User Manual of the new visualization and navigation features in LTSA

Cédric Delforge

May 30, 2013

1 Introduction

This user manual introduces the new visualization features added to LTSA in the context of my master's thesis: *Visualisation and navigation of large state spaces in LTSA*. All the changes were additions rather than modifications of the behavior of the existing tool. Therefore I will mostly describe the new features. What isn't mentioned in this manual can be assumed to be the same as in the latest version provided by the authors at the time of writing: LTSA v3.0, available at `http://www.doc.ic.ac.uk/~jnm/book/ltsa/download.html`.

2 Compiling and visualizing LTS in the Layout panel

Processes can be modeled in the FSP notation in the *Edit* window. The examples introduced in the book *Concurrency: State Models and Java Programs* can be loaded from *File* > *Examples*. Processes can be compiled and composed in the *Build* menu, or with the corresponding buttons in the tool bar.

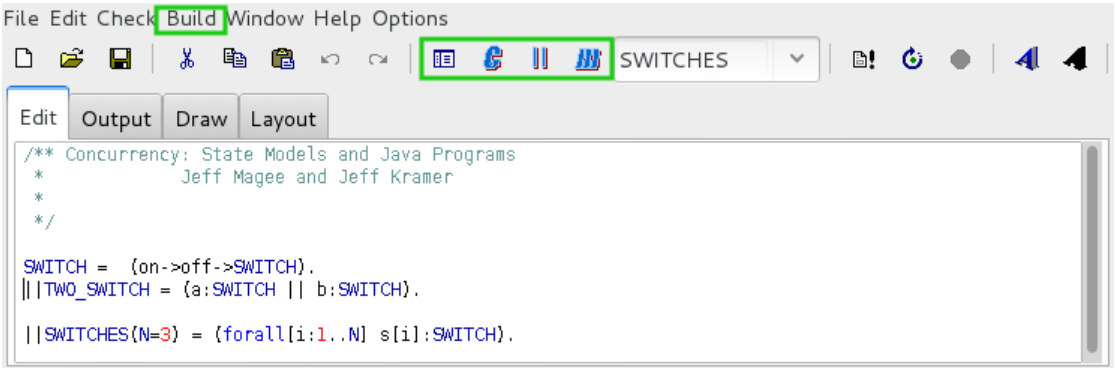

Figure 2.1: Edit window

At this point, the user can either visualize the resulting LTS in the *Draw* window, or in the new *Layout* window. The original and new visualization methods are both available in their own window and can be used indifferently. In the *Layout* window, one can find a list on the left of all the single processes compiled as well as one optional composite process, see (1) in Figure 2.2. Selecting a LTS from the list will display on the canvas (3). It can be cleared by pressing the *Clear* button in the tool bar (2). Multiple LTS can be displayed simultaneously by checking *Options* > *Multiple LTS in Draw and Layout windows*. In multiple mode, selecting a LTS from the list will add it to the canvas, alongside those already displayed, and deselecting it will remove it. The state machines are allocated an equal partition of the canvas as seen in Figure 2.3. As the number of states grows rapidly for highly concurrent models, it might be desirable to set a higher bound on the total number of states a LTS can have to be displayed. This is provided in the menu under *Options* > *Set limit on the number of states to display*. State machines beyond that limit can still be displayed but the user will be asked for a confirmation.

