The TempEst Program Verification Toolset

What is it ...

TempEst is a toolset for formal verification of safety properties of programs written in the Esterel programming language. The safety properties are to be expressed in linear time temporal logic.

The TempEst toolset is made up of the following tools:

- **tl2strl:** compiler of temporal logic safety properties into Esterel programs.
- **ocreach:** obtain reachability analysis on a state machine object file.
- **octrace:** obtain execution traces leading to violation of safety properties.

How do you verify properties of an Esterel program ...

1. Write the Esterel program to be verified.
2. Write the temporal logic formulas to be checked (you'll need to read the tl2strl manual to understand the syntax of the temporal logic expressions).
3. Translate the temporal logic formulas into Esterel, using tl2strl.
4. Compile the parallel composition of the original program and the Esterel version of the formulas using the Esterel compiler.
5. Use ocreach on the state machine object file to find out whether any of the properties has been violated.
6. Or, use octrace on the state machine object file to find out about any violations of a given property and if so an execution trace that leads to the violation the property.

Details, details ...

For a more detailed view of the process, you may want to check out the papers:

- "Safety Property Verification of Esterel Programs and Applications to Telecommunications Software," Lalita J. Jagadeesan, Carlos Puchol and James Von Olnhausen. Conference on Computer-Aided Verification (CAV `95), Liege, Belgium, July 1995.
- "A Formal Approach to Reactive System Software: A Telecommunications Application in Esterel," Lalita J. Jagadeesan, Carlos Puchol and James Von Olnhausen, International Workshop on Industrial-Strength Formal Methods, Boca Raton, FL, April 1995.

(both of them available from the TempEst home page)

Legalese ...

Contact information ...

We hope you find the toolset useful. Feel free to contact us at any of the following addresses:

Carlos Puchol
Department Of Computer Sciences
The University of Texas at Austin
P.O. Box 7817
Austin, TX 78713-7817
<cpg@research.att.com>

and

Lalita J. Jagadeesan
AT&T Bell Laboratories
Room IHC 1U-332
1000 E Warrenville Rd.
Naperville, IL. 60566-7013
<lalita@research.att.com>

Two Examples:

- A Microwave Oven
- A Mutual Exclusion Problem

The Microwave Oven Implementation:

```
module Oven:

        input START, STOP, DOOR_OPEN, SECOND, MINUTE;

        output HEAT, LIGHT, BEEP;

        relation START # STOP;

        loop
                await immediate START;
                do
                        do
                                sustain HEAT
                        upto immediate [(DOOR_OPEN or STOP)];
                watching MINUTE
                timeout
                        emit BEEP;
                end;
                await tick
        end loop

        ||

        every immediate DOOR_OPEN do
                do
                        sustain LIGHT
                upto [not DOOR_OPEN]
        end every

end module
```

The Main Module:

```
module Main:

        input START, STOP, DOOR_OPEN, SECOND, MINUTE;

        output HEAT, LIGHT, BEEP, Safety0_VIOLATED,
                Safety1_VIOLATED, Safety2_VIOLATED;

        relation START # STOP;

        run Oven

        ||              % Notice how the code for checking the
                        % properties is run in parallel
        run Safety0

        ||

        run Safety1

        ||

        run Safety2

end module
```


Carlos Puchol **AT&T Bell Laboratories** **cpg@research.att.com**

The Properties to Check of the Oven Implementation:

Property Safety00:

```
%
% If the door is open, the light must be on.
%

Signals: BEEP, LIGHT, HEAT, START, MINUTE, DOOR_OPEN, STOP;

SA :=  { DOOR_OPEN -> LIGHT }

Safety00 := Always { SA }
```

Property Safety01:

```
%
% If the heat mechanism is on, the door must not be open.
% (or, the heat mechanism can't be on if the door is open.)
%

Signals: BEEP, LIGHT, HEAT, START, MINUTE, DOOR_OPEN, STOP;

SA := { HEAT -> (Not DOOR_OPEN) }

Safety01 := Always { SA }
```

Property Safety02:

```
%
% If START is pressed, then in at most one MINUTE, if the door is
% not OPENed and STOP is not pressed, the BEEP must be sounded.
%

Signals: BEEP, LIGHT, HEAT, START, MINUTE, DOOR_OPEN, STOP;

SA := { (Not (STOP | DOOR_OPEN)) -> BEEP } RespondsTo { START } In 1 MINUTE

Safety02 := Always { SA }
```

http://www.cs.utexas.edu/users/cpg/TempEst

Carlos Puchol **AT&T Bell Laboratories** **cpg@research.att.com**

Checking the Oven Implementation

Compilation:

```
$ esterel -v3 -oc -show -B OvenGood OvenGood.strl Safety0.strl
Safety1.strl Safety2.strl
--- lcoc: created states: 7     analysed states: 7
```

Checking for Safety0:

```
$ octrace OvenGood.oc Safety0_VIOLATED
Signal Safety0_VIOLATED is never emitted in file esterel.oc.
```

Checking for Safety1:

$ octrace OvenGood.oc Safety1_VIOLATED
Signal Safety1_VIOLATED is never emitted in file esterel.oc.

