SimGrid 101

Getting Started with the SimGrid Model-Checker

Da SimGrid Team

June 13, 2012

About this Presentation

Goals and Contents

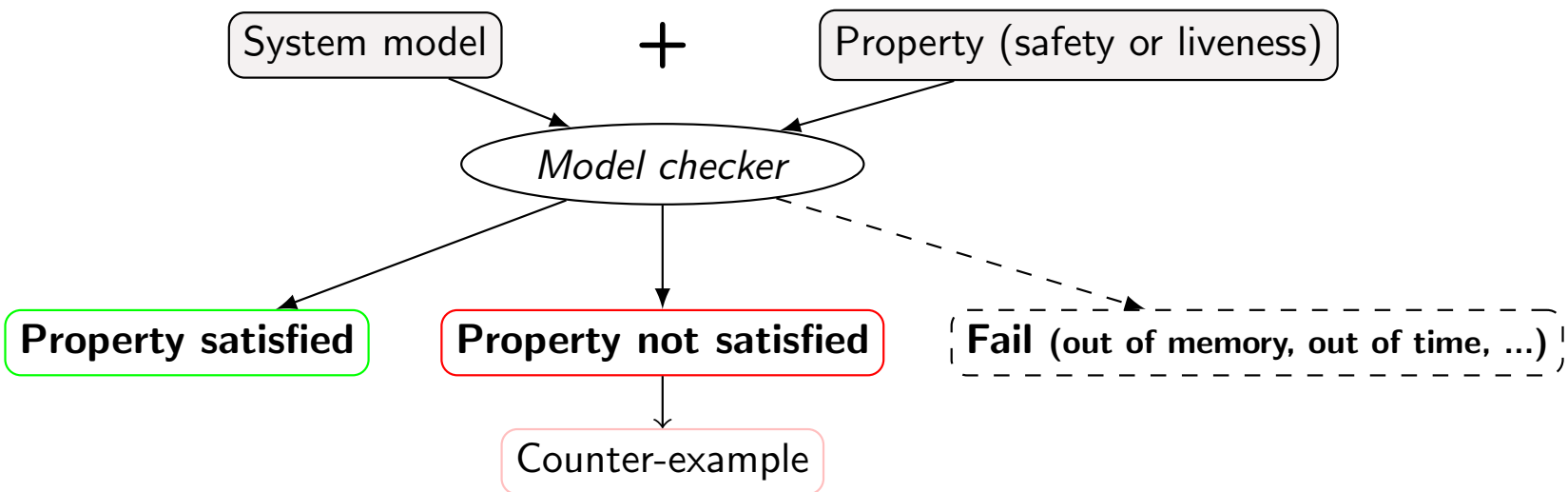
- Understanding the basics of Model checking
- Running SimGrid as a Model Checker
- Analysing the counter-example traces produced

The SimGrid 101 serie

- This is part of a serie of presentations introducing various aspects of SimGrid
- SimGrid 101. Introduction to the SimGrid Scientific Project
- SimGrid User 101. Practical introduction to SimGrid and MSG
- SimGrid User::Platform 101. Defining platforms and experiments in SimGrid
- SimGrid User::SimDag 101. Practical introduction to the use of SimDag
- SimGrid User::Visualization 101. Visualization of SimGrid simulation results
- SimGrid User::SMPI 101. Simulation MPI applications in practice
- SimGrid User::Model-checking 101. Formal Verification of SimGrid programs
- SimGrid Internal::Models. The Platform Models underlying SimGrid
- SimGrid Internal::Kernel. Under the Hood of SimGrid
- Retrieve them from `http://simgrid.gforge.inria.fr/101`

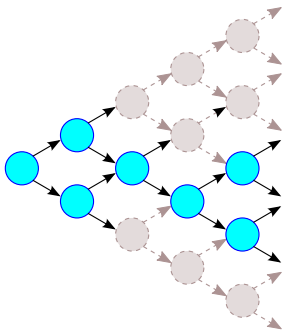
Model checking

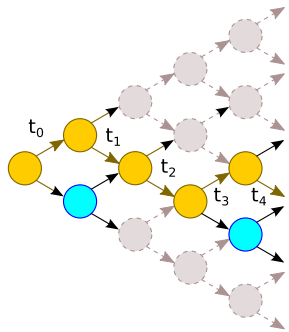
- Automated verification method (hardware or software)
- Checks whether a given model of a system satisfies a property
- Gives a counter-example in case of violation of the property



