MTCPS-Notes

Daniel Nykjær, Marco Justesen, Patrick Østergaard, Ivik Hostrup

June 2022

Contents

1 UPPAAL

Mathematical	UPPAAL	Meaning
$\forall\Box\phi$	`A[]` ϕ	For all states and all paths, ϕ must hold.
$\forall \Diamond \phi$	`A<>` ϕ	For all states there is a path where ϕ holds.
$\exists\Box\phi$	`E[]` ϕ	There exists a path on which ϕ always holds.
$\exists \Diamond \phi$	`E<>` ϕ	There exists a reachable state such that ϕ holds.
$\neg$*deadlock*	`!deadlock`	Not deadlock.
-	ϕ `-->` ψ	Whenever ϕ holds for a state, then ψ will always hold eventually for all paths starting from that state.

Table 1: Explanation and comparison of mathematical and UPPAAL notation for predicate logic.

Symbol	Operator name	Meaning
`&&`	and	`e && f` is true if both e and f evaluate to true
`||`	or	`e || f` is true if e evaluates to true or f evaluates to true
`==`	equal	`e == f` is true if e and f evaluate to the same value
`imply`	implication	`e imply f` is true if e evaluates to false or f evaluates to true
`not`	negation	`not e` is true if e evaluates to false

Table 2: UPPAAL Verifier operators.

- TCTL = Timed Computation Tree Logic -¿ verifier
- || = *Synchronous* parallel composition
- | = *Asynchronous* parallel composition
- $\mapsto$ = Renaming
- $\setminus$ = Hiding

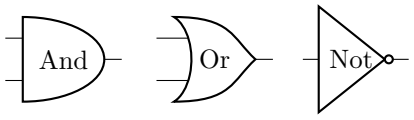

Figure 1: Logic gates.

1.1 Bonus verifier syntax

You can use for loops in the verifier to check i.e. that all instances of a template eventually lead to a specific state. An example of this is shown in listing 1 where we check that there exists a path such that all processes `P` reach the state `done` while $n \leq 2$.

```
1 E<> (forall (i : proc_t) P(i).done && n <= 2)
```

Listing 1: Example of a for loop in the UPPAAL verifier. n is an arbitrary variable.

1.1.1 Fastest diagnostic trace

In some UPPAAL exercises, you are often asked to find the minimum time it takes for a model to satisfy some condition. In this case, you need to set the diagnostic trace to "Fastest" in the options tab, as shown in figure 2.

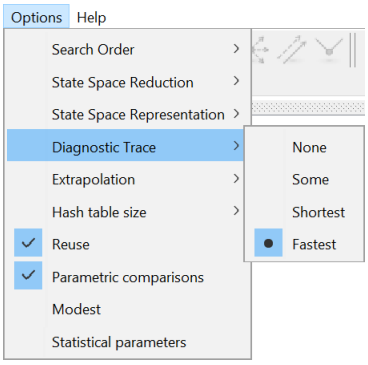

Figure 2: Finding the options for Diagnostic Trace.

Then, run the verifier on the property that needs to be satisfied, and click on the "Get trace" button - this will give you a trace of the fastest execution of the model that satisfies the property (see figure 3).

