# Prototype A1

Alkis Gotovos

April 29, 2010

# 1 Introduction

The aim of this document is to describe the *structure* and *features* of a fully functional prototype for the Erlang concurrency testing tool that we intend to develop. Our rationale is to use this prototype (PRA1) to gain insight into the inner workings of our main project and detect potential pitfalls as early as possible.

In the following sections we present the subset of functionality that we shall implement in PRA1, as well as an overview of its architecture (which possibly forms a structural preview for our main project).

# 2 Functionality

Being the very first prototype, PRA1 shall only implement a core subset of the overall functionality. Specifically, we shall consider the following Erlang "constructs" for now:

- The **`spawn`**`/1` BIF
- The `send/2` BIF (equivalently the **`!`** operator)
- The simple **`receive`** expression (not containing an **`after`** clause)

The current implementation shall accept a single erlang module for analysis and output a log, containing all possible process interleavings with respect to the aforementioned "constructs".

# 3 Instrumentation

To be able to analyze a module we have to use wrapper functions and statements in place of the standard Erlang ones. The easiest way to do this (without having to mess with the Erlang runtime system) is to preprocess the module source before handing it over to the scheduler for analysis. This preprocessing step is called *instrumentation*.

Next, we present the code transformations done by the instrumenter for the three constructs mentioned above (functions with a `'rep_'` prefix are custom wrapper functions).

- The statement

  **`spawn`**`(`*`Function`*`)`

  will become

  `sched:rep_spawn(`*`Function`*`)`

- The statement

  *`Process`* **`!`** *`Message`*

  will become

  `sched:rep_send(`*`Process`*`, `*`Message`*`)`

- Finally, the statement

```
receive
    Pattern1 [when GuardSeq1] -> Body1;
...;
    PatternN [when GuardSeqN] -> BodyN
end
```

  will become

```
sched:rep_receive(
    fun(Aux) -> receive
                    Pattern1 [when GuardSeq1] ->
                      {Pattern1, Body1};
                ...;
                    PatternN [when GuardSeqN] ->
                      {PatternN, BodyN}
                after 0 -> Aux()
                end
    end)
```

  For the above to work, patterns containing the anonymous variable (_) will have to be replaced with equivalent patterns containing only named variables.

# 4 Structure

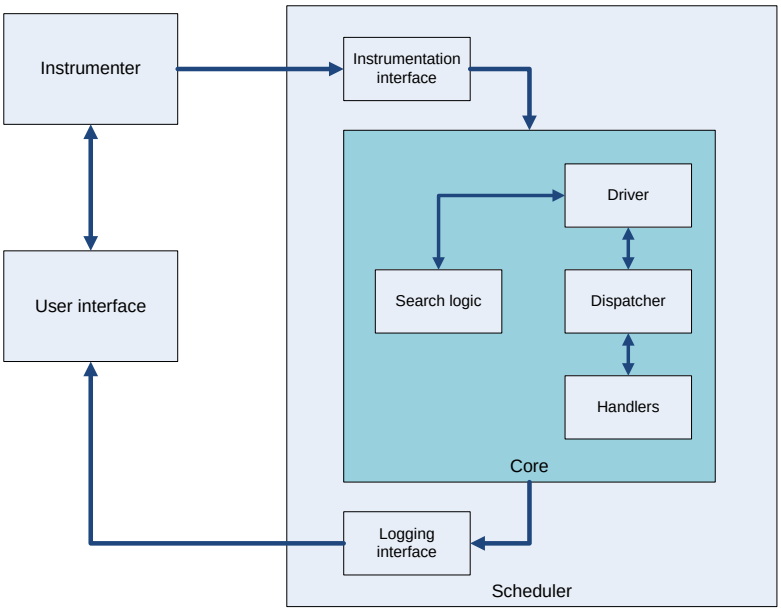


Figure 1: Architecture overview