HPA Verification Tool Manual

Yue (Cindy) Ben*

May 1, 2015

The HPA Verification Tool is developed to present our research on Model Checking Failure Prone Open Systems Using Probabilistic Automata. Systems are modeled as Hierarchical Probabilistic Automata (HPA), and verified against our Forward and Backward algorithms.

Project Website: https://github.com/benyue/HPA/

1 Installation

There are several options to install the HPA Demo tool.

Download JAR	Download ”HPA.jar” file, which is the single executable file of the HPA demo tool. Then double-click the JAR file to execute it.
Download JNLP	Download ”HPA.jnlp” file, which will connect to Internet and download the latest JAR file.
Online Access	To run the tool from web browser, please click ”Launch” on the following webpage (IE browser recommended): http://www.cs.uic.edu/ yben/tools/HPA/dist/launch.html

Make sure you have Java Runtime Environment (JRE) installed, with a recommended version of 1.7.0 or higher.

Bypass Java security blocking. For all operating systems, the JAVA security settings may probably block the Jar program from running at the first time, since it requires access to your local files to load and save HPA files. Please adjust the Java security settings accordingly[1] by adding ”http://www.cs.uic.edu” in the exception website list.

Bypass Mac OS security blocking. Mac OS wouldn't allow this third-party app (the executable file, either the JAR file or JNLP file) to run at the first time. Please bypass the security settings[2] for the app: 1) In Finder, Control-click or right click the icon of

[0]University of Illinois at Chicago. Email: yben2@uic.edu

[1]Reference: http://java.com/en/download/help/jcp_security.xml

[2]For mode details, please search ”How to open an app from a unidentified developer and exempt it from Gatekeeper” on webpage https://support.apple.com/en-us/HT202491

the app. 2) Select Open from the top of contextual menu that appears. 3) Click Open in the dialog box. If prompted, enter an administrator name and password.

Also, because of this Mac security setting, on-line direct access to the app will be blocked. Please DOWNLOAD the executable file (.JAR, or .JNLP) instead to run.

2 Manual

There are mainly 2 parts in the tool: One interacts with user to create a 1-HPA; The other is for verification - decidability and robustness problem. To get started, please read the manual below or simply follow instructions and tips by hanging your mouse over labels, buttons, text fields, etc. in the tool. Examples can be found under the folder "examples" on the project website. Or you can write your own PA and HPAfiles following corresponding file formates introduced later in this manual.

2.1 Verification

The Verification part of the tool implements the HPA-based verification algorithms. It loads an HPA file, does simply validity checking and leveling, and executes different verification algorithms.