Checking for Safety2:

$ octrace OvenGood.oc Safety2_VIOLATED
Signal Safety2_VIOLATED is never emitted in file esterel.oc.

Next we want to modify the original Oven to introduce a bug.

Carlos Puchol **AT&T Bell Laboratories** **cpg@research.att.com**

The BROKEN Microwave Oven Implementation

>>>>> Notice the 3 bugs injected! <<<<<

```
module Oven:

        input START, STOP, DOOR_OPEN, SECOND, MINUTE;

        output HEAT, LIGHT, BEEP;

        relation START # STOP;

        loop
                await immediate START;
                do
                        do
                                sustain HEAT
                        upto immediate [(DOOR_OPEN or STOP)];
                watching MINUTE;
                emit HEAT;              % Bug #1: HEAT is applied still once more!
                await tick;
                emit BEEP;              % Bug #2: it BEEPs too late!
        end loop

        ||
                                        % Bug #3: LIGHT should be on immediately
        every DOOR_OPEN do              %          after the DOOR_OPEN.
                do
                        sustain LIGHT
                upto [not DOOR_OPEN]
        end every

end module
```

The main module is still the same.

Next we take a look at the result of the analysis for the broken Oven implementation.

http://www.cs.utexas.edu/users/cpg/TempEst

Carlos Puchol **AT&T Bell Laboratories** **cpg@research.att.com**

The Result of the Analysis for the BROKEN Oven

Compilation:

```
$ esterel -v3 -oc -show -B OvenBaad OvenBaad.strl Safety0.strl Safety1.strl Safety2.strl
--- lcoc: created states: 14    analysed states: 14
```

Checking for Safety0:

```
$ octrace OvenBaad.oc Safety0_VIOLATED
Signal Safety0_VIOLATED is emitted in 1 states.
The following table should be read from bottom to top.
Trace for signal: Safety0_VIOLATED (8):
----------------------------------------------------------------------------|
Curr Next |                              |                                  |
St.  St.  | Input event                  | Output event                     |
----------------------------------------------------------------------------|
1    2    | START DOOR_OPEN              |  HEAT Safety0_VIOLATED           |
          |                              | Safety1_VIOLATED                 |
----------------------------------------------------------------------------|
```

Checking for Safety1:

```
$ octrace OvenBaad.oc Safety1_VIOLATED
Signal Safety1_VIOLATED is emitted in 13 states.
The following table should be read from bottom to top.
Trace for signal: Safety1_VIOLATED (9):
----------------------------------------------------------------------------|
Curr Next |                              |                                  |
St.  St.  | Input event                  | Output event                     |
----------------------------------------------------------------------------|
1    2    | START DOOR_OPEN              |  HEAT Safety0_VIOLATED           |
          |                              | Safety1_VIOLATED                 |
----------------------------------------------------------------------------|
```

Checking for Safety2:

```
$ octrace OvenBaad.oc Safety2_VIOLATED
Signal Safety2_VIOLATED is emitted in 1 states.
The following table should be read from bottom to top.
Trace for signal: Safety2_VIOLATED (10):
----------------------------------------------------------------------------|
Curr Next |                              |                                  |
St.  St.  | Input event                  | Output event                     |
----------------------------------------------------------------------------|
3    2    | MINUTE                       |  HEAT Safety2_VIOLATED           |
----------------------------------------------------------------------------|
1    3    | START                        |  HEAT                            |
----------------------------------------------------------------------------|
```


Carlos Puchol **AT&T Bell Laboratories** **cpg@research.att.com**

http://www.cs.utexas.edu/users/cpg/TempEst
Carlos Puchol **AT&T Bell Laboratories** **cpg@research.att.com**

The Mutex Implementation: Two Processes

```
module ProcessA:

        input A_REQUEST, A_DONE, B_IN_CS, B_READY;

        output A_IN_CS, A_READY;

        loop
                await A_REQUEST;
                emit A_READY;
                await immediate [not B_READY];  % This line only in process A to break symmetry
                await immediate [not B_IN_CS];
                trap RESTART in
                        sustain A_IN_CS
                        ||
                        [
                                await A_DONE;
                                exit RESTART;
                        ]
                end trap;
                await tick
        end loop

end module
```

(Process B is nearly identical to Process A)

