Epoch QuickStart guide

Guillaume Infantes

January 21, 2013

This guide describes usage of EPOCH temporal probabilistic assessment tool as of version 20130121.

1 Preliminaries

Epoch aims at ensuring the *validity* of *temporal properties* from an *initial state* of a given *dynamic model*.

- The *validity* is simply a true/false value output as a 1 or 0.
- The *temporal property* is for now only a small subset of PTCL* (more on this below).
- The *dynamic model* of the system is given as an AltaRica file, that can be directly exported from modelling/assessment tools like OCAS or others.
- The *initial state* may be specified directly in the dynamic model itself ("init" keyword of AltaRica files, can be given from the tool used to build the model itself); or it can be specified as a separate file (see section 2.2).

For version 20130121, time can have two very different semantics:

- First is an "abstract" notion meaning a timestep, that is any change in the system. It is denoted *discrete* time.
- Second is the natural time, meaning a real time in seconds/hours. It is denoted *continuous* time.

Temporal properties

A temporal property is generally a property on possible execution paths of the dynamic model (see technical report DCSD-T/R-120/11 for a general presentation). Epoch currently handles only simples constructs that should be sufficient for DIANA needs. We call a *query* the asking for a check of a temporal property.

An example of query is: "is the probability that variable FC_x takes the value true whithin 3 time units greater than 10e-9 ?".

- For discrete version the time unit is a timestep, i.e. a (non-immediate) change in the system. This means that if the model is a failure propagation model, then 3 timesteps means 3 failure events. Immediate events are denoted "dirac(0)" and are not taken into account for the horizon, as they

are used for ease of modelling in order to make the tool compute the state of the system for non-trivial updates. For instance, in the Bleed model, many dirac(0) updates are necessary to compute the resulting pressures after any event, while these intermediate states do not have any physical reality.

- For continuous time, the semantic of time is the intuitive one. The unit is not specified, it just has to be consistent over the model and the queries.

Formally, the previous queries can be written as:

$$?P_{\leq 3}(true\ Until\ FC\_x = true) \geq 10^{-9}$$

We will see in next section how to specify such query within EPOCH.

2 Use

`--help` gives description of command-line options.
`--verbosity` changes text output information level.

2.1 Model file

`--file` gives the name of the dynamic model (including initial state) to check. As files are not searched in any particular directory (except current directory), we strongly encourage to give full qualified names.
Example: `--file $HOME/model.alt`

