The pepaprobe Manual

Allan Clark

June 18, 2007

1 Introduction

PEPA[Hil96] is a popular stochastic process algebra. The `pepaprobe` utility takes as input a PEPA model and a number of probes descriptions. These probes are performance measurement specification probes in the style of [AKBD04] and describe, using a high-level regular-expression like language, additional components to be combined with the input model. Such probes are intended to be observational and therefore do not change the original behaviour of the given model. The output is the input model transformed according to all of the given probes.

The probe descriptions are translated into pepa components which then synchronise with the model (or a part of it). Information about the state at a given time t can be derived by interogating the state of the probe at time t.

2 Basic Usage

Probe descriptions are given via the `--probe` option. Section 3 describes the grammar in more detail, here a few examples are given.

Labels may be attached to a probe to signify an immediate communication action of some important event. There are two specially regarded labels *start* and *stop*. These labels indicate the entering and exiting of the state of interest. Either the start and end of a passage if a passage-time query is subsequently given to the output model or the entering and exiting of the state of interest in a steady-state computation.

The following probe will 'start' on any of the actions a, b or c and, following that will 'stop' on any of the actions x, y or z. Notice that all probes are cyclic and loop back to the start once they have completed.

$$(a \mid b \mid c) : start, (x \mid y \mid z) : stop$$

The following probe waits until it has seen three a actions without observing a b action. Once this occurs the probe is 'started', to stop the probe it must observe the sequence c, d.

$$(a, a, a)/b : start, (c, d) : stop$$

Probes may be attached to a particular component within the model using the double colon syntax. The following probe may be use to ask what is the

probe	:=	$location :: R$	A local probe
	|	R	A global probe
location	:=	$processId$	Attach to a single process
	|	$processId[n]$	Attach to an array of processes
R	:=	$action$	Observe an action
	|	$R : label$	Send a signal on matching R
	|	R_1, R_2	R_1 followed by R_2
	|	$R_1 \mid R_2$	R_1 or R_2
	|	$R*$	zero or more R
	|	$R+$	one or more R
	|	$R\{n\}$	n R sequences
	|	$R\{m, n\}$	between m and n R sequences
	|	$R?$	one or zero R
	|	R/a	R without observing an a

Figure 1: The grammar for probe specification in pepaprobe.

probability that the *Client* component has performed three or more requests and has yet to see any responses.

$$Client :: (request, request, request)/reponse : start, response : stop$$

3 Probe Grammar

The full grammar for probe specification is given in Figure 1

4 Inner workings

To compile a probe specification the specification is first transformed into a non-deterministic finite state machine.

This is then translated into a deterministic finite state machine.

This is then minimised, by combining equivalent states. Minimising the dfa is not quite the same as a traditional dfa minimiser. Two states are considered equal they have equal move sets. That is, two states s1 and s2 are considered equal if every action which can be taken by s1 can be taken by s2 and results in the same state and vice versa. For the purposes of the comparison any move that results in state s1 or s2 is considered the same (provided it has the same label). So for example if s1 loops on action 'a' that is equivalent to s2 looping on 'a' or performing 'a' and going to state s1. In fact if s1 performs 'a' and goes to s2, and s2 performs 'a' and goes to s1 then the two moves are considered equal.

If two states are considered equal (because they have equal move sets) we delete one of the states, remove all moves which originate from that state, and change all moves which target that state to target the equivalent state. We then recursively minimise the dfa as this coalascing of states may lead to further comparable states.

As an example consider the probe:

$$(a, b)?, c : start, d : stop$$

The following Figures: 2 3 4 and 5 denote the sequence of transformations done in pepaprobe. Figure 2 shows the probe as a non-deterministic finite automata. Figure 3 shows that nfa without the non-determinism. Figure 4 shows the same dfa minimised and finally in figure 5 the dfa has the self-loops added such that the probe may perform any action in every state.

5 Start, Stop and `--no-master`

6 Outputting a dot file

The finite state machine produced from a probe specification may be output in 'dot' file format for use with the graphviz program. This is done by citing the `--output-dot` option.

7 Controlling nfa operations

The options in this section are mostly for those interested in the inner workings of pepaprobe. There are four options for controlling how the finite state machines are manipulated. Please note that if you are using pepaprobe to generate a pepa model to be analysed you almost certainly do not want to give any of these options.

