NOZOMI NETWORKS

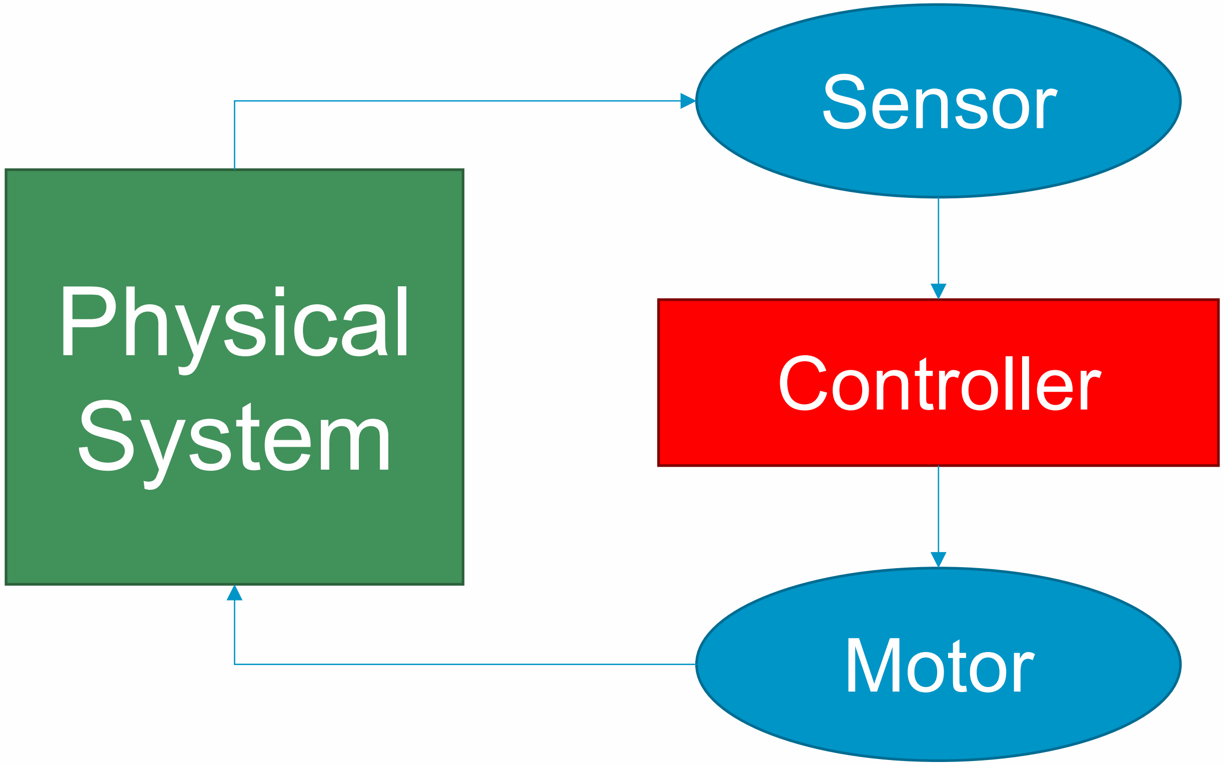
Roberto Bruttomesso

Intrepid: an SMT-based Model Checker for Control Engineering and Industrial Automation

SMT 2019

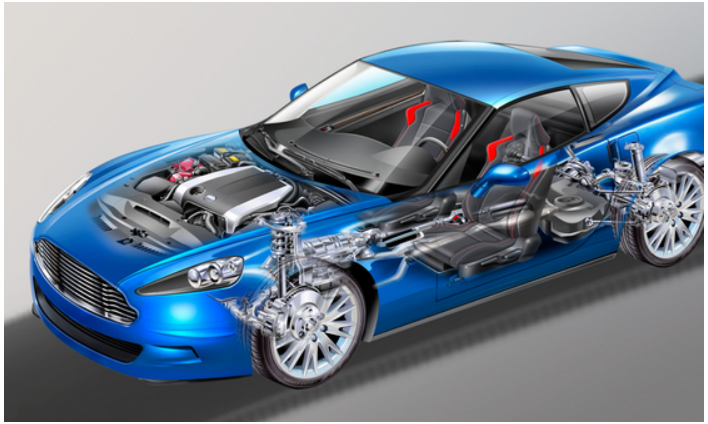

Control Engineering

Control Engineering



Avionics & Automotive

- Strict software development process, encoded in standards (e.g., DO-178C)
- Requirement-centric process
- Derive software from reqs.
- Provide tests to witness that requirements are met