2.2 Init file

`--init` allows to give path to a xml file specifying initial values. If initial values are specified both in model and init file, init file values are used. An example is show below:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<init
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="./init.xsd">
<node>
  <name>Rudder_BPS_RB</name>
  <type>class</type>
  <laws>
    <event name="loss" law="exp" param="1E-5"/>
  </laws>
  <values>
    <value var="Status" val="hs"/>
  </values>
</node>
<node>
  <name>main.BCM</name>
  <laws>
    <event name="loss" law="exp" param="2E-5"/>
  </laws>
  <values>
    <value var="Status" val="hs"/>
  </values>
</node>
</init>
```

The user has to specify the nodes where init values has to be set, using the `<node>` tag. Inside a node, several fields have to be specified:

- `<name>node_name</name>`: specifies the name of the node. It can be:
  - an instance, and so the full name has to be given, as for `main.BCM` (in this case `<type>instance</type>` is assumed as default)
  - a node definition (will affect all its instances). In this case, the `<type>class</type>` has to be added (`Rudder_BPS_RB` example).
- `<laws>` allows to change event law, specified by a list of tags with the syntax `<event name="ename" law="elaw" param="val"/>`; where `ename` is the event name inside the node, `law` can be `dirac` or `exp`, and `val` can be `0` (for diracs) or any double (for exponential law rates). Doubles can be stated as `1.5E-5` for example.
- `<values>` start a list of initial values stated with the following general syntax: `<value var="varName" val="varValue"/>`, where `varName` is the name of the variable (a string) and `varValue` its initial value (also a string).

The complete XSD definition is given at the end of the document.

2.3 Query specification

As the query is quite complicated, several switches are mandatory for query specification. In the following , we will use $?P_{\leq 3}(true\ Until\ FC\_x = true) \geq 10^{-9}$ as an example.

Variable

`--var` gives the name of the variable to check. The variable name has to be the full qualified *case-senstive* AltaRica name of the variable instance.
Example: `--var main.LGERS.SAFETY.fc_safety^FC15` (from A380 LGERS model)

Value

`--value` value of the variable. For string/enum values, the argument is *case-sensitive.*
Examples: "`--value true`" "`--value OK`"

Time horizon

`--dhor` specify time horizon for discrete time property checking.
Example: `--dhor 3`
`--chor` specify time horizon for continuous time property checking.
Example: `--chor 3`

Probability threshold

`--thres` threshold for probability.
Example: `--thres 10e-9`

Negation of the property

`--neg` specifies $\neq$ instead of $=$ for the inner property. For example, if FC_x can be "false", "true" or "dont_know", and both "true" and "dont_know" are problematic, then the property to check is FC_x $\neq$ false and one has to call EPOCH with:
`--var FC_x --value false --neg`

2.4 Algorithm selection

Several algorithms can be used by EPOCH. They are selected with the `--algo` switch.
`-1` runs all algorithms one after the other (usefull only for testing purposes).

2.4.1 Discrete time

- `0` runs exact computation over infinite time (do not honor `--dhor` switch). Not usefull for DIANA.
- `1` runs iterative computation over infinite time (do not honor `--dhor` switch). Not usefull for DIANA.
- `2` runs exact computation over infinite time using bound mechanism (do not honor `--dhor` switch). Not usefull for DIANA.
- `3` runs iterative computation over infinite time using bound mechanism (do not honor `--dhor` switch). Not usefull for DIANA.
- `4` runs exact computation over finite time specified by `--dhor` switch.
- `5` (default) runs exact computation over finite time specified by `--dhor` switch, using the bound mechanism.

Algorithm 5 is designed to be the most efficient and should be used for DIANA for discrete time computations.

2.4.2 Continuous time

- `6` runs exact computation without bounding mechanism.
- `7` runs exact computation with bounding mechanism.

These algorithms comes with different implementations for internal Hessenberg matrices, that can be changed using the `--hess` switch.

- `0` triangular Hessenberg matrices
- `1` banded Hessenberg matrices
- `2` dense Hessenberg matrices

Default value is `0`, this aims to be the most efficient version.

3 Output

3.1 Textual output

Currently, default output gives the following informations:

- `f1: true`

  Generally, we check $f1\ Until\ f2$ formulas, but for now f1 is always true;

- `f2: P`

  where P is the property to check (defined with `--var`, `--value`, `--neg`);

- then some data about the model are given:

