A Tutorial on Larch
and
LCL, A Larch/C Interface Language

John V. Guttag*and James J. Horning†

1 Introduction

The Larch family of languages is used to specify program interfaces in a two-tiered definitional style. Each Larch specification has components written in two languages: one that is designed for a specific programming language and another that is independent of any programming language. The former are the Larch interface languages, and the latter is the Larch Shared Language (LSL).

This tutorial material on Larch has three parts. Part 1, this part, is a short overview of the Larch approach to specification.

Part 2 contains material excerpted from a report describing version 2.3 of LSL. It introduces all the features of the language and briefly discusses how they are intended to be used.

Part 3 contains material aimed primarily at the C programmer who wishes to begin to integrate formal specifications into the program development cycle. It presents a specification language targeted specifically at C and discuss how it can be used to support a style of C programming in which abstraction plays a vital role.

*Address: MIT LCS, 545 Technology Square, Cambridge, MA, USA. Support for this work was provided by the Advanced Research Projects Agency of the Department of Defense, monitored by the Office of Naval Research under contract N00014-89-J-1988, and by the National Science Foundation under grant CCR-8910848.

†Address: DEC SRC, 130 Lytton Ave, Palo Alto, CA, USA
Larch, and especially LSL, has benefited from our study of other specification languages. However, since this is intended strictly as a tutorial for those who might wish to use Larch, we do not discuss these influences here. Instead we give a short Larch bibliography and refer the reader to discussions contained in those papers.

The material presented here is excerpted from two reports on Larch: *Report on the Larch Shared Language: Version 2.3*, by John V. Guttag, James J. Horning, and Andrés Modet, and *Introduction to LCL, A Larch/C Interface Language*, by John V. Guttag and James J. Horning. These reports can be obtained by writing to Reports Distribution, Digital Equipment Corporation Systems Research Center, 130 Lytton Avenue, Palo Alto, CA 94301-1044, USA, or by sending e-mail to src-report@src.dec.com.

2 A Short Overview of Larch

The most vexing problems in building systems concern overall system organization and the integration of components. Modularity is the key to controlling them, and specifications are essential for achieving program modularity. Abstraction boundaries make it possible to understand programs one component at a time. However, an abstraction is intangible. Without a precise description, there is no way to know what it really is, and it is easy to confuse an abstraction with one of its implementations.

Specifications can be written in natural languages, in semi-formal notations (with a restricted syntax but no formal semantics), or in truly formal notations. The potential advantages of formal specifications are that they have unambiguous meanings and are subject to manipulation by programs. The latter advantage can be fully realized only by using tools that support constructing and reasoning about them. The Larch Project is developing languages, tools, and techniques to aid in the productive application of formal specifications to software design, implementation, integration, and maintenance.

A Larch *interface specification* describes the interface that a program component provides to *clients* (programs that use it). Each interface specification is written in a programming-language-dependent *Larch interface language*. It relies on definitions from an *auxiliary specification*, written in a programming-language-independent specification language, the *Larch Shared*