Avionics & Automotive

Google

boeing 737 max

boeing 737 max **8**
boeing 737 max **crashes**
boeing 737 max **design flaws**
boeing 737 max **news**
boeing 737 max **grounding**
boeing 737 max **explained**
boeing 737 max **flight simulator**
boeing 737 max **pilot training**

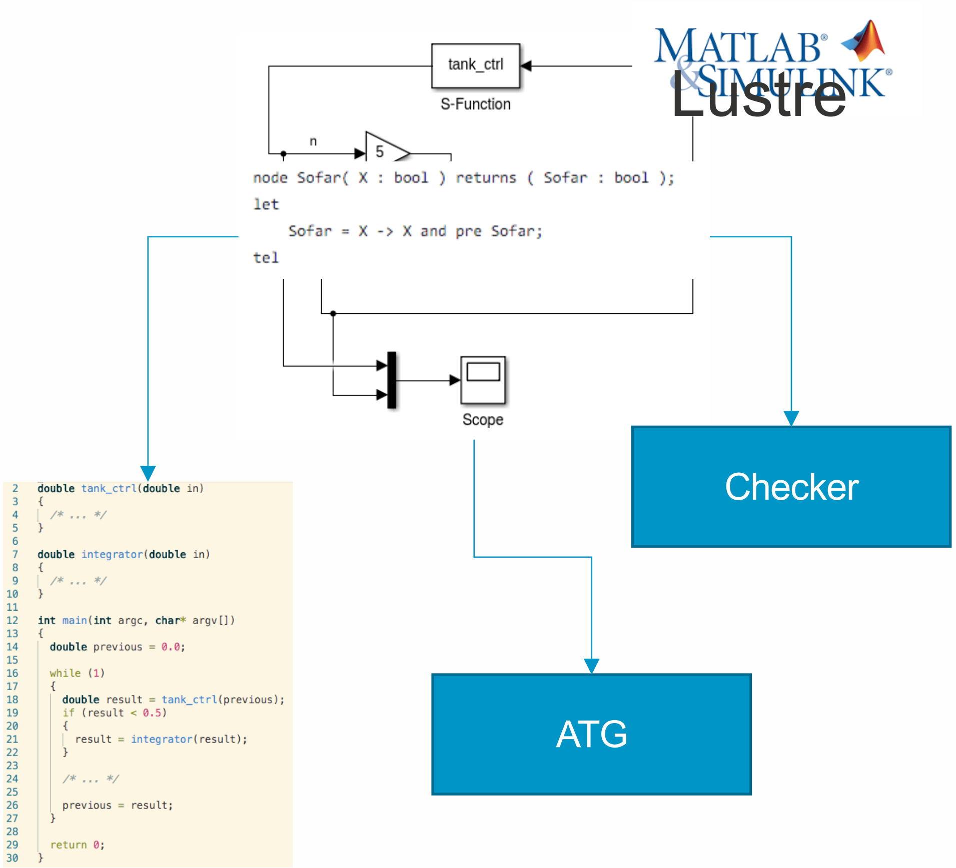
The Maneuvering Characteristics Augmentation System (MCAS), a newly introduced automated flight control on the MAX, was suspected of forcing both aircraft into a dive due to erroneous data from an angle of attack (AoA) sensor. The airplane flight manual contained no description of MCAS, and pilots had no knowledge of the system until the Lion Air crash. Aviation engineers criticized Boeing's safety analysis of MCAS: the system used only one of two AoA sensors, creating a single point of failure; it could move the tail beyond operating limits specified in the certification; and it could repeatedly override pilot control and push down the airplane nose. After the accidents Boeing disclosed that a cockpit indicator to warn of AoA sensor failure was inoperative. Boeing insisted the aircraft was safe, but admitted MCAS was a factor in both accidents.