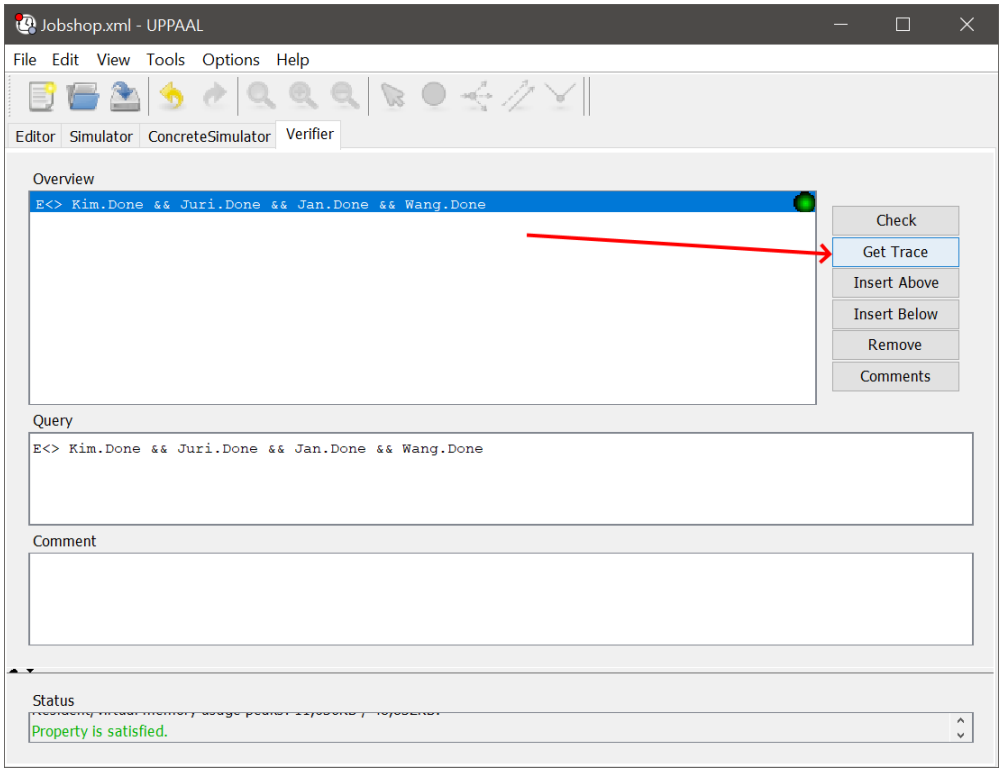

Figure 3: The fastest trace satisfying the property can be found by pressing "Get trace".

Once you have the trace, simply go into the simulator and read the value of the clock variable (you need to make sure to add a clock if there isn't already one as part of the model). Figure 4 shows an example of this where the lowest possible time is 37.

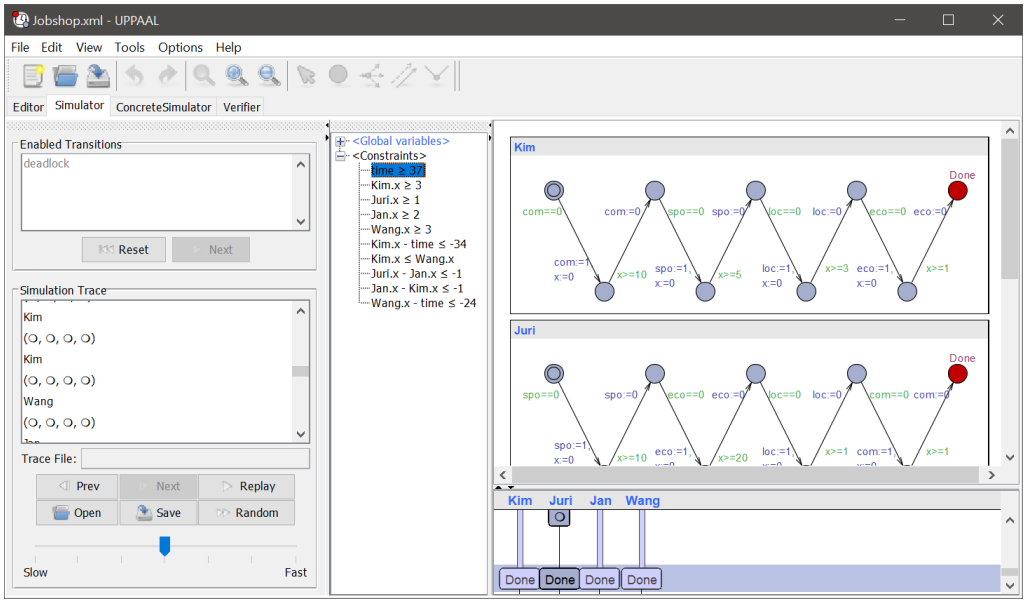

Figure 4: $time \geq 37$ means that the lowest possible time to satisfy this property is 37.

2 MATLAB

2.1 Quirks

2.1.1 Always include at the top of a script

To ensure your script runs correctly without the need for manual configuration and tweaking, make sure to always include the three lines shown in listing 2.

```
clear variables; % clears variables from previous runs
close all; % closes all currently open figures
clc; % clears the command window
```

Listing 2: Include these three lines at the start of all MATLAB scripts.

2.1.2 Discarding variables

The ~ character is used to discard unneeded variables from function invocations. An example of this can be seen on line 2 in listing 3.

2.1.3 Return values

To indicate a return value in MATLAB, you need to define which variable should be returned in the function declarations. An example of this is shown in the `Lap` function on listing 3.

```
1  function d = Lap(B)
2      [~,n] = size(B);
3      if n == 1
4          d = B;
5      else
6          d = 0;
7          for j = 1:n
8              B1 = B(2:n, [1:j - 1, j + 1:n]);
9              d = d + (-1)^(j + 1) * B(1, j) * Lap(B1);
10         end
11     end
12 end
```

Listing 3: Example of how to define a function that returns the value of d.