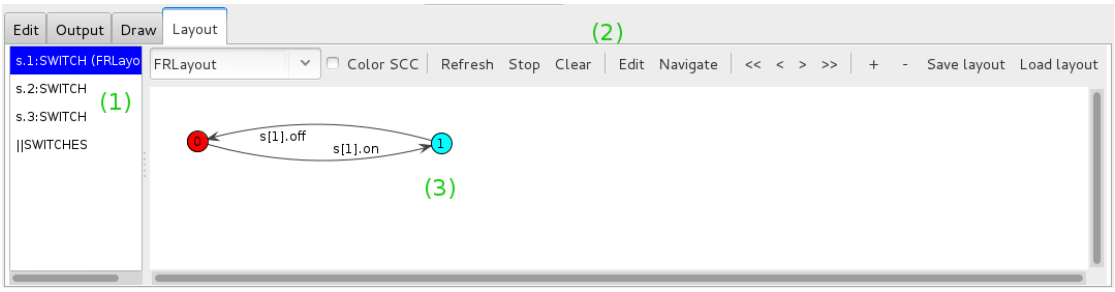

Figure 2.2: Layout window

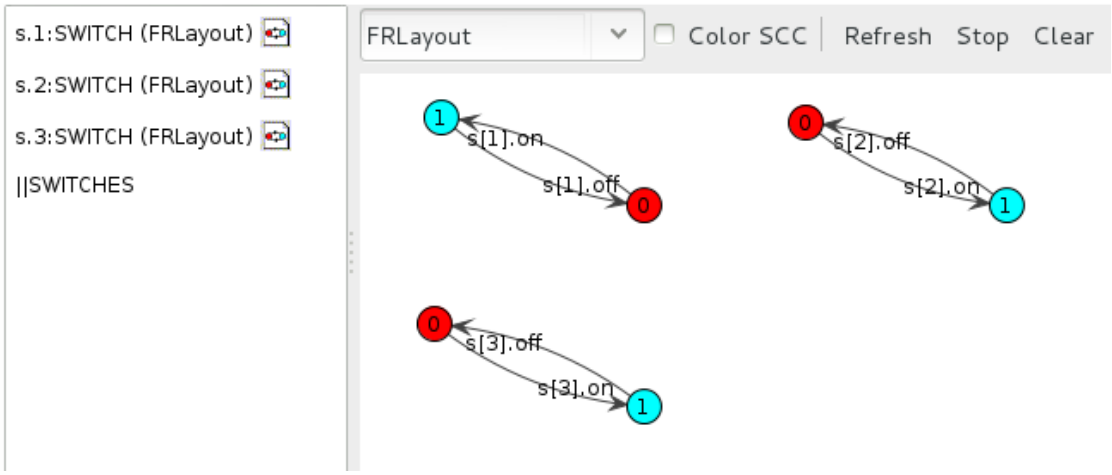

Figure 2.3: Partitioned canvas

3 Layouts

LTS are displayed according to layout algorithms.

Iterative algorithms

Kamada-Kawai, *Fruchterman-Reingold* and *ISOM* (inverted self-organizing map) are three force-directed algorithms for graphs. These algorithms generate layouts iteratively by trying to keep vertices connected by an edge close, and disconnected vertices not too close, and by minimizing edge crossing. They produce the best results on graphs without any particular structure.

Circle algorithm

The *Circle* algorithm places all the vertices on a circle the size of the canvas. As it doesn't attempt to minimize edge crossing, it rarely produces interesting results, except for very small processes.

Hierarchical algorithms

The *Tree-like LTS* layout algorithm adapts a tree algorithm so that it can be applied on LTS graphs. The initial state, numbered 0, is the root of the tree, and all the states one transition away become to its children, those then become the root of a new tree, recursively. The strength of this layout algorithm is to properly visualize the hierarchy inherent to the interleaving of instructions in concurrent programs. Another algorithm produces hierarchical results while using the space more efficiently: *Radial LTS*. It uses the idea of a radial tree. In a radial tree, every node at a successive depth of the tree is put on a concentric circle increasingly larger than the one of its parent. While more dense, it doesn't visualize a hierarchy as clearly as the *Tree-like* layout.

Those layout methods can be selected from the drop-down box of the layout toolbar, see (1) in Figure 3.1. Once selected, a layout will be applied for all the LTS displayed

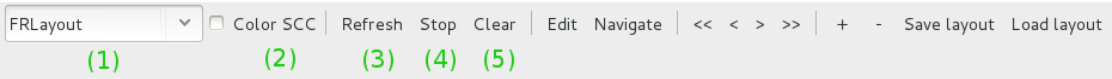

Figure 3.1: Layout controls

next. If the user wishes to apply right away, he can press the *Refresh* button (3). If multiple LTS are displayed, refreshing will apply the selected layout on all of them.

