μLarvaScript: Rethinking the Larva Scripting Language

Adrian Francalanza
adrian.francalanza@um.edu.mt

University of Malta

1 Synopsis

polyLarva, the latest incarnation of the Larva runtime-verification (RV) tool suite, experienced a major redesign to its scripting language (used for specifying the monitors that carry out the RV.) As opposed to previous versions, where the programmer specified the monitoring *automata* describing the RV procedure, in pLarvaScript (the scripting language for polyLarva) the programmer specifies monitoring as a *sequence of (guarded) rules* of the form

$$p, c \longrightarrow a$$

where p denotes an event pattern, c denotes a condition and a stands for a monitor action. A monitor sythesised from this sequence of rules would then listen to a stream of events: for each event e, it attempts to match an event pattern of the list of rules; when this happens, the monitor evaluates the corresponding condition of the rule and, if successful, the respective rule action is triggered.

By and large, the new version of LarvaScript has so far been developed organically, motivated more by the pragmatic concerns of the applications considered rather than by language design issues. Also, because of backward-compatibility concerns, a number of design decisions were inherited, carried over from its percursors. Although effective and pragmatic, this method of development has hampered a full understanding of the resulting language: at present, the only formal semantics available is the polyLarva compiler itself which is not ideal for a number of reasons (*i*) an understanding of the langauge constructs requires a detailed understanding of the compiler; (*ii*) we have no way how to determine whether the compiler implementation is correct; and (*iii*) concurrency often introduces subtle semantic and language design issues that are best studied at a higher level of abstraction.

In this talk, I will discuss preliminary investigations regarding the analysis of the pLarvaScript language from a foundational perspective. For instance, we consider inherited constructs such as `foreach` for defining sub-monitors together with associated design decisions, such as the hierarchic organisation of these submonitors, and reasses them again from first principles. We do not not have any conclusive results, and thus far our guiding principles have been simplicity, elegance (admittedly both subjective measures) and a potential concurrent implemenation of the tool. I will therefore be seeking feedback from the audience

during the talk, which should help us hone our present positions regarding the understanding of the language.