AFR-0
The Amlantis System Overview

Version 0.11

Markéta Lisová

June 28, 2020

Special Thanks To

Sabina Eva Lisová

Marie Strnadová

Ondřej Profant

Vojtěch Biberle

Contents

Part I

Introduction

Chapter 1

A Brief Introduction to the Amlantis System

Amlantis System is a collection of specifications of programming and data definition languages and their related tools, runtimes and libraries.

Those specifications do not always require particular implementations, but attempt to avoid undefined behaviours as much as possible.

Amlantis also provides an open source reference implementation of those specifications, but it is indeed possible for other people to write their own implementations or forks[1] of this default implementation, probably focusing on optimising other aspects of the system, maybe exploring new options of future development to be pull-requested into the default implementation.

A Few Notes on the Name

Amlantis' name has quite some history. The project started being named *Coral*, but that collided with another language of a similar name, *CORAL 66*. Then it got renamed to *Gear*, but that again collided with another language of the same name, which seemed inactive at the time, *zippers/gear*. Than an idea was born and Aml was named *Amlantis*, which is whatever you want it to be. It could be a misspelling of *Atlantis*[2], it could be an acronym like *A ML Language*, or maybe even something like *A ML Language And Neat Technology Improvement System*, or maybe *Caml* without the *C*. For the meaning of the cryptic *ML* part, search for the *Standard ML* or *OCaml*.

[1]Forks and pull requests are preferred way of help!

[2]Intentionally – because otherwise, it would be named Atlantis, but there is already a city of that name.

1.1 The Amlantis Languages

The Amlantis System is a home to more than only one programming language – in fact, it can be home to any number of languages, ones that Amlantis users can either extend from existing languages create or even create new ones entirely. Every language build within the Amlantis System shares a common type system and value models, and thus are interoperable.

The minimal Amlantis System contains these languages:

- `Aml/Base`, a minimal Lisp-like[3] language that is easy to parse and limited in functionality.
- `Aml/Core`, an ML-like[4] language that is easier to read and harder to parse, fully featured in functionality.
- `Aml`, an ML-like[5] language that extends `Aml/Core`, is easier to read and the hardest one to parse, also fully featured in functionality, and introducing more grammar than its parent `Aml/Core`.

More languages may be added to this list over time.

[3]One might say that `Aml/Base` is a grandchild of Racket and Clojure.

[4]One might say that `Aml/Core` is a grandchild of OCaml, SML and `F#`.

[5]One might say that `Aml` is a grandchild of `Aml/Core`, Haskell and Ada.

Part II

The Amlantis System Environment

Chapter 2

Readers

Amlantis System provides multiple interfaces to talk to it with. Those interfaces would be:

- A REPL console, where user's text input is immediately read, evaluated, printed and requested again.
- Source code, where user's text input is stored for reuse in general, usually to either do some scripting, or building modules and complex applications.
- Bytecode or native (machine) code files, which originate from textual source code.

More ways to interact with the Amlantis System might be possible in the future.[1]

Either way, for now, text-based interface to all Amlantis System tools require it to have a component that is able to read the text. We call them *Readers*.

Every Reader needs a source port to read from. Such a port could be console input for the REPL, a file on local filesystem, a TCP connection, socket, or just an in-memory string[2]. Reading program source starts with a top-most Reader, which we may call the *Root Reader*. Such Reader does not know about which language it is reading (whereas recursively-descendent created Readers may inherit such knowledge), so we say it is reading in a *language-less mode*.

The Root Reader's `read-program` method is used on the top-most initial port.[3]

[1]Some ideas for that would include natural speech interface (at least in English).

[2]More about strings later.

[3]Might be different in future, when we allow it to use `read` instead. That is not very useful now though.

2.1 Language-less Root Reader Abilities

The language-less Root Reader can recognise basic delimiters, such as all whitespace and newlines.

Delimited by these, it can recognise the following forms:

- Shebang, such as `#!/usr/bin/env aml`, to point to the entry-level Amlantis System tool.
- Reader switch, `#reader` *name* `...`, which specifies a reader to use for the following forms, and switches to it. The *name* should be a simple identifier.[4]
- Language switch, `#lang` *name* `...`, which expands right away into `#reader` *name*`.Lang.Reader ...`.[5] The *name* should be a simple identifier.[6]

For now, that's it. In the next iteration, the list may include line comments or alias for `#lang` in order to support languages such as R^{6}RS.

The language-less Root Reader should not be accessible from user's programs directly, hence we define no system name for it.

[4]Specification for identifiers will be provided later.
[5]It may expand to other forms as well in certain order, due to be defined later.
[6]Specification for identifiers will be provided later.

Part III

The Amlantis System Tools

Chapter 3

The Top-Level System or REPL – `aml`

Chapter 4

The Compiler – `amlc`

In this chapter, we're going to describe the Amlantis System Compiler `amlc`, which compiles source code files to bytecode files. The compilation process involves reading through the input source files, expanding them by evaluating the read programs by using the specified language, and serializing the type system's state along with modules' state to bytecode files, which are then to be run by `amlrun` (§5) or `aml` (§3).

Chapter 5

The Runtime System – `amlrun`

Chapter 6

The Debugger – `amldebug`

Chapter 7

The Profiler – `amlprof`

Chapter 8

The Builder – `amlbuild`

Chapter 9

The Book Environment – **`amlbook`**