Search **Bloomberg**

Technology

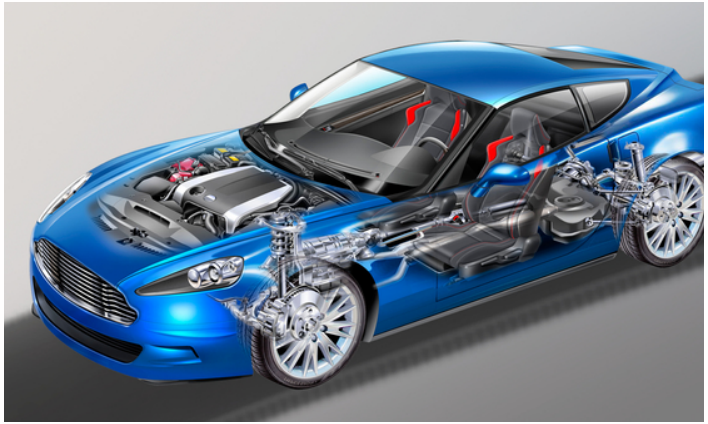
Boeing's 737 Max Software Outsourced to $9-an-Hour Engineers

Avionics & Automotive



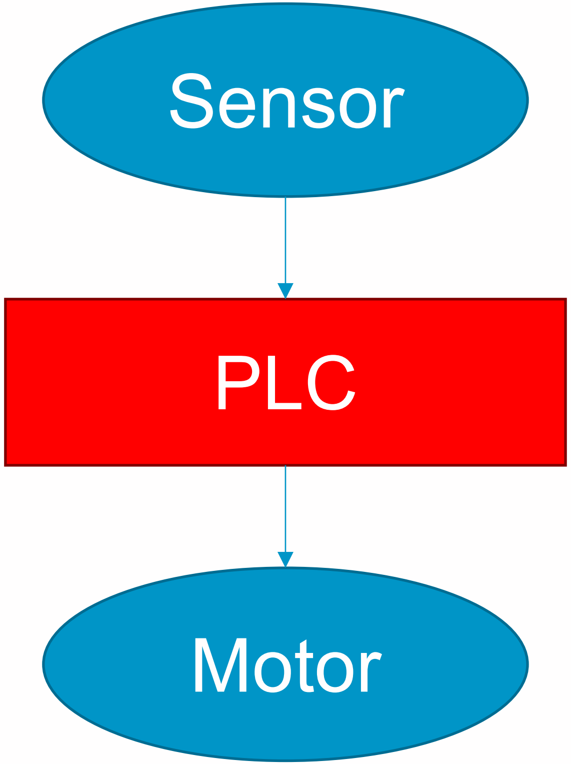

```
node Sofar( X : bool ) returns ( Sofar : bool );
let
    Sofar = X -> X and pre Sofar;
tel
```

```c
double tank_ctrl(double in)
{
  /* ... */
}

double integrator(double in)
{
  /* ... */
}

int main(int argc, char* argv[])
{
  double previous = 0.0;

  while (1)
  {
    double result = tank_ctrl(previous);
    if (result < 0.5)
    {
      result = integrator(result);
    }

    /* ... */

    previous = result;
  }

  return 0;
}
```