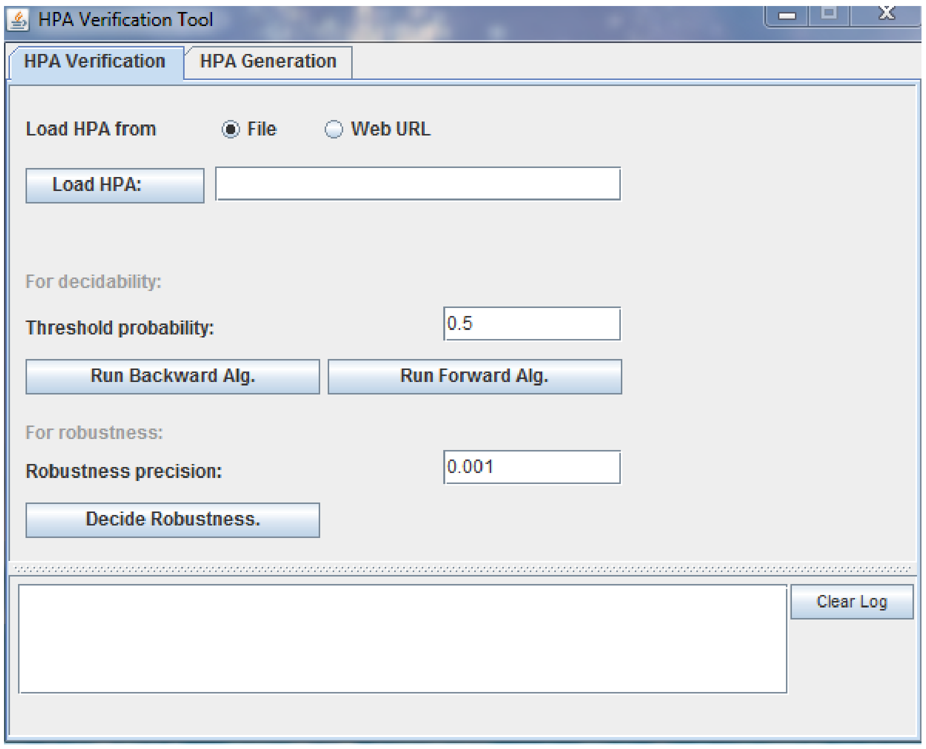


Figure 1: HPA Tool User Interface for Verification

2.1.1 Loading HPA

Current the tool supports two ways to load HPA:
(1) Default HPA files in the format of plain text file, .txt, .hpa files (please refer to Section 2.1.2).
(2) PRISM[3] -generated file in the format of .tra (rows), which consists only transition matrix and thus requires loading additional PRISM-generated labeling/configuration file from the same model: .lab files for adding propositions to states, including specifying INITIAL and FINAL state(s).

2.1.2 .hpa File Formatting

HPA is different from PA in that: HPA requires determinization and reachability of all states from initial state; each state/node in an HPA is assigned a level.Their formatting only differ in specifying transition.

```
....
//Transition Format:
//"source state id" input "end state id" probability distribution
//(sid) input (eid1) pr1 (eid2) pr2 ...
2 2Sitem 1 1
2 1Sbid 3 3/5 5 2/5
....
```

2.1.3 Emptiness

To check the emptiness of the language $\mathsf{L}_{>x}(\mathcal{A})$, user needs to specify the probability threshold x, which is given a default value of 0.5 in the tool UI.

To execute verification, click on the ”forward” and/or ”backward” buttons.

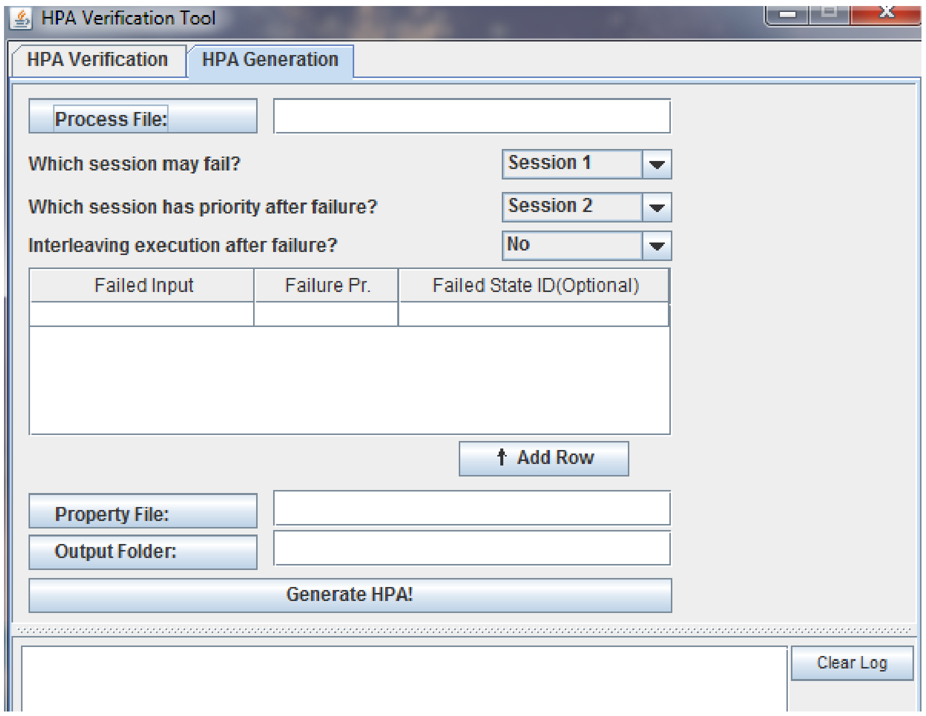
2.1.4 Robustness

Exact robustness can be obtained in case backward algorithm converges on the HPA example; or else, the ”precision” specified by user will be used to decide a robustness range.

2.2 1-HPA Abstraction

The HPA generation part of the tool does the following: it abstract 1-HPA model from Single Process, Failure Specification, and Property.