Parameters can be set for some layouts in *Options* > *Layout parameters* seen in 3.2:

Kamada-Kawai

Length factor is a multiplicative factor that specifies the preferred length of an edge.

Disconnected distance multiplier is a multiplicative factor that specifies the fraction of the graph diameter that is used as distance between two disconnected vertices.

Fruchterman-Reigold

Attraction multiplier characterizes how much the edges keep their connected vertices close by.

Repulsion multiplier characterizes how much the vertices tend to push each other away.

Tree-like LTS

Horizontal distance The horizontal spacing between two nodes on the same level.

Vertical distance The vertical spacing between two nodes with one level of difference.

Radial LTS

Horizontal distance The initial horizontal spacing between two nodes on the same level before transformation.

Vertical distance The initial vertical spacing between two nodes with one level of difference before transformation.

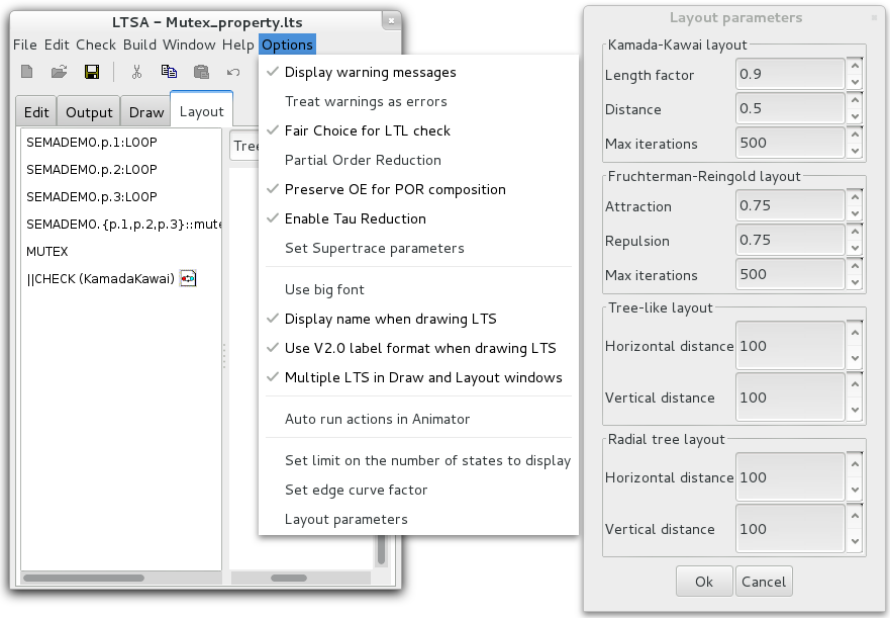

Figure 3.2: Options including the layout parameters

For LTS with hundreds of states, iterative algorithms can take a long time to complete. The maximum number of iterations can be set for the *Kamada-Kawai* and *Fruchterman-Reingold* layouts, under *Options* > *Layout parameters*. Regardless, an iterative layout algorithm can be stopped at any time with the *Stop* button (4).

4 Strongly connected components

A directed graph, such as LTS, can be partitioned into sets of strongly connected components (SCC). A subgraph of SCC is a set of states such there is a path between every two states in the SCC. Unique colors can be assigned to SCC clusters by selecting the *Color SCC* option in the toolbar, see (2) in 3.1.