`--no-minimise` Do not minimise the deterministic finite automata

`--min-self-loops` Add the self-loops before performing minimisation

`--no-min-self-loops` Perform the minimisation before adding the self-loops. This means the self-loops do not contribute to the minimisation.

`--no-self-loops` Do not add the self-loops (useful mostly for debugging)

`--non-deterministic` Leave the finite automata non-deterministic (again useful mostly for debugging)

The graphics in the section 4 were all produced using the `--output-dot` and the options given above. If you wish to just see the diagram of probe then a useful command to give is something such as

```
pepaprobe --output-dot <fa options> PEPAFILE | \
   dot -Tpng -o <filename>.png && \
   eog <filename.png>
```

Note that some of these options cause others to have no effect. For example if the option `--no-self-loops` is given then the options `min-self-loops` and `no-min-self-loops` will have no effect. Similarly for the `no-minimise` option.

References

[AKBD04] A. Argent-Katwala, J.T. Bradley, and N.J. Dingle. Expressing performance requirements using regular expressions to specify stochastic probes over process algebra models. In *Proceedings of the Fourth International Workshop on Software and Performance*, pages 49–58, Redwood Shores, California, USA, January 2004. ACM Press.

[Hil96] J. Hillston. *A Compositional Approach to Performance Modelling.* Cambridge University Press, 1996.