  ```
  MODEL STATS:
  number of state vars: s
  number of _different_ flow vars: f
  number of atomic events: e
  ```

  s is the total number of states variables in the fully instanciated model; f is the number of different flow variables (for assertions of type $v1 = v2$, only one instance is considered); e is the number of events (generally failures), including dirac(0) but not taking into account synchronizations.

- `Check [recall of user selected algorithm] (Pr[f1Uf2](0,h) >= t):`

  where h is the horizon given by `--hor` and t is the probability threshold given by `--thres`.
  For continuous time algorithms, the probability at specified time is given once here: `PROB: xx.xxx`
  The following 0 or 1 is the result for the query; 0 for false, 1 for true. *For scripting purposes, this value is also given to the shell as an exit value* `$?`.

- `time taken: n seconds`

  n is the time taken by the execution (if alone);

- `probability: [either a value for unbounded versions or something like:] [ {([0,10]->0)} ; {([0,10]->0)} ]`

  gives the probabilities as intervals. The format for bounded version is [l,u], where l is the lower bound, u the upper, and l,u are given as interval/value sequences, $\{(i_1)(i_2)\ldots\}$, whith i_i of the form `[a,b] -> p`, meaning that from timestep a to timestep b, the probability of the property is p.

- `explored states: n`

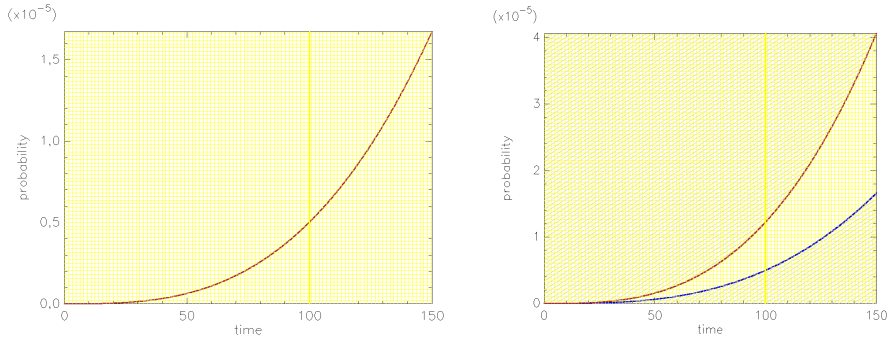
  here n is the total number of states that EPOCH had to instanciate in order to answer the query.

3.2 Graphical output for continuous time algorithms

For continuous time algorithms, a graphical view of the analytical solution is displayed, as shown on figure 1.

In these figures, x-axis is time, y-axis is probability, vertical bold yellow line shows time horizon specified by the query.

In unbounded version (figure 1(a)), the red line show the evolution of the probability of the specified property with time. In bounded version, as in example of figure 1(b), the red and blue curves show the enveloppe of the true probability which is sufficient to answer the query.



(a) no bounds (b) bounds

Figure 1: Output examples for continuous time algorithms

4 Note on queries

Due to the bound mechanism used by the "bounded" algorithms, it can be interesting to ask for the complementary query, depending on the model. For instance, $?P_{\leq 3}(true\ Until\ FC\_x \neq true) \leq 1 - 10^{-9}$ gives the same information as $?P_{\leq 3}(true\ Until\ FC\_x = true) \leq 10^{-9}$. Depending on the model, these two queries may need very different computation times.

XSD definition of init files

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xs:complexType name="initvalue">
    <xs:attribute name="var" type="xs:string" use="required"/>
    <xs:attribute name="val" type="xs:string" use="required"/>
  </xs:complexType>

  <xs:simpleType name="lawType">
    <xs:restriction base="xs:string">
      <xs:enumeration value="dirac"/>
      <xs:enumeration value="exp"/>
    </xs:restriction>
  </xs:simpleType>

  <xs:complexType name="initlaw">
      <xs:attribute name="name" type="xs:string" use="required"/>
      <xs:attribute name="law" type="lawType"/>
      <xs:attribute name="param" type="xs:double" default="0"/>
  </xs:complexType>

  <xs:simpleType name="nodeType">
    <xs:restriction base="xs:string">
      <xs:enumeration value="instance"/>
      <xs:enumeration value="class"/>
    </xs:restriction>
  </xs:simpleType>

  <xs:complexType name="initvalues">
    <xs:sequence>
      <xs:element name="value" type="initvalue" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>

  <xs:complexType name="initlaws">
    <xs:sequence>
      <xs:element name="event" type="initlaw" minOccurs="0" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>

<xs:complexType name="initnode">
  <xs:all>
    <xs:element  name="type" type="nodeType" default="instance" minOccurs="0"/>
    <xs:element name="name" type="xs:string"/>
    <xs:element name="values" type="initvalues" minOccurs="0"/>
    <xs:element name="laws" type="initlaws" minOccurs="0"/>
  </xs:all>
</xs:complexType>

<xs:element name="init">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="node" type="initnode" maxOccurs="unbounded"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

</xs:schema>
```