Intrepid
a scriptable and cloud-ready SMT Model Checker

Roberto Bruttomesso

Intro

- Circuit-based symbolic Model Checker
  - Represent a model and a property with a circuit
  - Translate the circuit and its evolution over time as formulas
  - Ask a solver for satisfiability

- BDD, SAT, **SMT**
  - Microsoft Z3 solver

- Beyond Booleans types
  - Better preservation of the structure of the model

Motivation

- Quote from `fmics.inria.fr` :

  > *"[…] the use of formal methods in the industry is still quite limited. Apparently, major reasons for that are the notational difficulty of most formal methods available nowdays and the lack of integration between them. Notational complexity is often a deterrent to the use of formal methods stronger than the advantages of such methods […]"*

- Recurring problems with model checkers (in my experience):
  - Language issue (understand the tool input, reparse cex)
  - Libraries
  - Lack of interactivity and scriptability
  - Multiple targets

Intrepid models: python scripts

```
10    def myfunc(self):
11        context = intrepyd.Context()
12        context.mk_m      You, seconds ago • Uncommitted changes
13
14
15
16
17
18
19
20
21
22
23
```

- mk_minus
- mk_mod
- mk_mul
- mk_bmc
- mk_implies
- mk_number
- mk_simulator
- mk_assumption
- mk_optimizing_bmc

```
mk_minus: (x, name=None) -> Any
```

Creates the term -x


```
13 ctx.set_latch_init_next(latch2,\
14                          ctx.mk_true(),\
15                          ctx.mk_not(clk))
```

Libraries

```
1 def mk_clock(ctx, name):
2     bool_t = ctx.mk_boolean_type()
3     i = ctx.mk_input(name + '_input', bool_t)
4     first = ctx.mk_latch(name + '_first', bool_t)
5     ctx.set_latch_init_next(first, ctx.mk_true(), ctx.mk_false())
6     inv = ctx.mk_latch(name + '_inv', bool_t)
7     clk = ctx.mk_ite(first, i, inv, name=name)
8     ctx.set_latch_init_next(inv, ctx.mk_true(), ctx.mk_not(clk))
9     return clk, i
```

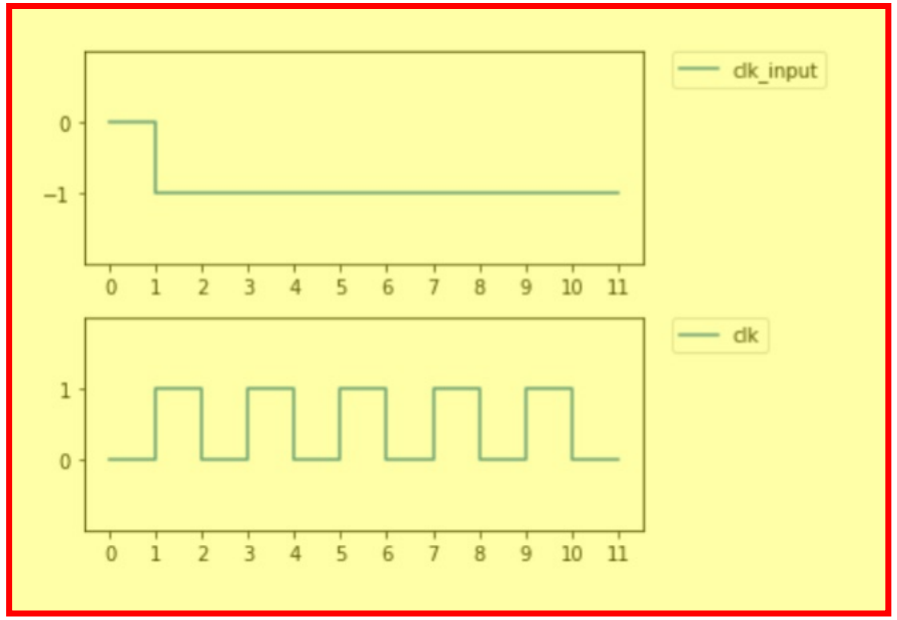
...

```
100 ctx = Context()
101 clk1, i1 = mk_clock(ctx, ‘my_clock’)
102 clk2, i2 = mk_clock(ctx, ‘your_clock’)
```