5 Interacting with the visualization

Clicking on a state with the left mouse button will select it, and optionally deselect any other. A state can be added to an existing selection by holding the *shift* key while

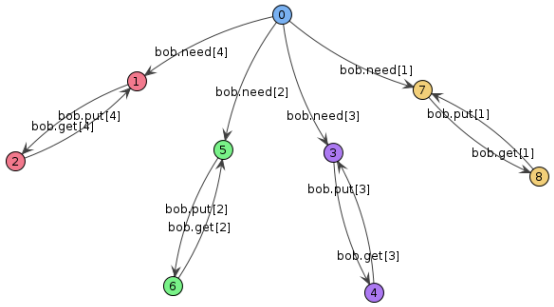

Figure 4.1: SCC colors

clicking. Multiple states can be selected at once by clicking in an empty part of the canvas and dragging. This will create a selection box such that all the states inside it will be selected. Boxing can of course also add to an existing selection by holding *shift*.

Selection is useful to move states on the canvas, by clicking on one or more selected states and dragging them. With this, it is possible to manually alter the layout of a LTS.

To handle large LTS, one can zoom in and out with either the mouse wheel or the + and - buttons on the tool bar, (5) and (6) in Figure 5.1. The content of the canvas can be translated by clicking somewhere in the canvas with the right mouse button and dragging. This is important to take a close look at a fully maximized LTS.

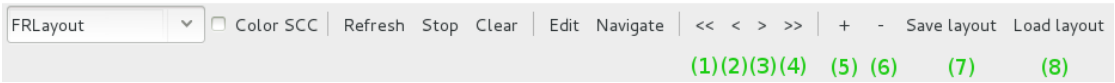

Figure 5.1: Interaction controls

6 Selecting notable states

To ease the manipulation of complex graphs, special selections can be performed automatically, so that those states can handled as a group.

Clicking with the right mouse button on one or more selected states will display a context menu. It offers the following choices:

Select LTS will select the full LTS the selection belongs to. If the selection contains states from multiple LTS, all the corresponding LTS will be selected. Use this to separate overlapping state machines.

Select SCC will select all the SCC clusters the selection belongs to.

Select next states will select all the states one transition away form the currently selected state(s).

Select previous states will select all the states that can reach the currently selected state(s) in one transition.

Select reachable states will select all the states reachable form the currently selected state(s).

Select reaching states will select all the states that can reach the currently selected state(s).

The *reaching, previous, next* and *reachable* commands can also be activated by pressing the corresponding buttons, (1) to (4) in that order on the tool bar, see Figure 5.1. An example of the advanced navigation commands is provided in section 8.

The context menu also offers the option to apply one of the layout algorithms discussed previously, on a local part of the LTS defined by the selected states.

7 Navigation

It is possible to interact with the LTS in a different way. In the *navigation* interaction mode, the layout cannot be modified, but it is possible to simulate the execution of a process by navigating its sequences of states and transitions. The user can freely move between the edition and navigation modes with the *Edit* and *Navigate* buttons, see Figure 7.1. Starting from one or more currently activated states, all the actions possible

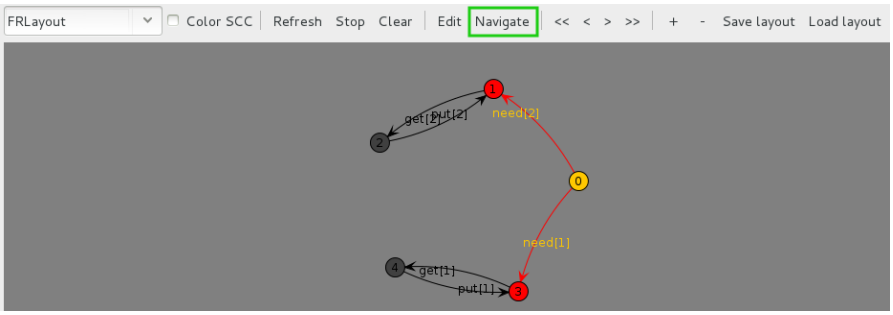

Figure 7.1: Navigation mode

at that point and the consequent states are highlighted. By clicking with the left mouse button on a highligted state, the atomic action is performed and the new state is reached, modifying the set of possible transitions. An example is provided in Figure 7.2 where, from left to right, the following trace is reproduced: `need[1], get[1], put[1]`.

