MLton Cross Compiler Bootstrap Overview

Ánoq of the Sun, Hardcore Processing [*]

September 22, 2003

1 Introduction

This documents tries to give an overview of the bootstrapping process for turning MLton into a cross compiler.

2 Notation

The basic notation will be as the notation used in [?] and [?]. As a brief summary of this notation the function $[|\cdot|]_L$ will take the meaning of a program according to the semantics of the language L. For example the meaning of running the program *source* written in the language S with input $in_1, in_2, \ldots, in_n$ is the same as running an interpreter *int* for the language S written in the language L with the program *source* and it's input $in_1, in_2, \ldots, in_n$ as inputs for *int*. This is expressed with the following notation:

$$[|source|]_S[in_1, in_2, \ldots, in_n] = [|int|]_L[source, in_1, in_2, \ldots, in_n]$$

Compiling the program *source* with a compiler *comp* written in the langauge L is expressed as:

$$target = [|comp|]_L source$$

If *comp* performs a correct translation of programs from the language S to some target language T, the following should hold:

$$output = [|source|]_S[in_1, in_2, \ldots, in_n] = [|target|]_T[in_1, in_2, \ldots, in_n]$$

For simplicity we assume that there are no termination problems.

[*] ©1998-2001 Ánoq of the Sun (alias Johnny Andersen)

3 Compilation of MLton

3.1 Abbreviations

- $MLton_{SML}$: The Standard ML source code for MLton
- $MLton_{binLinux}$: A compiled version of MLton for Linux (assumed to be on x86)
- $Linux$: The Linux platform (also considered a "language")

3.2 Native Linux Compilation of MLton

Compiling MLton with itself once is usually done as follows:

$$MLton_{binLinux} = [|MLton_{binLinux}|]_{Linux}[MLton_{SML}, "linux", consts]$$

Where the input string "*linux*" can be considered a commandline argument for the platform to compile to and *consts* is the constants file that MLton uses duing compilation. Actually *consts* is usually generated by MLton - but we pretend that it has been generated already - since one of the purposes of this document is to find out how to generate the *consts* files for cross compilation.

3.3 Bootstrapping MLton into a Linux to Win32 Cross-compiler

To turn the compiler into a cross compiler we would need to do the following compilation step:

$$MLton_{binWin32Cross} = [|MLton_{binLinux}|]_{Linux}[MLton_{SML}, "win32bootstrap", consts_{Win32Bootstrap}]$$

Cross compiling applications with the resulting compiler is done as:

$$App_{binWin32} = [|MLton_{binWin32Cross}|]_{Linux}[App_{SML}, "win32cross", consts_{Win32Cross}]$$

And we hope to run this application on Win32:

$$output = [|App_{binWin32}|]_{Win32}[in_1, in_2, \ldots, in_n]$$

The interesting parts are how $MLton_{binWin32Cross}$, $App_{binWin32}$, $consts_{Win32Bootstrap}$ and $consts_{Win32Cross}$ are generated. The way that $consts_{Win32Bootstrap}$ and $consts_{Win32Cross}$ are generated can probably be controlled just by the flags "*win32bootstrap*" and "*win32cross*".

To get an overview of how $MLton_{binWin32Cross}$ is generated and how the $MLton_{SML}$ code should be modified to achieve that it is probably a good idea to pretend that the $MLton_{SML}$ code, the $consts_{Win32Bootstrap}$ file and the resuling binary $MLton_{binWin32Cross}$ is divided into 2 disjoint parts:

- The part of the code that implements reading and writing compiled files and other compiler output, input or other interfacing with the host environment (hostIO)

- The part of the code that generates code and basis library functions which is to be executed (outgen)

The constants file for the outgen part will have to be split further into 2 parts:

- The constants that are used to implement the outgen algorithms in the MLton code (impl)
- The constants that are used in the produced output (prod)

We will split the cross compiling equation into 2 equations, and to simplify the notation just a wee bit and focus on the parts we need to solve we will rename $MLton_{binLinux}$ to M_n (MLton native), $MLton_{SML}$ to S, $consts_{Win32Bootstrap}$ to c_b and $"win32bootstrap"$ to $"b"$:

$$MLton_{binWin32Cross_{hostIO}} = [|M_n|]_{Linux}[S_{hostIO}, "b", c_{b_{hostIO}}]$$

$$MLton_{binWin32Cross_{outgen}} = [|M_n|]_{Linux}[S_{outgen}, "b", c_{b_{outgen_{impl}}}, c_{b_{outgen_{prod}}}]$$

When we start to compile an application by running the entire $MLton_{binWin32Cross}$ on Linux each of the compiled parts will do their work in a different way:

- $MLton_{binWin32Cross_{hostIO}}$: Must produce files etc. as it is done on Linux. From this we can conclude that S_{hostIO} must be compiled for Linux and that $c_{Win32Bootstrap_{hostIO}}$ should be Linux constants.
- $MLton_{binWin32Cross_{outgen}}$: Must generate code and basis library calls for Win32. From this we can conclude that S_{outgen} must be compiled for Linux and generate code for Win32. This means that the constants $c_{b_{outgen_{impl}}}$ should be Linux constants and that $c_{b_{outgen_{prod}}}$ should be Win32 constants.

The file $consts_{Win32Cross}$ should be a constants file with only Win32 constants.

References

[1] Nils Andersen, Fritz Henglein, Niel D. Jones, *Notes for Dat2V-Programminglanguages at DIKU*, 1999.

[2] Carsten K. Gomard, Niel D. Jones, Peter Sestoft *Partial Evaluation and Automatic Program Generation*, Prentice Hall 1993.