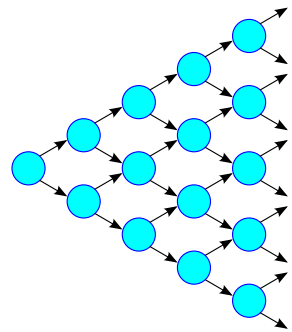
Simulation vs. Model Checking

- Simulation explores **one possible execution** of the program according to the features/limitations of the platform
- Model checking explores **all possible executions** of the program





State space with simulation



State space with model checking

- Simulation and model checking are complementary :
  - Simulation for performance evaluation
  - Model Checking for the **verification of execution properties**
  - Both run automatically

Model checking implementation with SimGrid

Step 1: Express the property that you want to assess

Step 2: Instrument your code with MC primitives

Step 3: Compile and run with the proper MC configuration options

Step 4: Analyze the produced traces

1. Express the property that you want to assess

Safety Property

- *"A given bad behavior never occurs"*
- Ex : no deadlock, $x \neq 0 \rightarrow$ **boolean expression**
- Work on **all states** separatly
- **Assertion** on each state of the execution

Liveness property

- *"An expected behavior will happen in all cases"*
- Ex : Any process that asks a resource will obtain it eventually
- Work on **execution path**
- **Temporal logic formula** (LTL, CTL, ...)

Only safety properties can be verified for now...

2. Instrument your code with MC primitives

Very few changes are mandatory at source level

- Include header file of model checker:

```
#include <simgrid/modelchecker.h>
```

- Add verification of safety property with assertion in source code:

```
void MC_assert(<boolean expression of the property>)
```

- Beside of that, you keep your MSG code inchanged

3. Configure, compile and execute

At configuration time

- Set the `enable_model-checking` option of cmake:
- Either in ccmake, or on the command line:

```
cmake -Denable_model-checking=ON ./
```

Run your code

- Model checking with reduction:

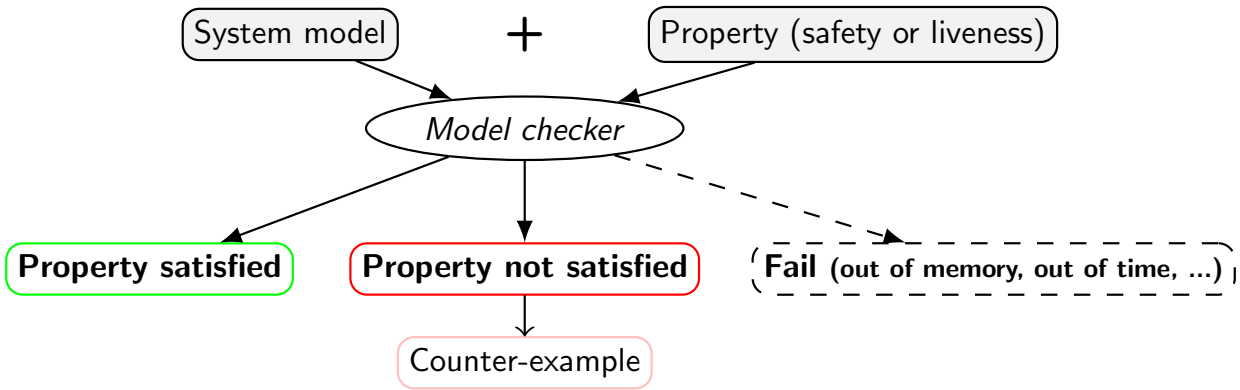
```
./<executable_application> --cfg=model-check:safety
```

- Disabling the reduction (you don't want to do it):

```
./<executable_application> --cfg=model-check:raw
```

The exact name of these options may be cleaned up in the future

4. Analyze the produced traces



Execution results

- If the property is satisfied, normal exit
- If the property gets violated, produces **counter-example** (execution trace)

Model Checking Statistics

- (produced in any case; you want to keep an eye on them)
- Expanded states: number of states created
- Visited states: number of states created and checked
- Executed transitions: number of enabled transitions executed