Industrial Automation

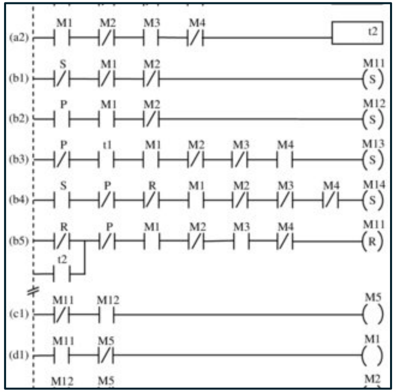
Industrial Automation



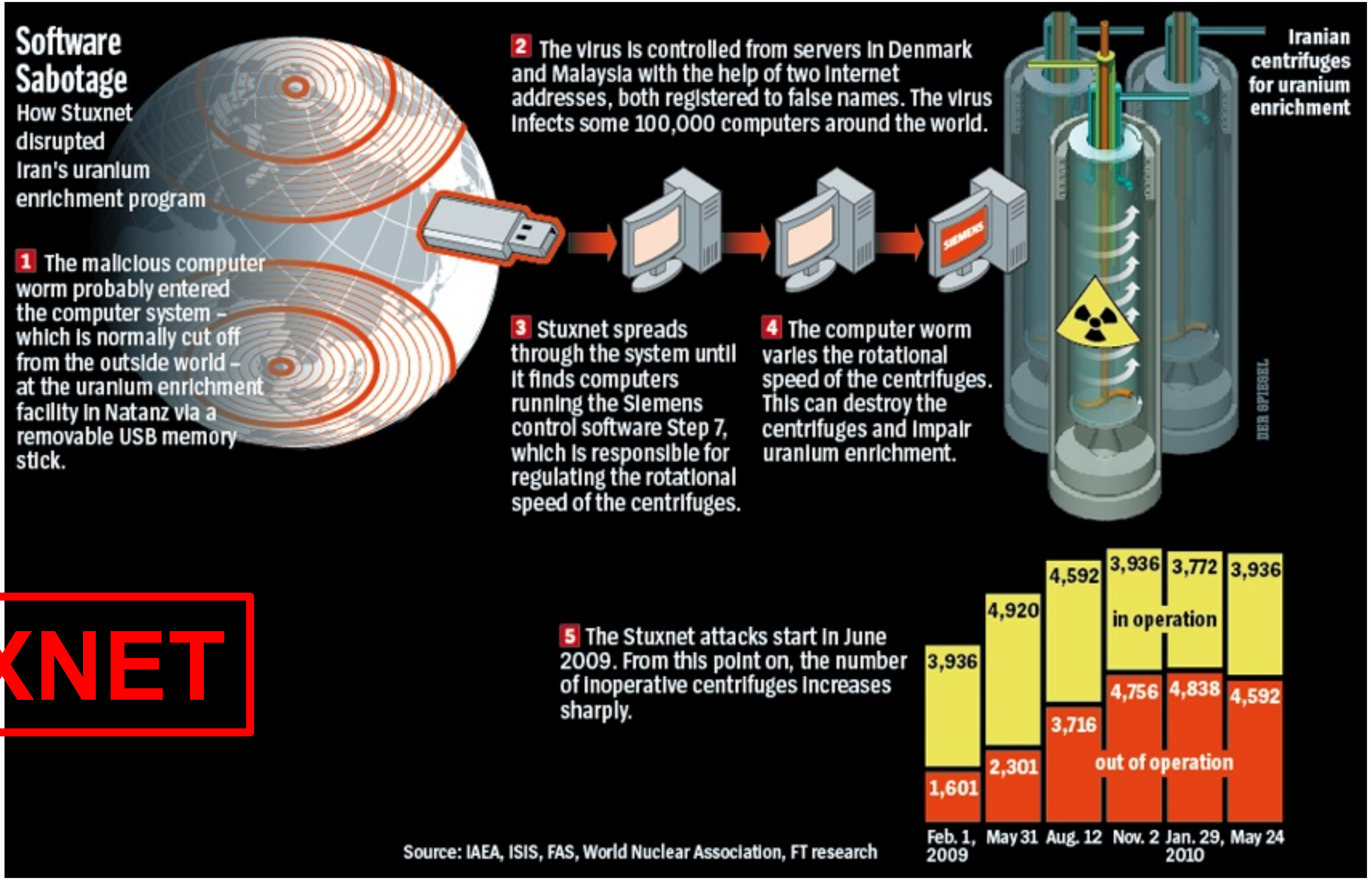
Industrial Automation

- IEC-61131 defines 5 programming languages for PLCs
- Two textual (ST, IL)
- Two graphical (FBD, LD)
- A “mixed” (SFC)

```
    END_IF;
Setpoint_IN_STAGE_1_FAILED:
    (* During 'STAGE_1_FAILED': '<S1>:119' *)
    IF (stage3_sensor <= 0) OR (stage2_sensor <= 0) THEN
        (* Transition: '(<S1>:150' *)
        (* Transition: '(<S1>:152' *)
        IF stage2_sensor > 0 THEN
            (* Transition: '<S1>:155' *)
            is_c2_Setpoint := Setpoint_IN_STAGES_1_3_FAILED;
            (* Entry 'STAGES_1_3_FAILED': '<S1>:120' *)
            rtb_stage1_setpoint := L0;
            rtb_stage2_setpoint := L0 - overall_target;
            distributed_target := rtb_stage2_setpoint;
        ELSE
            (* Transition: '<S1>:154' *)
            IF stage3_sensor > 0 THEN
                (* Transition: '<S1>:159' *)
                is_c2_Setpoint := Setpoint_IN_STAGES_1_2_FAILED;
                (* Entry 'STAGES_1_2_FAILED': '<S1>:121' *)
                rtb_stage1_setpoint := L0;
                rtb_stage2_setpoint := L0;
                distributed_target := L0 - overall_target;
            ELSE
                guard_0 := TRUE;
            END_IF;
        END_IF;
    ELSE
        guard_0 := TRUE;
    END_IF;
```

