Usage of the model checker

Augusto Peres

May 27, 2020

1 Expressing the formulas

The DTL formula we want to model check should be passed as one of the arguments. To express the formulas all parenthesis should be used whenever an operator is use, this includes $\wedge$, $\implies$, $\vee$.

For example $@_1[p \implies q]$ should be expressed as `@_1((p)=>(q))`. Spaces should also not be used to express formulas. Communication formulas are expressed as `c_2(otherformulas)`.

Note: When not using the bounded options it is not possible the usage of $\vee$, $\wedge$ or any other dual operators.

2 Expressing transition systems

The transition system should be encoded in a file. The lines in that file must be of the following form:

1. `states 1 2 3 4`. This line says the states present in the system
2. `initial 1 2`. This lines states the initial states
3. `actions agent a1 a2`. This line states the actions of each agent. agent should be an integer
4. `symbols agent p1 p2 p1`. States the propositional symbols available to each agent.
5. `label state agent p1 p2`. States the propositional symbols in the language of the agent that are present in the state.
6. `state action state`. Describes the transition relation.

3 Using the model checker

The model checker is used as a command line application with the following options.

3.1 Visializations

Transition systems can be outputed to the graphviz format by calling

```
./Main -toGraphviz path-to-the-file-containing-the-transition-system
```

This outputs the full the transition system and the full simplified transition system, *i.e*, the transition system with all dead or unreachable states removed.

This can be copy pasted here to visualize.

3.2 Model checking

To get a simple yes or no answer just use for

```
./Main -modelCheck <path-to-transition-system> <formula> <number of agents>
```

This uses the default automata theoretic approach.

To use the bounded approach

```
./Main -modelCheck <path-to-transition-system> <formula> <number of agents> -bounded <n
```

To get one counter example we use

```
./Main -oneCounterExample <path-to-transition-system> <formula> <number of agents>
```

This outputs something of the form

```
CounterExample [ [(s1, x1), (s2, x2)...(sn, x2)], [(sn, xn), ..., (sn, xn)] ]
```

This corresponds to the infinite path in the dot product of DTS with the automaton that witnesses the persistence property. Projecting the first coordinates yields an infinite path in the transitioon function.

To use the bounded model checking approach

```
./Main -oneCounterExample <path-to-transition-system> <formula> <number of agents> -bou
```

The output should be something of the form

```
fromList [("0_a":True)..("0_p1":True)...]
```

Corresponding to the solution of the formula and the symbols present in each state. The number before "_" indicates the action taken at that step.

4 Some examples

```
./Main -modelCheck t8States2Agents1.txt "(@_1(p1))=>~(@_2(q1))" 2
True


./Main -modelCheck t8States2Agents1.txt "@_2(X(c_1(p2)))" 2
True


./Main -modelCheck t8States2Agents4.txt "(@_1(c_2(q1)))=>(@_1(p1))" 2
False


./Main -modelCheck t8States2Agents4.txt "@_1(F((p1)/\\(p2)))" 2 -bounded 0
True

./Main -modelCheck t8States2Agents4.txt "@_1(F((p1)/\\(p2)))" 2 -bounded 2
False


./Main -oneCounterExample t8States2Agents4.txt "@_1(F((p1)/\\(p2)))" 2 -bounded 2
Just (fromList [(0_"a",True),(0_"b",False),(0_"c",False),(0_p1,False),(0_p2,False),(0_c
```