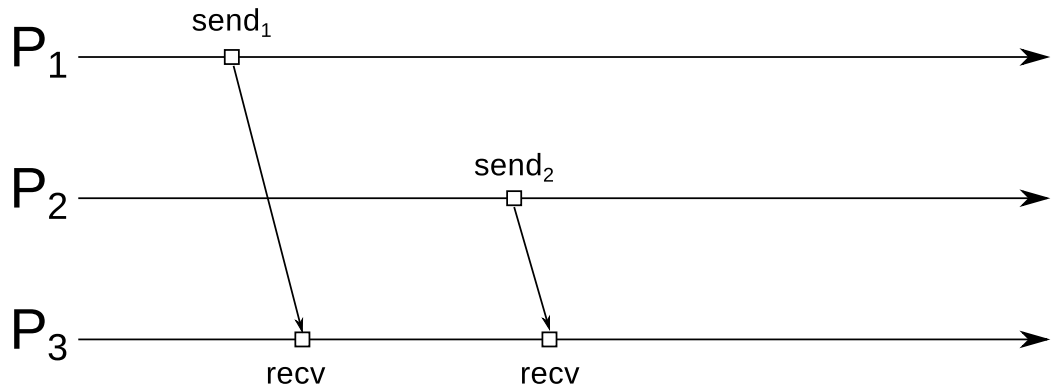
Example: Out of order receive (bugged1.c – 1/3)

- Two processes send a message to a third one
- The receiver expects the message to be in order
- This may happen...

```
P1() {
  Send(1,P3);
}

P2() {
  Send(2,P3);
}

P3() {
  x=RecvAny();
  y=RecvAny();
  ASSERT(x<y);
}
```



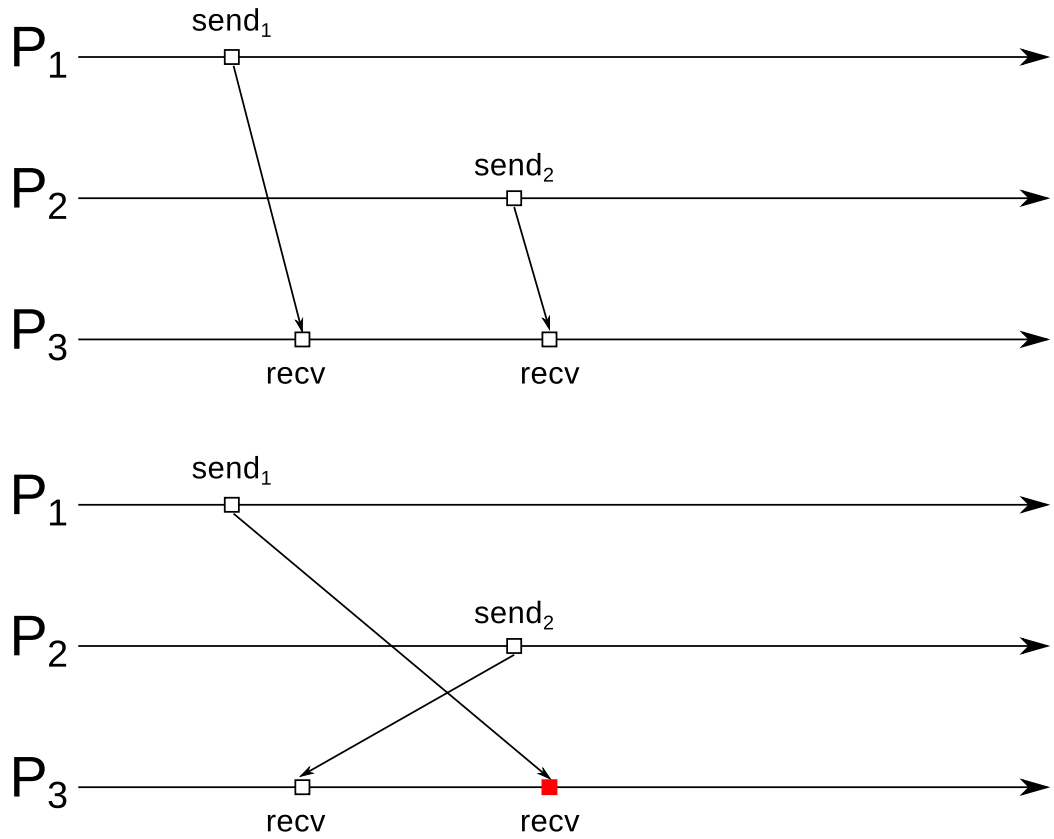
Example: Out of order receive (bugged1.c – 1/3)

- Two processes send a message to a third one
- The receiver expects the message to be in order
- This may happen... or not

```
P1() {
  Send(1,P3);
}

P2() {
  Send(2,P3);
}

P3() {
  x=RecvAny();
  y=RecvAny();
  ASSERT(x<y);
}
```