[3]PRISM model checker website: http://www.prismmodelchecker.org/


HPA Verification Tool

HPA Verification | HPA Generation

Process File:

Which session may fail? Session 1

Which session has priority after failure? Session 2

Interleaving execution after failure? No

Failed Input	Failure Pr.	Failed State ID(Optional)

Figure 2: HPA Tool User Interface for Model Abstraction

To generate a 1-HPA, user loads one PA file as g_0, another PA file as g_p, and specify failure definitions. Then simply click ”Generate HPA!”, the program will generate a HPA. The way the program creates multi sessions is that it creates 2 different instances of g_0 with only one thing common - the T symbol.

In detail, user loads one PA file as g_0, specify failure definitions, and load another PA file as g_p. Then simply click ”Generate HPA!”, the program will generate a HPA.

The way the program creates multi sessions is that it creates 2 different instances of g_0 with only one thing common - the T symbol. Note that in each session, the set of states absorbing T symbols and the set of states absorbing external inputs have to be disjoint. Any violation will fail our abstraction process.

We then compose each two sessions in the traditional way with partial order reduction, and also introduce failure on one session. Note the single sessions are all PAs with only probability-1 transitions, but the composed session with failure is a 1-HPA, say g_{12f} , which has probabilistic distributions on the states where failure occurred.

2.2.1 PA File Formatting

Our tool allows users to specify PAs and HPAs in the form of adjacency lists with transitions labeled with input symbols and probabilities. Optionally, it also allows definition of propositions over states and transitions. All processes in the HPA Generation part of the tool share the same formatting for probabilistic automata (PA). Below is an sample PA file.

```
//Comments are after "//", will be omitted in parsing
3 //First specify total number of states
//each state line starts with the state name (String),
//followed by propositions separated by #
s0 #INITIAL //only one initial state
s1
s2 #FINAL //space is NOT required before #
//Transition format:
//source state name-input->
//end state name1,pr1 #prop1;end state name2,pr2 #prop2;...
s0-a->s1 //one end state, probability undefined, so pr=1
s0-b->s1,0.8 #SYNC;s2,0.2 //a probability distribution
s1-T->s2 #proposition-required-for-triggering
```

2.2.2 PA Attributes

INITIAL state and FINAL state(s) For now it's required there is one and only one INITIAL state, and the initial state has no incoming edge from other states than FINAL state(s) on "new" symbol.The number of FINAL states are not forced.

Propositions Simple string propositions are defined to states and transitions in the process file. State propositions implies "required-for-sync" in property processes. Transition propositions are classified into two groups of "required-for-triggering" and "satisfied", while all propositions defined in PA files are required for triggering.

T symbol and T-transition The T symbol is a special input on transitions, different from external inputs. It's used temporarily for timing and synchronizing purpose, and will be removed before generating HPA, thus in the final HPA there will be no T symbols nor T-transitions. Each T symbol (i.e. one T-transition) denotes a time unit dealing with an external input, so T-transitions always come after external inputs. For now, only the T-transitions require and force synchronization in composing g_1 and g_2.

Cache is allowed, so it's legal to define multiple external inputs and then add T-transitions in process files, e.g. $1 - a \to 2 - b \to 3 - T \to 4 - T \to 5 - T \to 6$.

2.2.3 Composition with Partial Order Reduction - g_{12}

Partial order reduction is automatically applied while composing the two processes g_1 and g_2. Since g_1 and g_2 are identical, g_1 is always given priority in the reduction. We will use the notation "$g_1 > g_2$" to represent this priority. Thus whenever g_1 and g_2 are both active, g_1 will keep executing until reaching T-transitions or final states. Specially, we have "new" symbols defined for each process. Each FINAL state goes to the INITIAL state on a special input "new", and beginning a new session executes non-deterministically. Thus when both servers are at FINAL states, both of them can start new sessions on their own “new”, ignoring priority. In the case priority of $g_1 > g_2$ is defined, when g_1 is at FINAL state while g_2 is not, g_1 can start new session on “$g_1$new” when g_2 is not dealing with external inputs; when g_2 is at FINAL state while g_1 is not, g_1 will keep running, and no synchronization required since g_2 is inactive.

