Chapter 1

Introduction

This document is a tutorial introduction to functional programming, and, more precisely, to the usage of Caml Light. It has been used to teach Caml Light[1] in different universities and is intended for beginners. It contains numerous examples and exercises, and absolute beginners should read it while sitting in front of a Caml Light toplevel loop, testing examples and variations by themselves.

After generalities about functional programming, some features specific to Caml Light are described. ML type synthesis and a simple execution model are presented in a complete example of prototyping a subset of ML.

Part I (chapters 2–6) may be skipped by users familiar with ML. Users with experience in functional programming, but unfamiliar with the ML dialects may skip the very first chapters and start at chapter 6, learning the Caml Light syntax from the examples. Part I starts with some intuition about functions and types and gives an overview of ML and other functional languages (chapter 2). Chapter 3 outlines the interaction with the Caml Light toplevel loop and its basic objects. Basic types and some of their associated primitives are presented in chapter 4. Lists (chapter 5) and user-defined types (chapter 6) are structured data allowing for the representation of complex objects and their easy creation and destructuration.

While concepts presented in part I are common (under one form or another) to many functional languages, part B (chapters 7–11) is dedicated to features specific to Caml Light: mutable data structures (chapter 7), exception handling (chapter 8), input/output (chapter 9) and streams and parsers (chapter 10) show a more imperative side of the language. Standalone programs and separate compilation (chapter 11) allow for modular programming and the creation of standalone applications. Concise examples of Caml Light features are to be found in this part.

Part C (chapters 12–16) is meant for already experienced Caml Light users willing to know more about how the Caml Light compiler synthesizes the types of expression and how compilation and evaluation proceeds. Some knowledge about first-order unification is assumed. The presentation is rather informal, and is sometimes terse (specially in the chapter about type synthesis). We prototype a small and simple functional language (called ASL): we give the complete prototype implementation, from the ASL parser to the symbolic execution of code. Lexing and parsing of ASL programs are presented in chapter 12, providing realistic usages of streams and parsers. Chapter 13 presents an untyped call-by-value semantics of ASL programs through the definition of an ASL interpreter. The encoding of recursion in untyped ASL is presented in chapter 14, showing the

[1]The “Caml Strong” version of these notes is available as an INRIA technical report [24].

expressive power of the language. The type synthesis of functional programs is demonstrated in chapter 15, using destructive unification (on first-order terms representing types) as a central tool. Chapter 16 introduces the Categorical Abstract Machine: a simple execution model for call-by-value functional programs. Although the Caml Light execution model is different from the one presented here, an intuition about the simple compilation of functional languages can be found in this chapter.

Warning: The programs and remarks (especially contained in parts B and C) might not be valid in Caml Light versions different from 0.7.

Part I

Functional programming