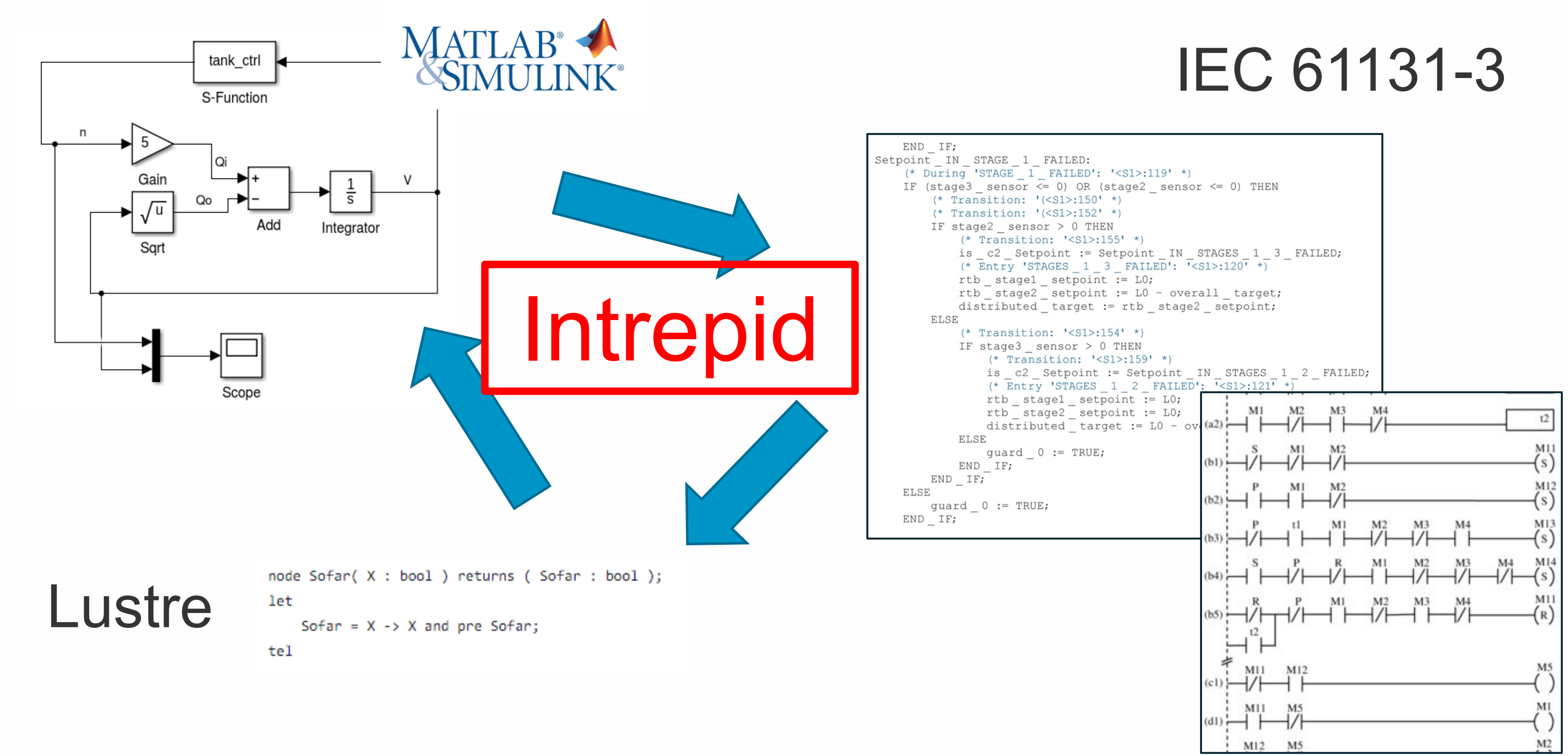


Industrial Automation



STUXNET

Control Engineering Languages



Control Engineering Languages

- Types
  - Booleans
  - Signed Integers (SINT, INT, …)
  - Unsigned Integers (USINT, UINT, …)
  - Floats (REAL, LREAL)

- Semantics of the above:
  - Fixed-width
  - Discrete evolution of memory values