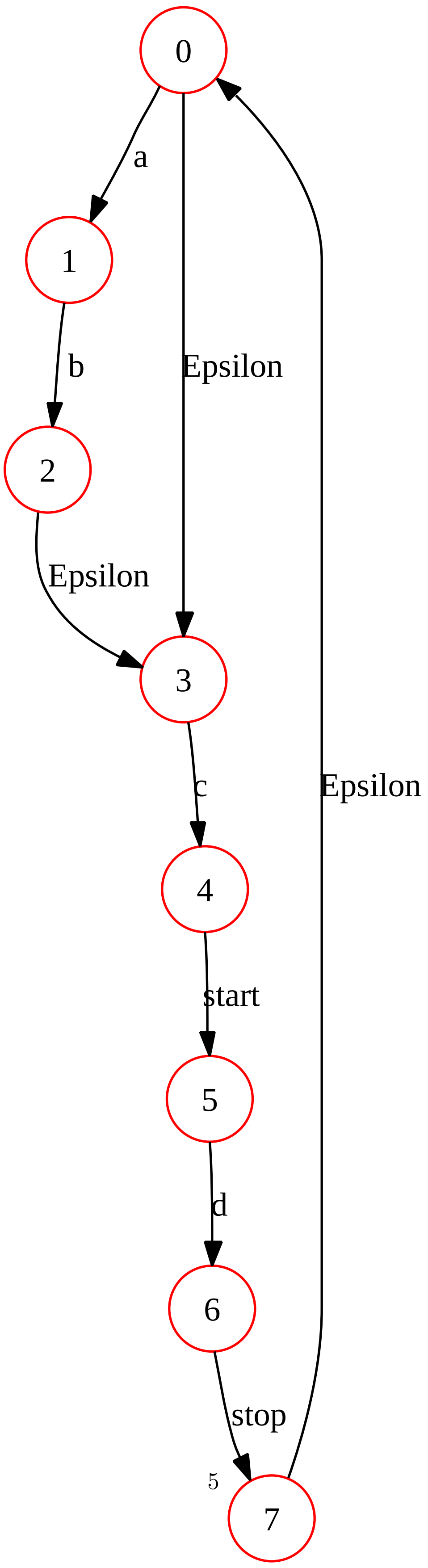

Figure 2: The probe translated into a non-deterministic finite automata

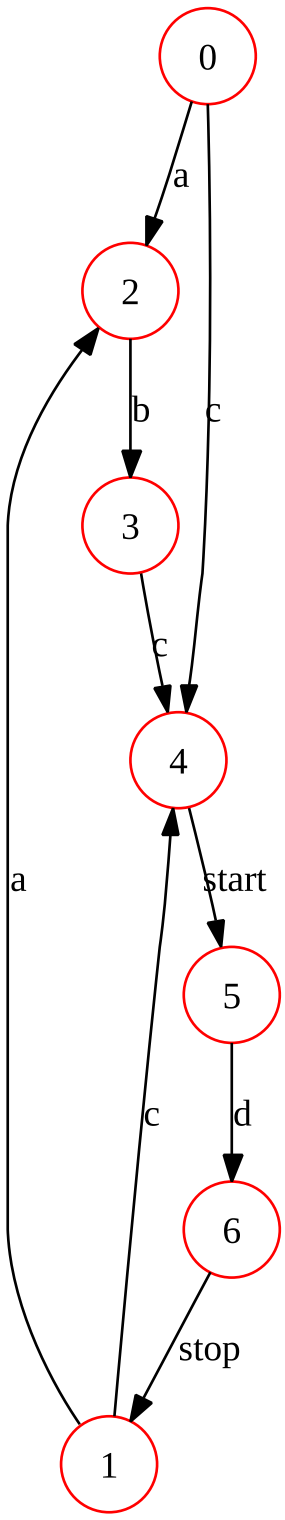

Figure 3: The nfa translated into a deterministic finite automata

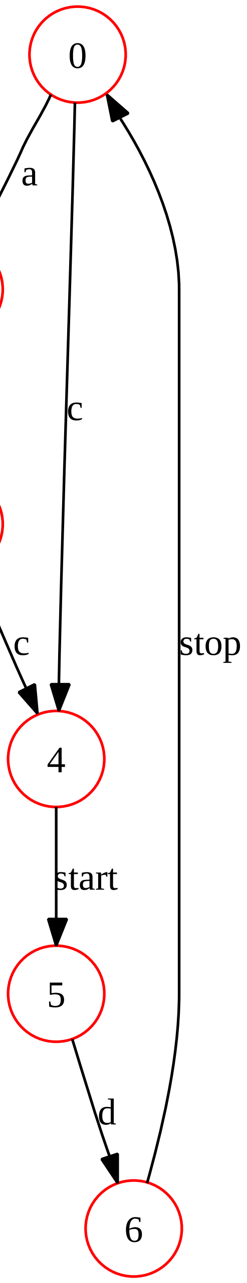

Figure 4: The dfa minimised by removing equivalent states.

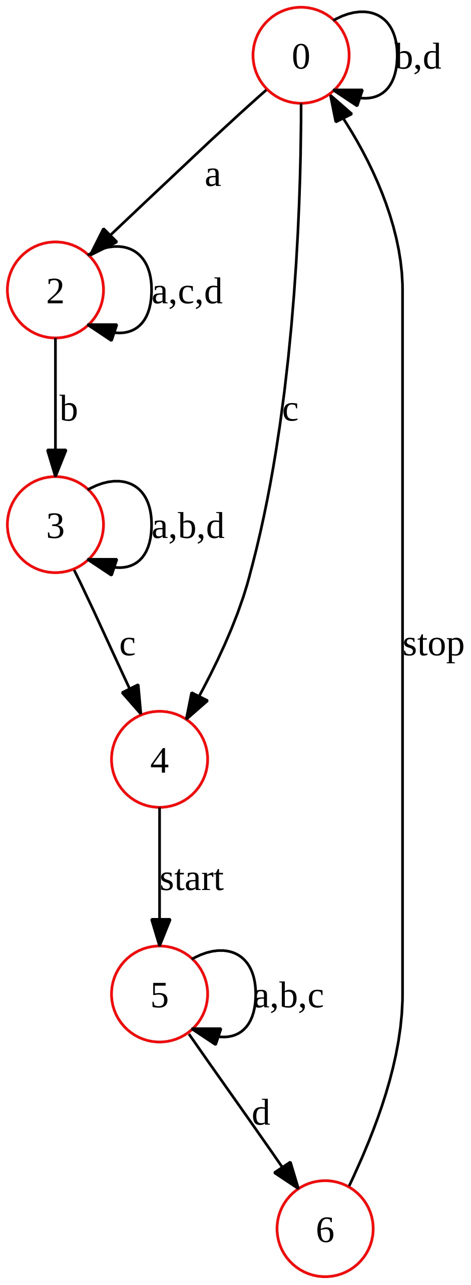

Figure 5: The final probe achieved by adding the self-loops