2.2 Matrices & operations

2.2.1 Matrix definition

To define a MATLAB matrix, there are a few different ways.

```
A = [[1  2]; [3,  4]];
```

Listing 4: Example of how to define a matrix.

```
A = [1  2; 3  4];
```

Listing 5: Example of how to define a matrix.

```
A = [1  2
     3  4];
```

Listing 6: Example of how to define a matrix.

```
A = [1,  2
     3,  4];
```

Listing 7: Example of how to define a matrix.

```
A = ones(n,m);
```

Listing 8: Creates matrix with only ones, of size m by n.

```
A = zeroes(n,m);
```

Listing 9: Creates matrix with only zeroes, of size m by n.

Listing 10: Creates a row vector of numbers from x to y with a step size of s.

```
A = x:s:y;
```

2.2.2 Matrix access

When accessing matrices remember **indexes start at 1**. When a matrix is accessed the index is input as A(row, column).

```
A = [1  2
     3  4];

x = A(2,1);

Output:
x = 3
```

Listing 11: Example of how a matrix can be accessed.

```
A = [1  2
     3  4];

x = A(2,:);

Output:
x = [3,4]
```

Listing 12: Example of how the row of a matrix can be accessed.

```
A = [1  2
     3  4];

x = A(:,2);

Output:
x = [2,4]
```

Listing 13: Example of how the column of a matrix can be accessed.

Matlab matrices defining and accessing (remember we start indexing 1)

2.2.3 Operations

transposing matrices To transpose a matrix, we use the ' character.

```
A = [1  2  3  4];

x = A'

Output:
x = [1
     2
     3
     4]
```

Listing 14: Example of transposing a matrix.

Matrix multiplication * vs .*
When using * in matlab be aware of whether you want to perform matrix multiplication or element by element multiplication. To use matrix multiplication you must use * and the number of columns in the first matrix must be equal to the number of rows in the second matrix.

```
A = [1  2
     3  4];
B = [5  6
     7  8];
```

```
x = A*B

Output:
x =
    19    22
    43    50
```

Listing 15: Example of matrix multiplication.

To perform element by element multiplication, use `.*` instead. Assuming both matrices have the same dimensions, element by element multiplication simply multiplies each entry of the first matrix with the corresponding entry in the second.

```
A =  [1  2];
B  = [5  7];

x = A.*B

Output:
x =
    5    14
```

Listing 16: Example of element by element multiplication.

If element by element multiplication is performed on a horizontal and a vertical matrix, each combination of the matrices will be multiplied.

```
A =  [1  2];
B  = [5
      7];

x = A.*B

Output:
x =
     5    10
     7    14
```

Listing 17: Example of element by element multiplication. Where the matrices have different dimensions.

2.3 Structure of script with functions

When using functions in MATLAB you have to define functions at the bottom of the script and then you have to make the call of the function above the definition. See the example below in listing 18:

```
    myFun(3, A, b)

    function myFun(n, A, b)
        for i = 1:n
            x = mldivide(A, [b(1); b(2) + i]);
            x
        end
    end
```

Listing 18: Example of the structure of functions

2.4 Documentation

The documentation for a specific function can be found by typing `doc functionname` and then running the section.

```
doc mldivide
```

Listing 19: Accessing the documentation for mldivide.

2.5 Printing

When printing in MATLAB there are two approaches.
The first one uses `fprintf` and can be seen below in listing 20:

```
    A1 = [9.9, 9900];
    A2 = [8.8,  7.7 ; ...
          8800, 7700];

formatSpec = 'X is %4.2f meters or %8.3f mm\n';
fprintf(formatSpec,A1,A2)

This outputs:
    X is 9.90 meters or 9900.000 mm
    X is 8.80 meters or 8800.000 mm
    X is 7.70 meters or 7700.000 mm
```

Listing 20: Example of to use *fprintf*

- %4.2f in the *formatSpec* input specifies that the first value in each line of output is a floating-point number with a field width of four digits, including two digits after the decimal point.
- %8.3f in the *formatSpec* input specifies that the second value in each line of output is a floating-point number with a field width of eight digits, including three digits after the decimal point.

- \n is a newline character.
- **Note that including a semicolon at the end of the expression suppresses and hides the output. If you want the output to be printed do not include semicolon at the end of the expression.**

The second approach is to just write the variable that you want to print **without semicolon at the end**.
An example can be seen below in listing 21:

```
    A = [
    12
    34
];

b = [5; 6];

x = A\b;
x           % Notice that there is no semicolon

This outputs:
    x = 0.2031
```

Listing 21: Second example on how to print