STS-based Supervision Software

Chuan Ma
cma@control.utoronto.ca
January 21, 2007
(trial version)

Content

What's in the software package?

The following is the content of this package:

```
./readme.odt - this doc in OpenOffice format
./readme.pdf - this doc in Adobe PDF format
./nbc - the NonBlocking Control program compiled in Debian Linux
./nbc.exe - the NonBlocking Control program compiled in Windows XP
./examples - this directory contains a number of tutorial examples
   ./examples/TL - transfer line example
   ./examples/TLand - transfer line example with a simple state replaced by an AND state
   ./examples/TLor - transfer line example with a simple state replaced by an OR state
   ./examples/mutex - mutual exclusion specification expressed in STS framework.
```

While reading this document, the author suggests that the monograph, *Nonblocking Supervisory Control of State Tree Structures*, should be at hand for reference.

Use the software

To use the software in Debian Linux, type

```
      cd path/to/the/control/problem
      path/to/nbc/nbc sts spec
```

in a terminal window. In a windows XP, after running cmd.exe, type

```
      cd path\to\the\control\problem
      path\to\nbc.exe\nbc.exe sts spec
```

in the cmd.exe window. Here

- `sts` is a *text*[1] file containing an STS model
- `spec` is a *text* file containing the logical specification for the STS.

Each control problem (`sts, spec`) must be put in a separate file directory, because the output of the

1 Compared to the *binary* .des file in TCT, a model built in STS is a text file, legible to human.

above command will be printed out in the *current* directory.

Example: To give it a try, go to the directory './examples/TL'. This directory includes the control problem for the Transfer Line example[2]. Take a look at the files `sts, spec` in the directory, and run this example.

If successful, the program `nbc` will print out the following files in the current directory:

1. For each controllable event `ev`, there is an associated *simplified control function* [3] in BDD. The possible file name is either `ev.en` or `ev.en.all`. The 1st file name is used when the event is disabled somewhere in the system. Otherwise the 2nd file name is used.
2. For each controllable event `ev`, there is an associated *largest eligible state tree*[4] in BDD. Its file name is `ev.elig`.
3. The file `controlledbehavior` can decide which basic-state-tree is legal in the controlled system. It's also given as a BDD.

All output files are BDDs in `dot` format. To generate a .ps file for a BDD, it is required to install **`Graphviz`**[5] in your computer. For example, if you wish to draw a picture for the control function of event `alpha`, the following command should be used:

```
        dot -Tps alpha.en -o alpha.ps
```

The BDD in `alpha.en` is now drawn in the ps file `alpha.ps`. Pictures in other popular formats such as png, svg, jpeg and gif can also be generated using `dot`. In windows XP, there is also a windows interface for this program. For details, please refer to **`Graphviz documentation`**[6].

STS model

In this section, the STS models in the `./examples` directory are going to be used to help you create your own STS models.

An STS model consists of the following components in this order:

1. *state tree*
2. *a list of holons* matched to the OR states on the state tree
3. *initial sub-state-tree* (analogous to the initial state of an automaton)
4. *set of marker sub-state-trees* (analogous to the set of maker states in an automaton)
5. *set of memories* (analogous to the specification automata in TCT, for convenience purpose in generating STS specification)

Case 1 (synchronous product systems):

The STS model under `./examples/TL` is saved in the file `./examples/TL/sts`. Before taking a look at the file, note:

1. A state (or event) label is a string of alphanumeric letter or underscore.
2. Two different states on a state tree *cannot* have the same state label.
3. The basic construct is `set`. We use a simple syntax here. For example, `{ element1, element2 element3}` is a set of 3 elements. An element may be a state, an event, a transition, a function map, or even another set. We can use either whitespace or comma as the delimiter.

2 Refer to pages 123-125 of the monograph or the Notes by Professor Wonham.

3 Refer to Section 4.4 of the monograph.

4 Refer to page 47 of the monograph for the definition.

5 Graphviz can be downloaded at `http://www.graphviz.org/Download..php`

6 Graphviz dot documentation is available at `http://www.graphviz.org/Documentation/dotguide.pdf`

4. For convenience, to enter a holon, it is only required to enter its internal transitions. The boundary transitions (if exist) are actually described in its parent holon's internal transition structure. We can do this because a well-formed STS must satisfy the *boundary consistency,* defined on page 38 of the monograph.
5. A sub-state-tree is compactly described by the set of its *active states* on the sub-state-tree. Please refer to pages 27-30 of the monograph for detail.

Also note: in the following STS model, the strings starting with % are added to explain the format, which is not a part of the model. When creating your own STS model, don't include these comments.