Example: Out of order receive (2/3)

int recver(int argc,char**argv){

```
m_task_t task = NULL;
MSG_task_receive(&task, "mymailbox");
MSG_task_destroy(task); task = NULL;
MSG_task_receive(&task, "mymailbox");

MC_assert(atoi(MSG_task_get_name(task)) == 2);
return 0; }
```

int sender(int argc,char**argv){

```
m_task_t t = MSG_task_create(argv[1],0,10,NULL);
MSG_task_send(t, "mymailbox");
return 0;        }
```

int main(int argc,char**argv) {

```
MSG_global_init(&argc, argv);
MSG_create_environment("platform.xml");
MSG_function_register("recver", recver);
MSG_function_register("sender", sender);
MSG_launch_application("deployment.xml");
MSG_main();
MSG_clean();
return 0;
```

Deployment File

```
<platform version="3">
  <process host="P1" function="sender">
    <argument value="1"/>
  </process>
  <process host="P2" function="sender">
    <argument value="2"/>
  </process>
  <process host="P3" function="recver"/>
</platform>
```

Platform File

```
<platform version="3">
  <AS  id="AS0"  routing="Full">
    <host id="P1" power="1"/>
    <host id="P2" power="1"/>
    <host id="P3" power="1"/>
  </AS>
</platform>
```

Example: Out of order receive (3/3)

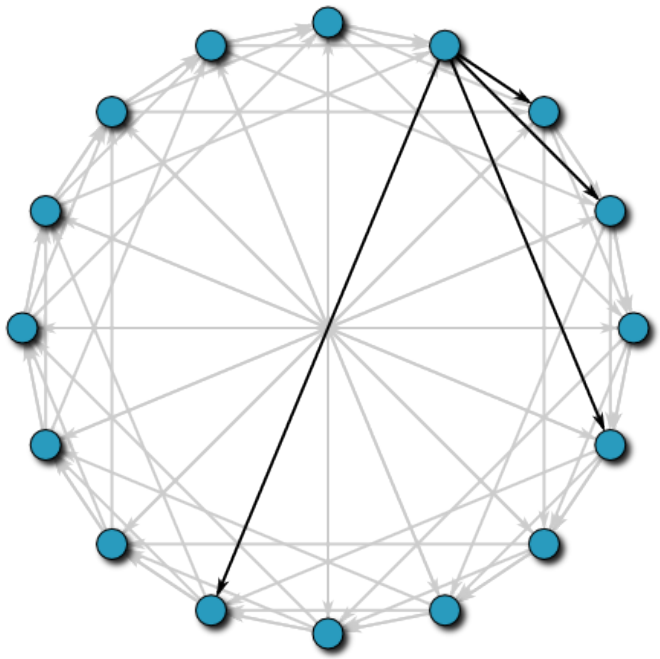
```
**************************
*** PROPERTY NOT VALID ***
**************************

Counter-example execution trace:
[(1)server] iRecv (dst=server, buff=(verbose only), size=(verbose only))
[(3)client] iSend (src=client, buff=(verbose only), size=(verbose only))
[(1)server] Wait (comm=(verbose only) [(3)client -> (1)server])
[(1)server] iRecv (dst=server, buff=(verbose only), size=(verbose only))
[(2)client] iSend (src=client, buff=(verbose only), size=(verbose only))
[(1)server] Wait (comm=(verbose only) [(2)client -> (1)server])

Expanded states = 43
Visited states = 114
Executed transitions = 101
Expanded / Visited = 2.651163
```

Chord Experiments

Chord P2P DHT protocol



SimGrid Implementation

- 500 lines of C (MSG interface)
- Suffered of bug in big instances
- Unable to spot it precisely

SimGrid MC with two Nodes

- DFS: 15600 states - 24s
- DPOR: 478 states - 1s
- Simple Counter-example!
- One line fix

Conclusion

Model-Checking in SimGrid

- This works already (although a bit fresh yet)
- This may help you to hunt hard bugs down
- You should test it! (feedback welcomed)

Ongoing Work: Verifying Liveness Properties

Why is it harder?

- Liveness: *"An expected behavior will happen in all cases"*
- Reason about the execution path, not only locally on each steps

Modified Steps

- Step 2 : express liveness property with LTL formula
  - ex: $G(r \rightarrow Fcs)$ (r = critical section requested, c = critical section granted)
- Step 3 : instrument source code for liveness verification
  - Atomic propositions of LTL formula correspond to global variables
- Step 4 : run and compile with configuration options
  - `cmake -Denable_model-checking=ON ./`
  - `--cfg=model-check:2`

This will work . . . soon