Simulating a model

```
1 from intrepyd import Context
2 from intrepyd.components.eda
        import mk_clock
3 from intrepyd.plots
        import plot_trace_dataframe
4
5 ctx = Context()
6 clk, clk_input = mk_clock(ctx, 'clk')
7 simulator = ctx.mk_simulator()
8 simulator.add_watch(clk)
9 simulator.add_watch(clk_input)
10 trace = ctx.mk_trace()
11 trace.set_value(clk_input, 0, 'F')
12 simulator.simulate(trace, 10)
13 df = trace.get_as_dataframe(ctx.net2name)
14 print(df)
15 plot_trace_dataframe(df)
```

	0	1	2	3	4	5	6	7	8	9	10
clk_in	F	?	?	?	?	?	?	?	?	?	?
clk	F	T	F	T	F	T	F	T	F	T	F



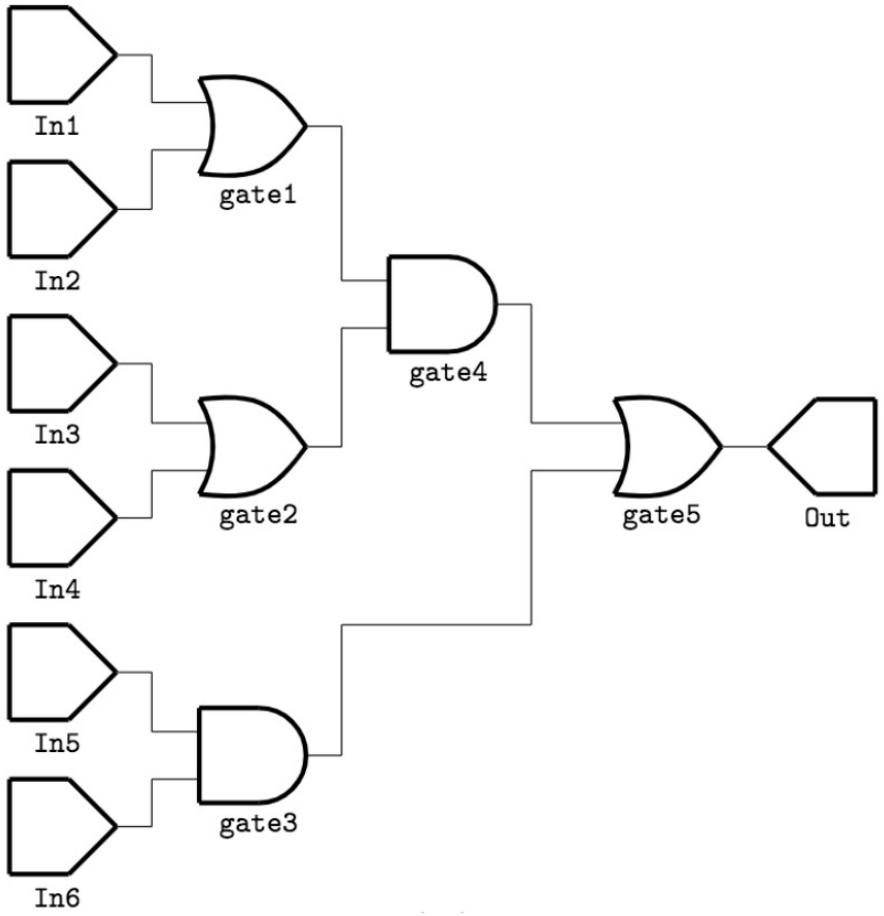
Checking properties: engines

- **Target**: a Boolean signal in the constructed circuit
- Engines can check for **target reachability**, i.e., if they ever get value 1
- Engines are multi-targets:
  - can be fed with N targets
- When a target T is found reachable
  - the search is “paused”
  - its counterexample (trace) can be retrieved
  - the search can be resumed (without T)