Intrepid intro

Intrepid’s guiding principles

- Fast simulation
- Bit-precise
- Scriptable
- Parsing real-world languages

Intrepid: a model-checking library

- Backend: C++ engine (intrepid)
  - State representation (SMT formulas in Z3)
  - State exploration (Satisf. and QE calls to Z3)
  - Exposes a C API

- Python API (intrepyd)
  - Wraps the C API, and provides OO Python API
  - Retains efficiency, but provides flexibility and fun

Intrepid’s input language

- There is no input language: you write benchmarks directly in Python

```python
def mkFunc1(ctx, in1, in2):
    return ctx.mk_and(in1, in2)

def mkCircuit(ctx):
    boolType = ctx.mk_boolean_type()
    in1 = ctx.mk_input('in1', boolType)
    in2 = ctx.mk_input('in2', boolType)
    in3 = ctx.mk_input('in3', boolType)
    myCirc1 = mkFunc1(ctx, in1, in2)
    myCirc2 = mkFunc1(ctx, in1, in3)
    myCirc3 = mkFunc1(ctx, myCirc1, myCirc2)
```

example1.py

```python
from intrepyd import context
import example1

def main():
    ctx = context.Context()
    myCirc = example1.mkCircuit(ctx)
    # do other interesting stuff here

if __name__ == "__main__":
    main()
```

example2.py

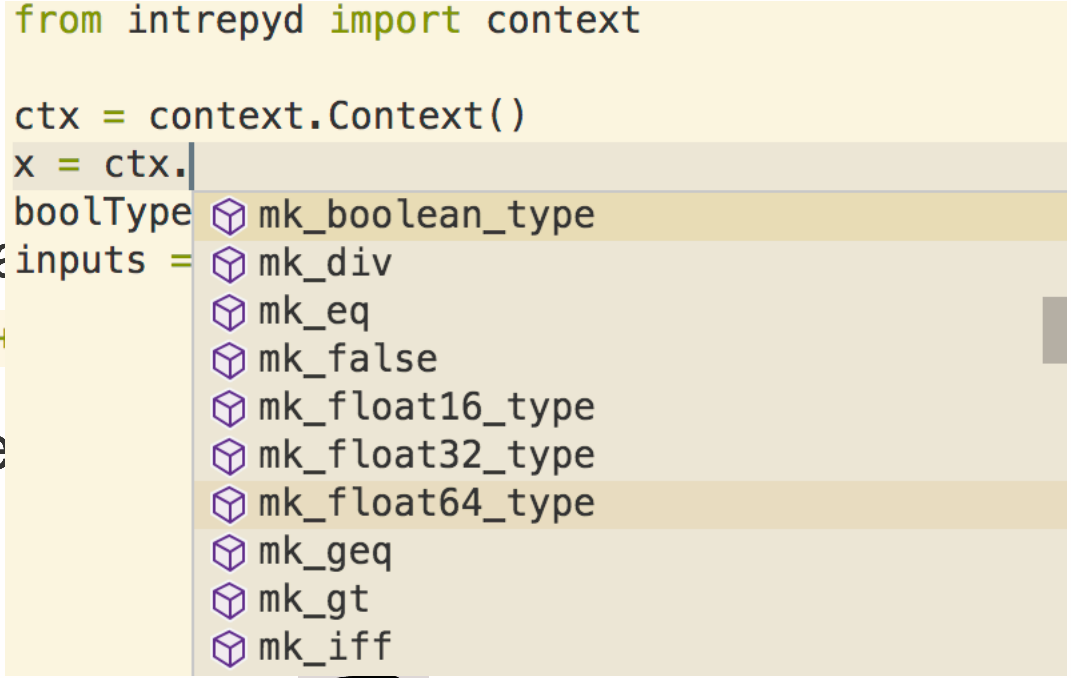
Intrepid’s input language

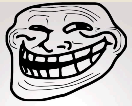
Some advantages

- Functions and classes
- Benchmarks are progra

```
inputs = [ctx.mk_input('in' +
```

- Can natively import the
- Autocompletion



- I don’t have to maintain a parser 

Intrepid’s Simulator

- Linear-time in size of circuit
- Fills out values of a “trace” object
  - Values for inputs can be specified for specific time-stamps, otherwise they are defaulted to false/0
- Traces can be converted into pandas dataframes
- Counter-examples are traces, so they can be readily re-simulated to check their validity

