Compiler bootstrapping and cross-compilation

Bootstrapping and cross-compilation are two classic and important concepts in compiler construction. Bootstrapping is the process of implementing a compiler in the language that it is supposed to compile. Alternatively, cross-compilation is the process where a compiler executes on one computer architecture and generates target programs to be executed on another computer architecture. This note compares and contrasts the two concepts.

Let us assume that a new computer architecture has been developed. Software has to be developed for this new computer architecture. However, before any other software program is written for this new computer architecture, language processors like assemblers and compilers have to be developed. The problem may be defined formally as follows. Let M and N represent an existing computer architecture and a new computer architecture respectively. Let HLL be a machine-independent high-level language. Also, let M.ML, N.AL and N.ML represent the machine language of M, assembly language of N and machine language of N respectively. The objective is to develop a compiler for HLL targeting N, i.e. a compiler that translates HLL into N.ML. This problem may be solved by either bootstrapping or cross-compilation. However, both approaches have their own merits and demerits.

The problem can be solved using bootstrapping ideally in two steps. In the first step, a compiler $C_{\text{N.AL}}^{\text{HLL N.ML}}$ to translate HLL into N.ML is implemented in N.AL. (Note that C_{Z}^{XY} represents a compiler that translates a source language X to a target language Y, and is implemented in a language Z.) We assume that an assembler for N has been already hand-coded in the machine language. This is a realistic assumption as implementing a compiler in a machine language will be quite forbidding. The compiler $C_{\text{N.AL}}^{\text{HLL N.ML}}$ will be good enough to produce target programs for N. However, the compiler will be difficult to modify or debug in future because it is implemented in an assembly language. So, in the second step, a compiler $C_{\text{HLL}}^{\text{HLL N.ML}}$ is implemented. Since the second compiler is implemented in HLL itself, it will have the desired properties of HLL. The two steps are represented in Figure 1 using T-diagrams[1].

An alternative approach is to use cross-compilation with M being the host computer architecture and N being the target computer architecture. An existing compiler $C_{\text{M.ML}}^{\text{HLL M.ML}}$ can be retargeted to *N* by modifying its back-end, i.e. the code optimization and code generation phases. The compiler $C_{\text{HLL}}^{\text{HLL N.ML}}$ thus obtained will execute on M but create programs that can be executed on N (Figure 2).

In the bootstrapping approach, the source language is the same as the language of implementation for the second compiler, which is called a self-hosting or self-compiling compiler. This allows the compiler developer to use the sophisticated features of the programing language to implement the compiler. Moreover, the compiler developer need not worry about the possibility of bugs in a compiler written by others. Bootstrapping is also a comprehensive test of the programing language. The bootstrapping approach makes the new computer architecture self-sufficient in the sense that no other computer architecture will be needed in future to develop software for the new computer architecture. However, any change in the definition of the source

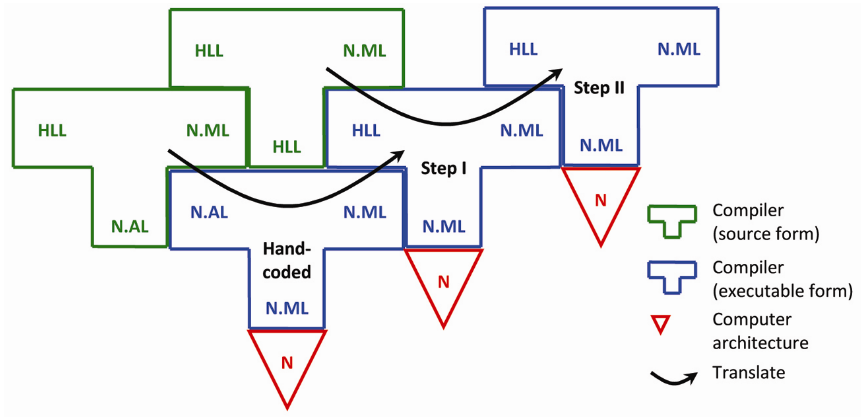

Figure 1. The bootstrapping approach. In step I, a compiler for HLL targeting N is written in the assembly language of N and assembled using a hand-coded assembler. Thus, a working compiler is obtained. In step II, the compiler is rewritten in HLL and compiled using the compiler obtained in step I. The result of step II is a self-hosting compiler. Note that a T represents an assembler or a compiler in either source or executable form. The labels on the left arm, right arm and foot of a T give the source language, target language and language of implementation of the assembler or the compiler respectively. An assembler or a compiler in its executable form can be used to translate the source form of a compiler to its executable form. A T representing the executable-form of a compiler has a triangle below it representing the computer architecture on which the compiler runs.

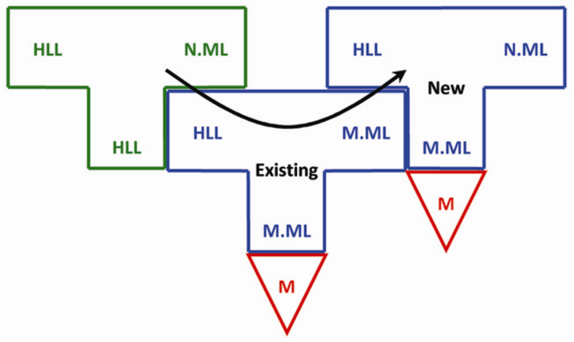

Figure 2. The cross-compilation approach. A compiler for HLL targeting N is written in HLL and compiled using the native compiler on M. A cross-compiler is thus obtained. The cross-compiler runs on M, but generates code for N. The label at the foot of a T representing the executable form of a compiler should be same as the machine language of the computer architecture represented by the triangle below it.

language will require modifications in the code of the self-hosting compiler. Another requirement is that the source language should be suitable and advanced enough for writing the compiler[2]. Bootstrapping may distort the design of a programing language that is not otherwise meant to implement compilers or similar programs. Additionally, the bootstrapping approach requires much time and effort, and is hence prescribed only for computer architectures which will be used for software development.