Checking properties: engines

- Bounded Model Checker (incomplete)
  - with or without Temporal Induction (complete)
- Optimizing Bounded Model Checker (incomplete)
  - attempts to find a trace that satisfies the most targets
  - relies on z3 optimization algorithms
- Backward Reachability (complete)
  - our own algorithm based on preimage computation and QE

Automated Test Generation MC/DC

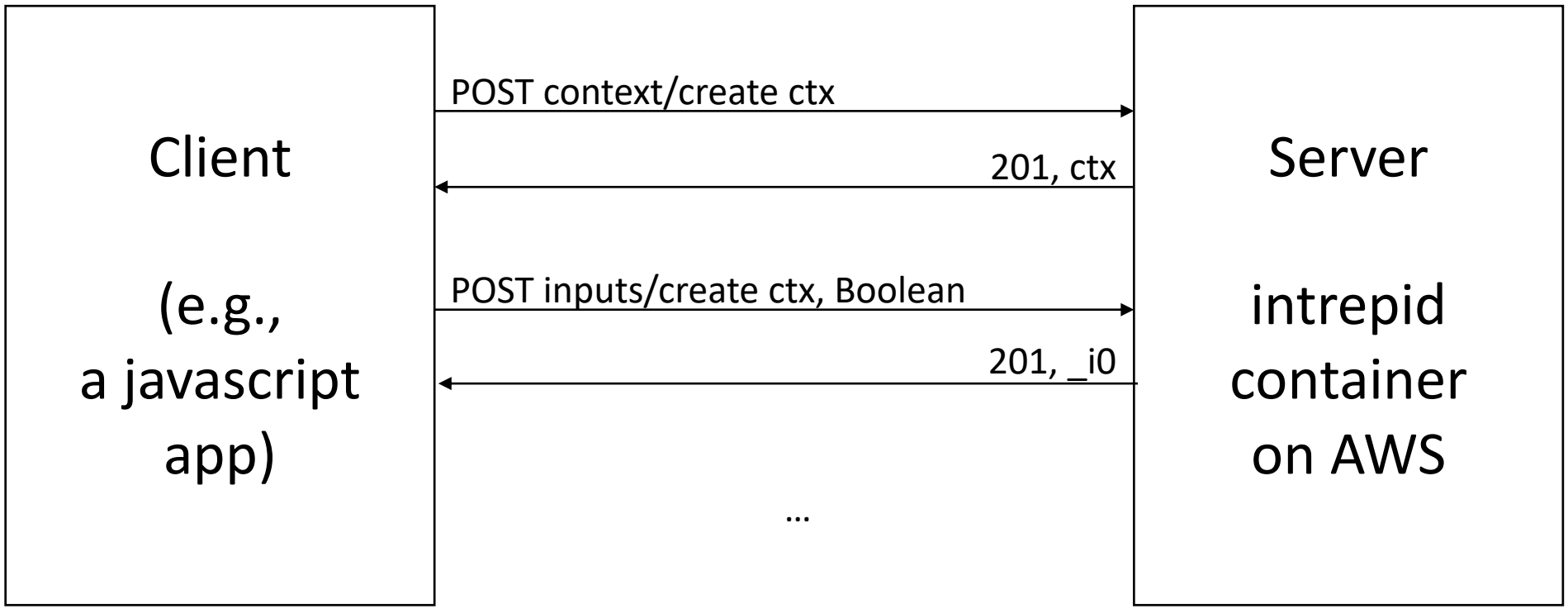


	In1	In2	In3	In4	In5	In6	Out
0	F	F	T	F	F	F	F
1	T	F	T	F	F	F	T
2	F	T	T	F	F	F	T
3	T	F	F	F	F	F	F
4	T	F	F	T	F	F	T
5	F	F	F	F	F	T	F
6	F	F	F	F	T	T	T
7	F	F	F	F	T	F	F

REST API

- Intrepid also comes as a docker container featuring a REST API

```
docker run -p 8000:8000 -d robertobruttomesso/intrepid
```



Conclusion

- Intrepid, a model checker with python and REST API
- Related work
  - check the paper
- A more hands-on video
  - https://youtu.be/n-0Y_iJqkqY