2.2.4 Failure Specification

As stated before, we are considering probabilistic failure. For two concurrent processes, only one process may fail, thus the failure is defined over the input and state of one single process only. After failure occurs, no more failure will occur. After both processes complete current sessions at failure, only the remaining process which did not fail will start new sessions. We provide two options for the overall system to recover from failure and complete remaining tasks. After failure and before starting new sessions: either one process completes first then the other; or they each take one step interleavingly and execute one external input.For example, suppose g_2 may fail and the defined after-failure priority is $g_2 > g_1$, then either: 1) g_2 will complete session first, and then g_1 completes current session, and finally g_2, the server which didn’t fail, will start new sessions, and this generates g_{2f1}; or: 2) g_2 then g_1 will execute one external input alternatively, then g_2 start new sessions, and this generates g_{2f1i}. Therefore before generating HPA user needs to specify which process may fail, the failure definitions, which process has priority after failure, and whether they execute interleaving after failure.

Next, We compose g_{12f} with an incorrectness property, which specifies a safety property and is denoted by a deterministic PA g_p with only probability-1 transitions.

2.2.5 Composition with Property Automaton g_p

Currently we study only safety property, and in the implementation we use incorrectness property instead of correctness property. A property automaton g_p is defined on one session, say g_1, but determinized for all symbols in g_1 and g_2. An ERROR state is defined for determinization, g_p will stay at current state on g_2-only inputs, and go to ERROR state on g_1’s undefined inputs.
The new FINAL states after composition are those states where both g_p and g_1 (suppose g_p is defined on g_1) are at FINAL states.
Note property automaton can be defined on either session, but without failure the property automaton won’t reach final states, i.e. concurrent processes will not violate the property, thus generated HPA will have no final states and robustness is always 1.

Finally, we remove all T symbols and do a normal determinization. Using the approach, we obtain a 1-level HPA and check its emptiness using both the forward and backward algorithms.

2.2.6 Output Format

There are 3 types of output files: plain files (usually named with prefix ”file_”) containing every detail of a PA; ”FAT_” files contain less information, are for graphically presentation in an external tool called FAT[4] ; ”HPA_” files (maybe in .hpa format) are ready to be loaded in the Verification part of the tool.

3 A Complete HPA Verification Example

Here we give a complete example including generating HPA and all its intermediate files,[5] and verification over the generated HPA as well. The example can be found at ”examples/s_small” on project website.

3.1 HPA Generation Example

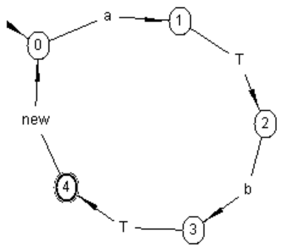
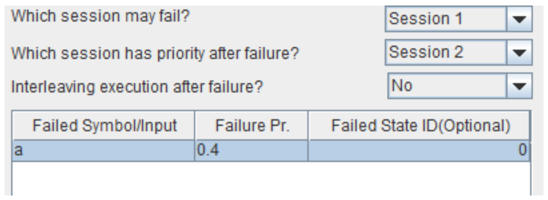


Figure 3: HPA Example: User Input Single Session g_0. Load file ”/examples/1Generation/s_small/s_small.txt” as single process in the Generation tab of the tool. The state with a right arrow is the initial state. Double circle denotes final states. On edges are the input symbols. Probabilities are not presented in the graph.



Which session may fail?	Session 1
Which session has priority after failure?	Session 2
Interleaving execution after failure?	No

Failed Symbol/Input	Failure Pr.	Failed State ID(Optional)
a	0.4	0

Figure 4: HPA Example: Failure Specification: g_1 failed, g_2 had priority after failure, and no interleaving behavior after failure.

[4]FAT tool website: http://cl-informatik.uibk.ac.at/software/fat/

[5]Graphs are displayed using external tool ”FAT”.

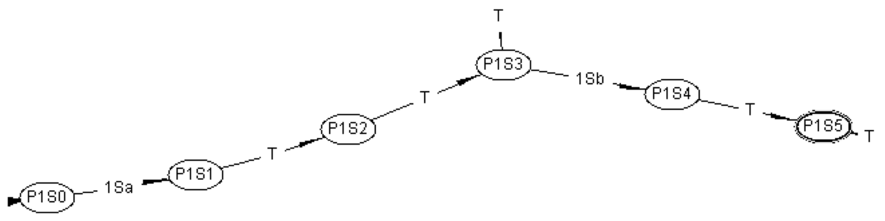


Figure 5: HPA Example: User Input property g_p. Load file "/examples/1Generation/s_small/s_small_property.txt" as property in the Generation tab of the tool. The property automaton is loaded then determinized by the program.