The main module:

```
module MutexMain:

        input A_REQUEST, A_DONE, B_REQUEST, B_DONE;

        output  A_IN_CS, B_IN_CS,Safety01_VIOLATED,Safety02_VIOLATED,SA1, SA2, SA3;

        signal A_READY, B_READY in

                run ProcessA

        ||

                run ProcessB

        ||

                run Safety01

        ||

                run Safety02

        end signal
end module
```


Carlos Puchol **AT&T Bell Laboratories** **cpg@research.att.com**

The Properties to Check of the Mutex Implementation

Property Safety01:

```
%
% Mutual exclusion Property
%

Signals: A_IN_CS, B_IN_CS;

SA := { Not (A_IN_CS & B_IN_CS) }

Safety01 := Always { SA }
```

Property Safety02:

```
%
% misc. properties
%

Signals: A_IN_CS, B_IN_CS, A_READY, B_READY;

SA1 := { ( A_READY & (Not B_READY) & (Not B_IN_CS) ) -> A_IN_CS  }

SA2 := { ( B_READY & (Not A_READY) & (Not A_IN_CS)) -> B_IN_CS }

SA3 := { ( A_READY & B_READY) -> B_IN_CS }

Safety02 := Always { SA1 & SA2 & SA3 }
```

The Result of the Analysis for the Mutex Implementation

Compilation:

```
$ esterel -v3 -oc -show -B MutexMain1 MutexMain1.strl ProcessA.str ProcessB.strl Safety01.strl
Safety02.strl
--- lcoc: created states: 12    analysed states: 12
```

Checking for Safety01:

```
$ octrace MutexMain1.oc Safety01_VIOLATED
Signal Safety01_VIOLATED is never emitted in file MutexMain1.oc.
```

Checking for Safety02:

```
$ octrace MutexMain1.oc Safety02_VIOLATED
Signal Safety02_VIOLATED is never emitted in file MutexMain1.oc.
```

Next we want to modify the original Oven to introduce a bug.

http://www.cs.utexas.edu/users/cpg/TempEst
Carlos Puchol **AT&T Bell Laboratories** **cpg@research.att.com**

The BROKEN Mutex Implementation:

Process A is Independent of B

```
module ProcessA:

        input A_REQUEST, A_DONE, B_IN_CS, B_READY;

        output A_IN_CS, A_READY;

        loop
                await A_REQUEST;
                emit A_READY;
                % await immediate [not B_READY];   % The bug: A is now independent
                % await immediate [not B_IN_CS];   %          of process B.
                trap RESTART in
                        sustain A_IN_CS
                        ||
                        [
                                await A_DONE;
                                exit RESTART;
                        ]
                end trap;
                await tick
        end loop

end module
```

The main module and Process B remain identical.

Carlos Puchol **AT&T Bell Laboratories** **cpg@research.att.com**

Analysis of the BROKEN Mutex Implementation:

Compilation:

```
$ esterel -v3 -oc -show -B MutexMain2 MutexMain2.strl ProcessA-bad.strl ProcessB.strl Safety01.strl
Safety02.strl
--- lcoc: created states: 13    analysed states: 13
```

Checking for Safety01:

```
$ octrace MutexMain2.oc Safety01_VIOLATED
Signal Safety01_VIOLATED is emitted in 2 states.
The following table should be read from bottom to top.
Trace for signal: Safety01_VIOLATED (6):
-----------------------------------------------------------------------------|
Curr Next |                              |                                   |
St.  St.  | Input event                  | Output event                      |
-----------------------------------------------------------------------------|
4    7    | B_DONE A_REQUEST             |  A_IN_CS B_IN_CS                  |
          |                              | Safety01_VIOLATED                 |
-----------------------------------------------------------------------------|
2    4    | B_REQUEST                    |  B_IN_CS                          |
-----------------------------------------------------------------------------|
1    2    |                              |                                   |
-----------------------------------------------------------------------------|
```

Checking for Safety02:

```
$ octrace MutexMain2.oc Safety02_VIOLATED
Signal Safety02_VIOLATED is emitted in 1 states.
The following table should be read from bottom to top.
Trace for signal: Safety02_VIOLATED (7):
-----------------------------------------------------------------------------|
Curr Next |                              |                                   |
St.  St.  | Input event                  | Output event                      |
-----------------------------------------------------------------------------|
2    3    | B_REQUEST A_REQUEST          |  A_IN_CS Safety02_VIOLATED        |
-----------------------------------------------------------------------------|
1    2    |                              |                                   |
-----------------------------------------------------------------------------|
```

http://www.cs.utexas.edu/users/cpg/TempEst

Carlos Puchol **AT&T Bell Laboratories** **cpg@research.att.com**