The *navigation* mode works together with the *edition* mode, as any state selected in edition is activated in navigation, and vice versa.

While no context menu is provided in navigation mode, the *next, previous, reaching* and *reachable* buttons are still enabled and can be used for faster navigation.

Alternatively, *Animator* window can be used to execute multiple LTS simultaneously.

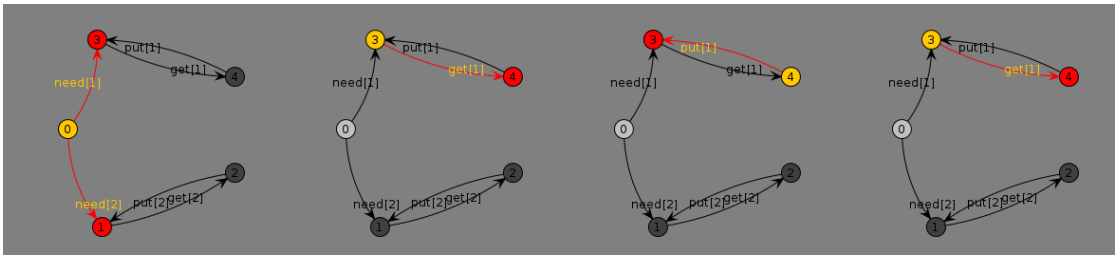

Figure 7.2: Simulating an execution

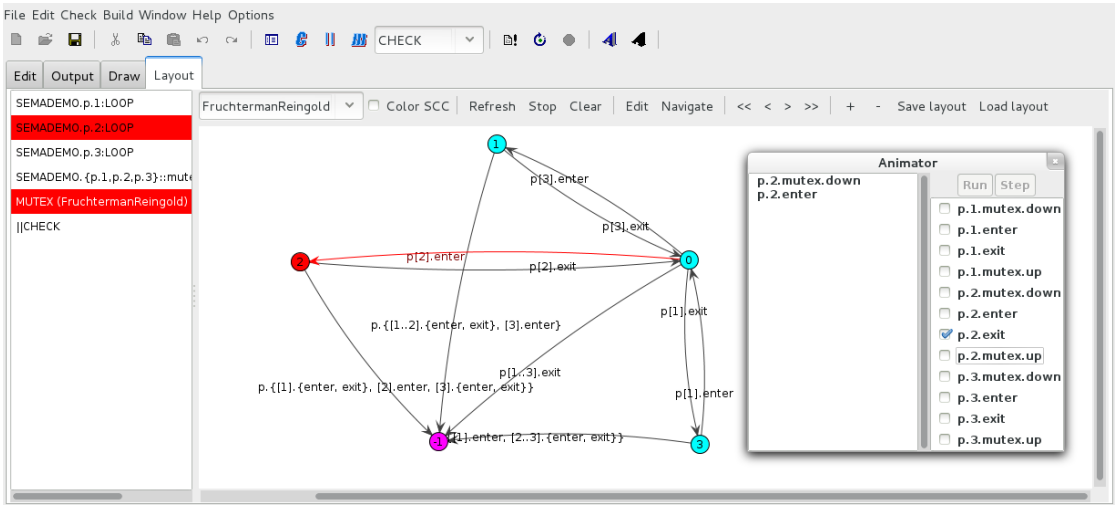

Figure 7.3: Animator

8 Example for the advanced navigation commands

We illustrate *next, previous, reachable* and *reaching* with the example of a thread, as modeled in Figure 8.1. A thread can be in five states: *created* (state 0), *running* (state 2), *runnable* (state 3), *non-runnable* (state 4) and *terminated* (state 1). Actions can