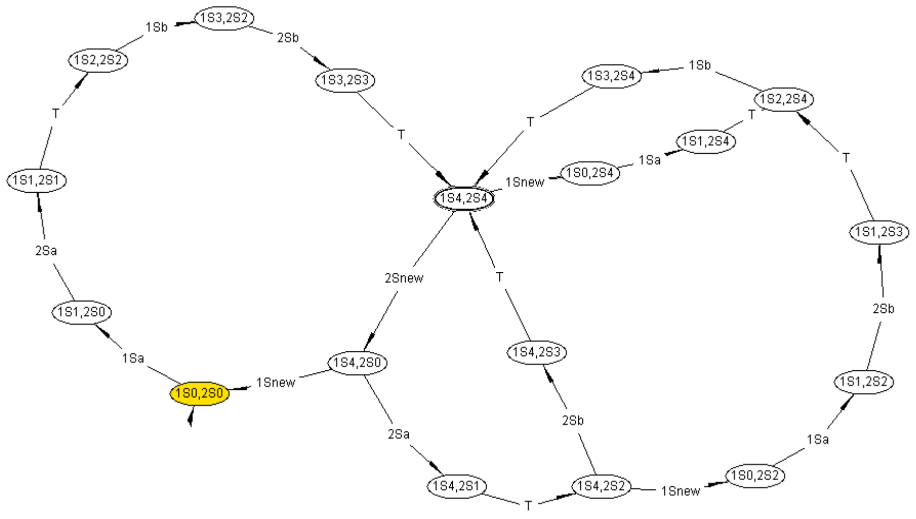


Figure 6: HPA Example: intermediate process g_{12}, the composition of the two single processes.

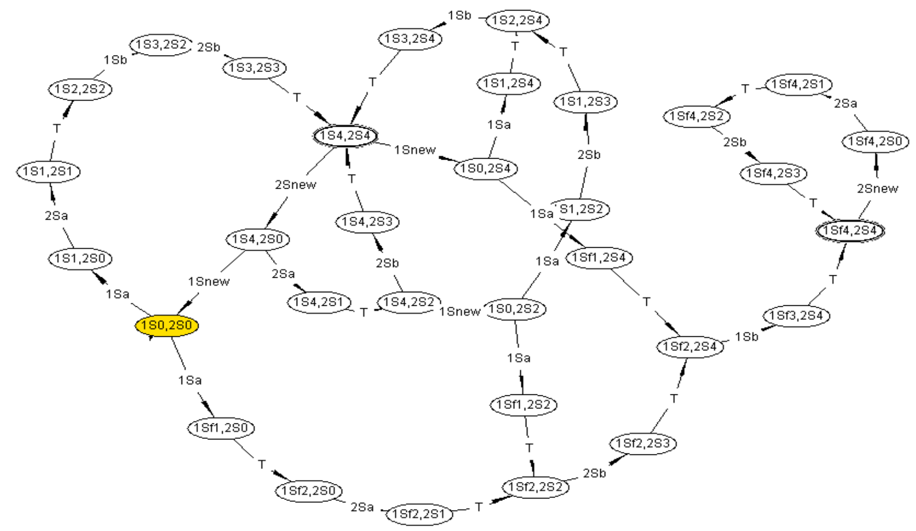


Figure 7: HPA Example: intermediate process g_{21f}, with failure specification applied to g_{12}.