Alternatively, the cross-compilation approach requires less time and effort. An existing compiler can be retargeted to the new computer architecture by modifying its back-end. It is particularly suitable if the new computer architecture is a smartphone, an embedded device or any other battery-powered programable device. Such devices typically have severe processing, memory and power constraints. Using the cross-compilation approach, software for the device can be developed on another computer architecture and copied onto the device. This leads to a lifelong dependence of the device on the other computer architecture. However, it is not a major issue as such devices are not used for software development. A cross-compiler running on a personal computer and generating target code for a battery-powered programable device can employ a large range of code optimization techniques, while a native compiler running on such a device may at most afford to perform peephole optimization because of its various constraints.

It is interesting to note that the final compilers in both the approaches can be represented as $\mathrm{C}_{\mathrm{HLL}}^{\mathrm{HLL\ N.ML}}$ when in their source forms. However, in their executable forms, they become $\mathrm{C}_{\mathrm{N.ML}}^{\mathrm{HLL\ N.ML}}$ in the bootstrapping approach (Figure 1) and $\mathrm{C}_{\mathrm{M.ML}}^{\mathrm{HLL\ N.ML}}$ in the cross-compilation approach (Figure 2).

Although the concepts of bootstrapping and cross-compilation have been known for a long time, they are still in use. Efficient use of these concepts is often helpful in programing language design and compiler construction.

1. Bratman, H., *Commun. ACM*, 1961, **4**, 142.
2. Lecarme, O. and Peyrolle-Thomas, M.-C., *Software: Pract. Experience*, 1978, **8**, 149–170.

Received 25 March 2016; accepted 12 November 2016

TWINKLE GUPTA
SANYA YADAV
PINAKI CHAKRABORTY*

Division of Computer Engineering, Netaji Subhas Institute of Technology, New Delhi 110 078, India
**For correspondence.*
e-mail: pinaki_chakraborty_163@yahoo.com

Prominent precursory signatures observed in soil and water radon data at multi-parametric geophysical observatory, Ghuttu for M_w 7.8 Nepal earthquake

A devastating earthquake (*M* 7.8) occurred in the central part of the Nepal Himalaya on 25 April 2015 at 06:11:26.27 (UTC). USGS reported the epicentre location at 28.147°N and 84.708°E, and focal depth 15 km. The earthquake strongly hit Nepal causing over 7500 deaths and widespread destruction. A historical temple of 19th century was reduced to ruins within a few seconds. More than 55 causalities were reported in the adjoining parts of India, mainly to the south and east of Nepal. The earthquake was followed by 65 aftershocks within a period of three days after the main event. Among these the strongest aftershocks, i.e. *M* 6.7 occurred on 26 April at 07:09:08 (UTC) and *M* 6.6 occurred on 25 April at 06:45:20 (UTC). The moment tensor solution of the main shock suggests thrust fault mechanism with strike 293° and dip 7° (USGS GCMT solution). It caused unilateral rupture of $100 \times 80\ \mathrm{km}^2$ towards east and south from the hypocentre and a maximum slip of 5 m. The dislocation mainly occurred on the Main Himalayan Thrust (MHT), which is a low-angle northerly-dipping boundary between the Indian and Eurasian tectonic plates.

In this communication, we report observation of anomalous radon gas emission measured in a borehole at India's first multi-parametric geophysical observatory (MPGO) located at Ghuttu, Garhwal Himalaya, established by the Wadia Institute of Himalayan Geology (WIHG), Dehradun. MPGO is located in the central part of the seismic gap between the epicentre of the 1905 Kangra earthquake (*M* 7.8) and 1934 Bihar–Nepal earthquake (*M* 8.2) immediately to the south of the Main Central Thrust (MCT) within the High Himalayan Seismic Belt (HHSB). The recent Nepal earthquake occurred 636 km to the east of MPGO (Figure 1). The spatial extent of the so-called seismic gap is slightly reduced towards the west due to occurrence of the recent *M* 7.8 earthquake. The observatory is equipped with simultaneously operated multiple geophysical equipments that can measure radon, magnetic, gravity, seismic, GPS and water level data. The facility also has rain gauge, temperature (atmospheric and underground) and atmospheric pressure observations[1].

The linkage of radon emanation variation with earthquake mechanism reported in many previous studies has prompted the inclusion of radon variation as one of the parameters at MPGO, Ghuttu for earthquake precursory research. Radon is the disintegration product of radioactive uranium and thorium, which was observed for the first time as an earthquake precursor during the great Tashkent earthquake of 1966 (ref. 2). The radon emanation is likely to vary in the crust during earthquake preparation and occurrence period based on the well-accepted dilatancy–diffusion model of earthquake generation mechanism[3]. The model holds some promise for short-term earthquake prediction using radon measurement[4]. However, nonlinear dynamic behaviour