./examples/TL/sts:

```
% state tree definition
root = plant % The root state is assigned a name 'plant'.
{ % The expansion (type) functions of the state tree is defined in this set
plant = AND { M1, M2, TU, B1, B2} % plant is an AND state expanded by M1, M2...
M1 = OR { M1_0, M1_1 } % M1 is an OR state expanded by M1_0 and M1_1
M2 = OR { M2_0, M2_1 }
B1 = OR { B1_0, B1_1 }
B2 = OR { B2_0, B2_1 }
TU = OR { TU_0, TU_1 } }

% list of holons. There are 5 holons matched to the 5 OR states
% defined in the state tree.

M1 % Holon M1 is matched to the OR state M1.
{1} % Set of controllable events in M1. {} if no controllable events
{2} % Set of uncontrollable events in M1. {} if no uncontrollable events
{ % set of internal transitions in M1.
[M1_0 1 M1_1]
[M1_1 2 M1_0]
}

M2
{3}
{4}
{
[M2_0 3 M2_1]
[M2_1 4 M2_0]
}

TU
{5}
{6 8}
{
[TU_0 5 TU_1]
[TU_1 6 TU_0]
[TU_1 8 TU_0]
}

B1
{3}
{2, 8}
{
[B1_0 2 B1_1]
[B1_0 8 B1_1]
[B1_1 3 B1_0]
}
```

```
B2
{5}
{4}
{
[B2_0 4 B2_1]
[B2_1 5 B2_0]
}

{M1_0 M2_0 TU_0 B1_0 B2_0} % initial sub-state-tree
{ {M1_0 M2_0 TU_0 B1_0 B2_0} } % set of marker sub-state-trees, set of sets!
{B1 B2} % set of memories. {} if no memory exists
```

This Transfer Line example is a special STS. In the following two cases, we will explain how to enter an STS model with hierarchy.

Case 2 (STS model with different levels of OR superstates):

In the above Transfer Line example, if we replace the originally simple state `M1_1` by an OR superstate (with the same state label), assume `M1_1` has two simple states, `M1_1_0` and `M1_1_1`, as its children, and assign a new holon to this new OR state, we create an STS model located at `./examples/TLor/sts`, which is also given as below. The changes are highlighted. Again, the comments starting with % shouldn't appear in a real STS model.

`./examples/TLor/sts:`

```
root = plant
{
plant = AND { M1, M2, TU, B1, B2}
M1 = OR { M1_0, M1_1 }
M2 = OR { M2_0, M2_1 }
B1 = OR { B1_0, B1_1 }
B2 = OR { B2_0, B2_1 }
TU = OR { TU_0, TU_1 }
M1_1 = OR { M1_1_0, M1_1_1 } % without this line, M1_1 is a simple state by default
}

% a new holon M1_1 is added
M1_1
{c1}
{}
{
[M1_1_0 c1 M1_1_1] % Again, only internal transitions are required.
}

M1
{1}
{2}
{
[M1_0 1 M1_1_0]  % It used to be M1_1. The change reflects the fact that the target state
                 % of this transition is actually M1_1_0, a child of M1_1. The program can
                 % automatically derive the internal transition of M1: [M1_0 1 M1_1] and
                 % the boundary transition of M1_1: [M1_0 1 M1_1_0].
[M1_1_1 2 M1_0]
}

M2
{3}
{4}
```

```
{
[M2_0 3 M2_1]
[M2_1 4 M2_0]
}

TU
{5}
{6 8}
{
[TU_0 5 TU_1]
[TU_1 6 TU_0]
[TU_1 8 TU_0]
}

B1
{3}
{2, 8}
{
[B1_0 2 B1_1]
[B1_0 8 B1_1]
[B1_1 3 B1_0]
}

B2
{5}
{4}
{
[B2_0 4 B2_1]
[B2_1 5 B2_0]
}

{M1_0 M2_0 TU_0 B1_0 B2_0}
{ {M1_0 M2_0 TU_0 B1_0 B2_0} }
{B1 B2}
```

Using the same approach, any level of OR states can be added into an STS model. The imposed constraint of a holon must be obeyed here: *the boundary states of a holon must be simple states* (Refer to Remark 1 on page 37 of the monograph).

Case 3 (STS model with local concurrency):

In the above Transfer Line example, instead of replacing the originally simple state `M1_1` by an OR superstate, we assume that `M1_1` is an AND superstate, which has two OR states, `M1_1_1` and `M1_1_2`, as its children, and assign 2 new holons to these 2 new OR states. In this way, we create an STS model with local concurrency. The STS model is saved in the file: `./examples/TLand/sts`, which is also given as below. The changes are highlighted. Again, the comments starting with % shouldn't appear in a real STS model.

```
root = plant
{
plant = AND { M1, M2, TU, B1, B2}
M1 = OR { M1_0, M1_1 }
M2 = OR { M2_0, M2_1 }
B1 = OR { B1_0, B1_1 }
B2 = OR { B2_0, B2_1 }
TU = OR { TU_0, TU_1 }
M1_1 = AND {M1_1_1 M1_1_2}
M1_1_1 = OR { a , b }
M1_1_2 = OR { c, d }
}

M1
{1}
{2}
{
[M1_0 1 {a, c} ] % This target state used to be M1_1. The change reflects the
                 % fact that the target state of this transition is actually (a,c),
                 % staying at state a of M1_1_1 and state c of M1_1_2 at the same time.
                 % The program can automatically derive the internal transition of M1:
                 % [M1_0 1 M1_1], the boundary transition of M1_1_1: [M1_0 1 a] and the
                 % boundary transition of M1_1_2: [M1_0 1 c] from this generalized
                 % transition.
[{b d} 2 M1_0]
}

M1_1_1
{sigma}
{}
{
[a sigma b]
}

M1_1_2
{beta}
{}
{
[c beta d]
}

M2
{3}
{4}
{
[M2_0 3 M2_1]
[M2_1 4 M2_0]
}

TU
{5}
{6 8}
{
[TU_0 5 TU_1]
[TU_1 6 TU_0]
[TU_1 8 TU_0]
}

B1
{3}
{2, 8}
{
[B1_0 2 B1_1]
[B1_0 8 B1_1]
[B1_1 3 B1_0]
}

B2
{5}
```

```
{4}
{
[B2_0 4 B2_1]
[B2_1 5 B2_0]
}

{M1_0 M2_0 TU_0 B1_0 B2_0}
{ {M1_0 M2_0 TU_0 B1_0 B2_0} }
{B1 B2}
```