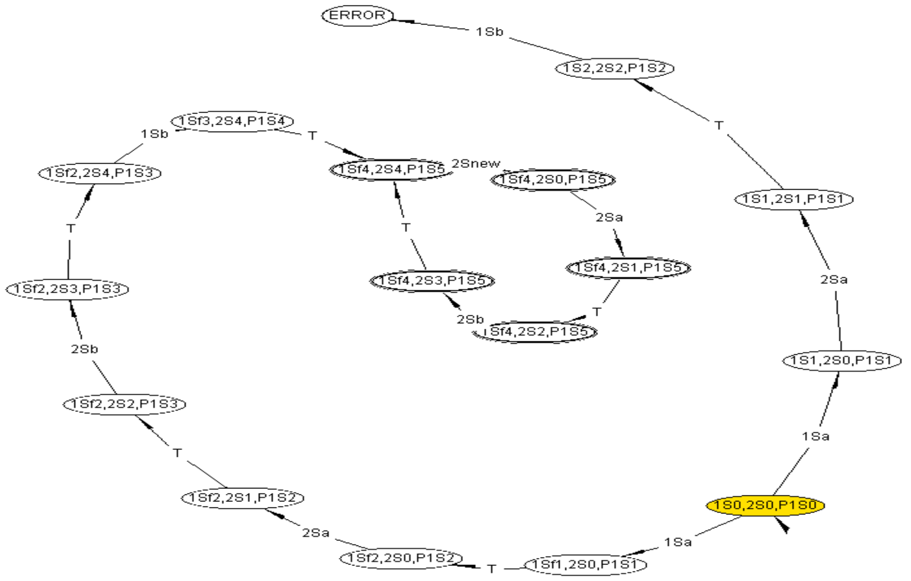


Figure 8: HPA Example: intermediate process g_{21fp}, the composition of g_{21f} and $g_{p-determinized}$, synchronized on shared variables.

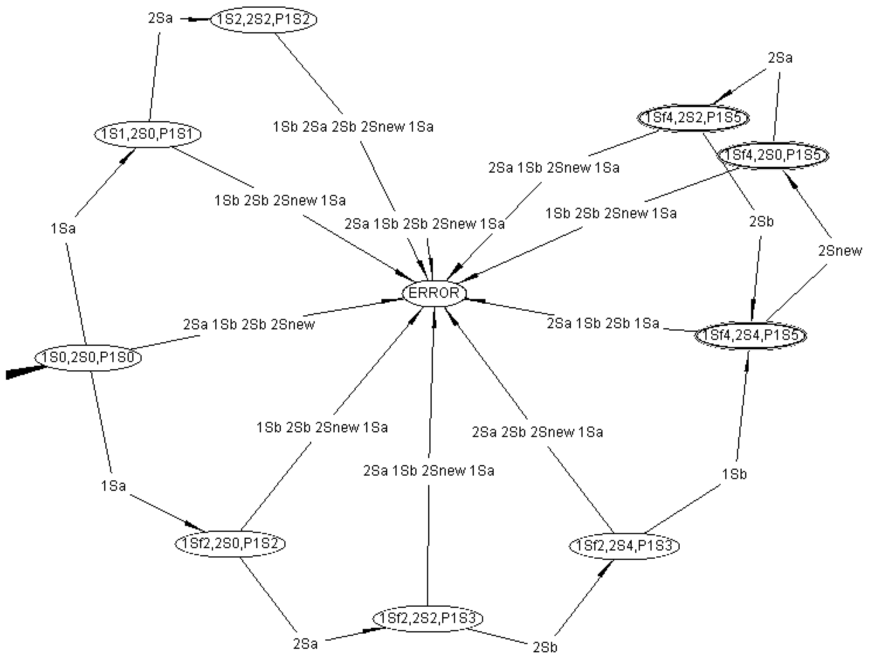


Figure 9: HPA Example: *hpa*

3.2 HPA Verification Example

In the Verification part of the HPA tool, simply load HPA and run. The log below is for the 1-HPA generated in last section. For non-empty results the tool will also give an example run.

```
----------ROBUSTNESS ALGORITHM----------
Note:L_UpperBound = 85899345920.
Minimum empty is 2 / 5 at L=3, thus robustness value is 3 / 5.
----------DECIDABILITY ALGORITHM: BACKWARD----------
L(A) is empty with threshold x=4001 / 10000 within L=3.
[Backward Convergence]
*Time spent on backward algorithm: CPU time=15600100
----------DECIDABILITY ALGORITHM: FORWARD----------
L(A) is empty with threshold x=4001 / 10000 at L=2.
[Forward Convergence]
*Time spent on forward algorithm: CPU time=15600100
----------DECIDABILITY ALGORITHM: FORWARD----------
L(A) is non-empty with threshold x=39999 / 100000 at L=1
Accepted sequence and runs: [2, 3]<-[1Sa]<-[0]
*Time spent on forward algorithm: CPU time=0
----------DECIDABILITY ALGORITHM: BACKWARD----------
L(A) is non-empty with threshold x=39999 / 100000 at L=2.
Accepted sequence and runs: [0]->[1Sa]->[2, 3]
```

```
*Time spent on backward algorithm: CPU time=0
```