => e
    | NONE => raise Fail ("Missing id: " ^ id)

fun comp () =
 let val v = Js.value (get "tC")
     val res = case Int.fromString v of
                  NONE => "Err"
                | SOME i => Int.toString(9 * i div 5 + 32)
 in Js.innerHTML (get "tF") res; false
 end

val () = Js.installEventHandler (get "tC") Js.onchange comp
```

The Inner Workings of SMLtoJs

- SMLtoJs is based on the **MLKit** Standard ML compiler.
- SMLtoJs compiles Standard ML to JavaScript through an MLKit **intermediate expression language** called *LambdaExp*.
- **SML integers** are implemented as **JavaScript numbers** and operations check explicitly for overflow.
- **JavaScript variables** are used for representing **SML variables**.
- **JavaScript closures** are used for representing **SML closures**:

```
[[fn x => e]]_exp  ==> function(x) { [[e]]_stmt }
```

- **JavaScript functions** are used for **variable bindings**:

```
[[let val x = e in e']]_exp  ==> function(x) { [[e']]_exp } [[e]]_exp
```

- When compilation naturally results in a JavaScript statement *stmt*, the statement is converted into an expression:

```
        [[e]]_stmt ==> stmt
------------------------------------
[[e]]_exp ==> function() { stmt; }()
```

Compiler Issues

Possible Optimizations:

- **Unboxing** of certain datatypes, such as lists.
- Transform simple recursive functions into **while-loops**.
- Proper implementation of **tail-calls** using trampolines; none of the important JavaScript interpreters implements tail-calls efficiently.

Notice:

- Important JavaScript implementations of the future (such as Adobe's Tamarin JavaScript compiler) deals with tail-calls efficiently.

Towards a Type-Safe DOM API

Use combinators and phantom types for constructing XHTML:

- Use phantom types to ensure that **constructed XHTML is valid** according to XHTML 1.0 Strict DTD (Friis Larsen & Elsman 2004).
- Use phantom types to ensure that **generated element id's** are used at most once in DOM-updated XHTML (**linear use** of id's).
- Use phantom types to ensure that **updates to the DOM** results in a valid XHTML document.

Trick for Ensuring Linear Use of Id's

```
signature LIN = sig
  type 'c id
  val newId : unit -> 'c id

  type ('i,'o) x and 'a s and z
  val div   : 'o id -> ('i,'o) x -> ('i,'o s) x
  val &&    : ('i,'t) x * ('t, 'o) x -> ('i,'o) x
  val $     : string -> ('c,'c) x
  val print : (z,'o) x -> unit
end
```

- The LIN signature allows for only linear uses of identifiers.
- That is, the signature ensures that each div-element (containing an id) receives a unique type, which denotes the “position” of the div-element in the document tree.

Example using the LIN API

```
functor T (L : LIN) =
  struct
    open L infix && nonfix div
    val _ =
        let val id1 = newId()
            val id2 = newId()
        in print(div id1 ($"hello") && ($" world")
                     && div id2 ($" again"))
        end
  end
```

NOTICE:

- Changing the use of ***id2*** to ***id1*** results in a type error.
- By "storing" the type of the `div` element in the id-type, also updates to the DOM tree can be made type safe.

Towards Type-Safe AJAX Programming

- By integrating SMLtoJs with **SMLserver**, for server-side Web programming, a service API (a signature) may be implemented natively on the server and as a PROXY on the client.
  - Notice that the two implementations may make use of the **same signature** file, which facilitates **type-safety**.
- Several serialization possibilities:
  - XML.
  - JSON.
  - Low-bandwidth type-safe serialization using combinators.

Related Work

- The Google Web Toolkit project (GWT).
- The SCM2JS project by Loitsch and Serrano.
- The Links project. Wadler et al. 2006.
- The AFAX F# project by Syme and Petricek, 2007.
- Previous work on phantom types for embedded languages and type-indexed functions. Fluet & Pucella 2002; Leijen & Meijer 2000; Fridlender & Indrika 2000; McBride 2002; Yang 1998; Fiis Larsen & Elsman 2004.
- The Haskell WASH/CGI library by Thiemann. PADL'02, JFP'02, WFLP'02.
- The JWIG/Bigwig projects (Brabrand, Møller, and Schwartzbach).
- Type-safe language bindings for XML; e.g., XStatic, XDuce, XMLambda, UUXML.

More Information

Visit the SMLtoJs Home Page:

- http://www.itu.dk/people/mael/smltojs