Using the same approach, any level of AND states can be added into an STS model.

Combining the above 3 cases, you can create an arbitrary STS model with both concurrency and hierarchical structure.

STS specification

As illustrated in Section 4.1 of the monograph (pages 77—82), the final STS specification must be in either one of the following types:

1. Type 1 specification: set of illegal sub-state-trees.
2. Type 2 specification: set of the (sub-state-tree, event) pair, meaning that the *uncontrollable event* is forbidden at the *sub-state-tree*[7].

So a specification file, which is the 2nd argument of the program `nbc` (recall: `nbc sts_file spec_file`), is in this form:

```
{ sub-state-tree1, sub-state-tree2, ... }
{ (sub-state-tree_a, event_a), (sub-state-tree_b, event_b),...}
```

Both sets in the specification file are allowed to be empty. Note that a pair is enclosed in round brackets `( )`, while a set is enclosed in curly brackets `{ }`.

An example of a Type 1 specification is given in the mutual exclusion example, located at `./examples/mutex/spec`.

```
{ {M11, M21} }
{}
```

Note again that a sub-state-tree in the program is compactly represented by the set of its active states. This specification says that the system is not allowed to stay at `M11` and `M21` at the same time.

For convenience, the program `nbc` can automatically generate type 2 specification from a memory in the STS model. Recall that the task of a memory (specification holon) is to record the system behavior and no control is attached. So all uncontrollable events should be defined in each state of a memory. If we tell the program that a holon is a memory in the STS model, but some uncontrollable events are missing in some states of the holon, the program can recover the original memory by adding all missing uncontrollable events and automatically add a set of Type 2 specifications to the control problem. That is, the program *thinks* that the memory wishes to forbid the missing uncontrollable events at the given states[8]. For example, in the last line of the Transfer Line model, `B1` and `B2` are defined as memories. However, events `2,8` are missing at state `B1_1`, and event `4` is missing at state `B2_1`. So the program can

7 A possible improvement is to allow all events in a Type 2 specification to be either uncontrollable or controllable.
8 This is consistent with the standard treatment in TCT.

automatically add the missing uncontrollable events to both `B1_1` and `B1_2`, and add the following Type 2 specifications:

```
{({B1_1}, 2),({B1_1}, 8),({B2_1}, 4)}
```

In summary, we have two basic approaches to derive STS specification:

1. Describe the logical specification in a separate file `spec`, and/or
2. describe the specification using the *memories* embedded in an STS model.

With the STS models and specification files in hand, it is suggested that you should try every examples under the directory `./examples`.

Important notes

<u>BDD encoding convention</u>

Each state label on a state tree is represented by an alphanumeric string. However, BDD cannot encode strings. So for each OR superstate, we have to at first map its components into a list of consecutive numbers and then let BDD encode these numbers. For example,

let A = OR { a0 a1 a2 a11 }.

In the 1st step, we construct an ordered-list by sorting the set { a0 a1 a2 a11 } in lexicographic order:

a0, a1, a11, a2 (notice that 'a11' < 'a2' in lexicographic order)

Then map each element to its index in the list,

map['a0'] = 0, map['a1'] = 1, map['a11'] = 2, and map['a2'] = 3.

Because the number of states in A is $4 \leq 2^2$, the program `nbc` assigns two bdd variables for this OR superstate, normally called A_0, A_1 Encoding an integer using binary variables is standard. For example, the number 2 (i.e., state 'a11') in the above example is encode by $A_0=0, A_1=1$. For details, please refer to the Buddy[9] documentation for the encoding convention of integer numbers.

I assumed that this encoding convention is easily understood. So the current version of the program `nbc` doesn't print out the above map (from state label to integer) . However, I may add this printout function in a later version if requested.

<u>What if I make a mistake while editing the STS and its specification?</u>

As far as I know, the program can catch a number of errors. Among them are the following cases

1. unknown state label,
2. unknown event label,
3. wrong state tree expansion,
4. missing the closing brace `}` or `)`

However, the error messages may not be that informative.

A better user interface is still under development. The ultimate goal is to develop a web-based interface for the STS framework. Time is ticking...

9 Buddy is a BDD package, available at `http://sourceforge.net/projects/buddy`