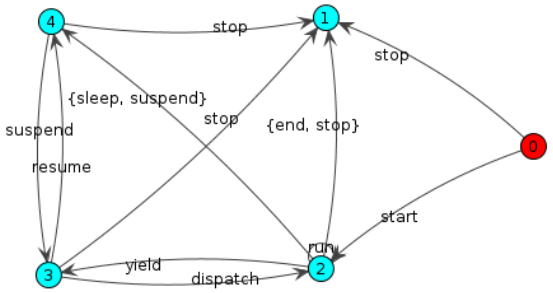

Figure 8.1: States of a thread

change its state, these are visible as labels on the transitions. The thread starts in the non-runnable state. *Next* activates the set of all states that can be reached in one transition from the current states. The thread can either resume, and be in the runnnable

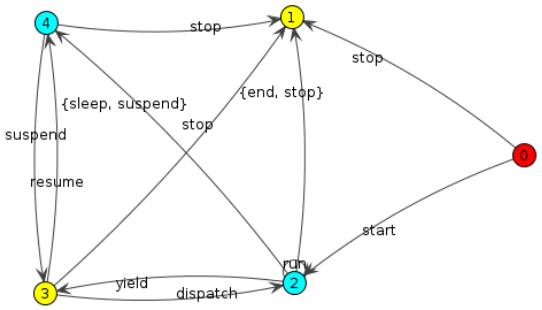

Figure 8.2: Next

state (state 3), or stop and be in the terminated state (state 1). See Figure 8.2.

Previous activates the set of all states that can reach the current state in one transition. To be non-runnable, a running thread (state 2) can be put to sleep, or either a running or a runnable thread (state 2 or 3) can be suspended. See Figure 8.3.

Reachable activates the set of all states that can be reached after any valid sequence of transitions from the current state. A non-runnable thread can end up in any other state, including returning to non-runnable, after any sequence of transitions, except in the created state it can never return to. See Figure 8.4.

Reaching activates the set of all states that can reach the current state after any valid sequence of transitions. A thread can be non-runnable being originally in any other state, including the non-runnable state, except in the terminated state, where it can never resume. See Figure 8.5.

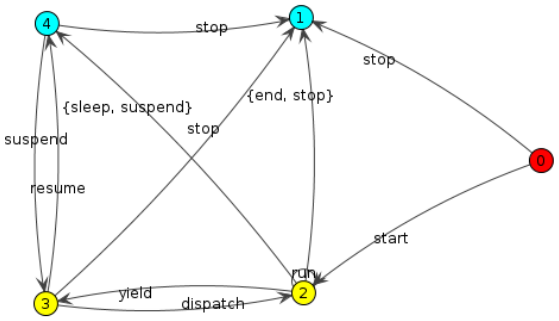

Figure 8.3: Previous

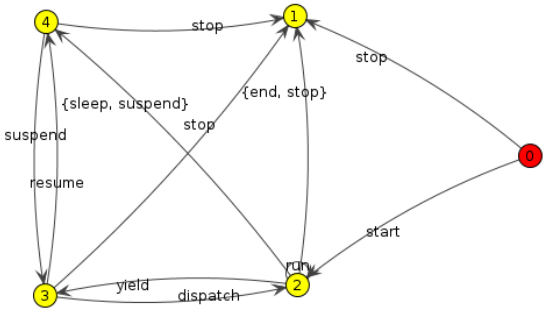

Figure 8.4: Reachable

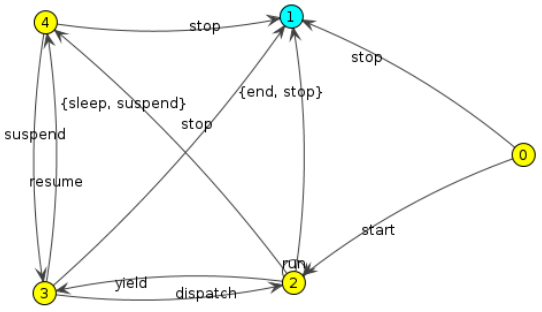

Figure 8.5: Reaching