Intrepid’s Engines

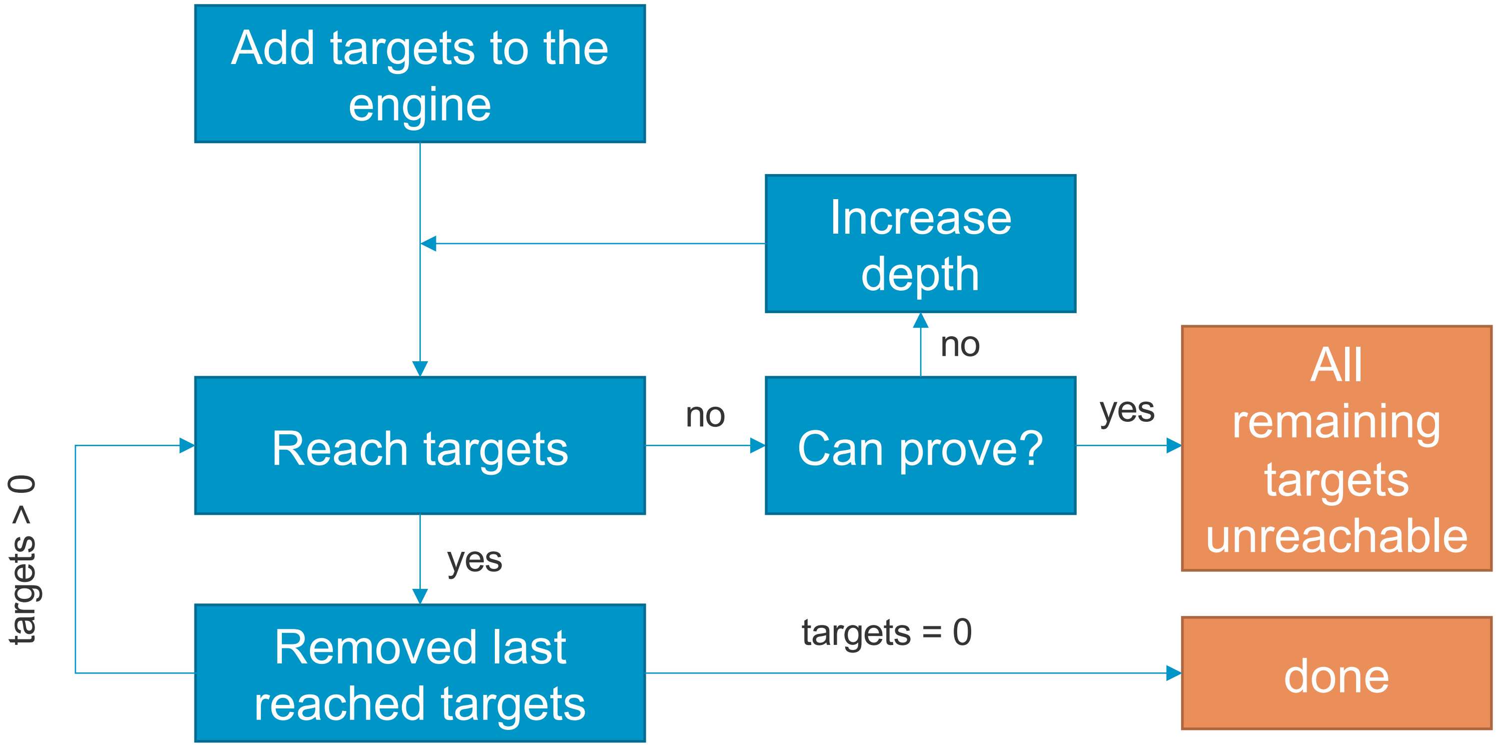
- BMC
  - Finds counterexamples for some targets, at some depth
- Optimizing BMC
  - Find counterexamples that satisfies the highest number of targets
- Backward Reachability
  - Finds counterexamples and proves targets unreachable

Intrepid's Engines

- Multi-target engines
- Target: a Boolean signal that we want to reach
- Watch: values that we want to see in trace

```
class Engine(object):
    """
    Abstract intreface for an Engine
    """

    def __init__(self, ctx):…

    def add_target(self, net):…

    def reach_targets(self):…

    def get_last_reached_targets(self):…

    def get_last_trace(self):…

    def remove_last_reached_targets(self):…

    def add_watch(self, net):…

    def can_prove(self):…

    def can_optimize(self):…
```

Intrepid's Engines



An example application: ATG

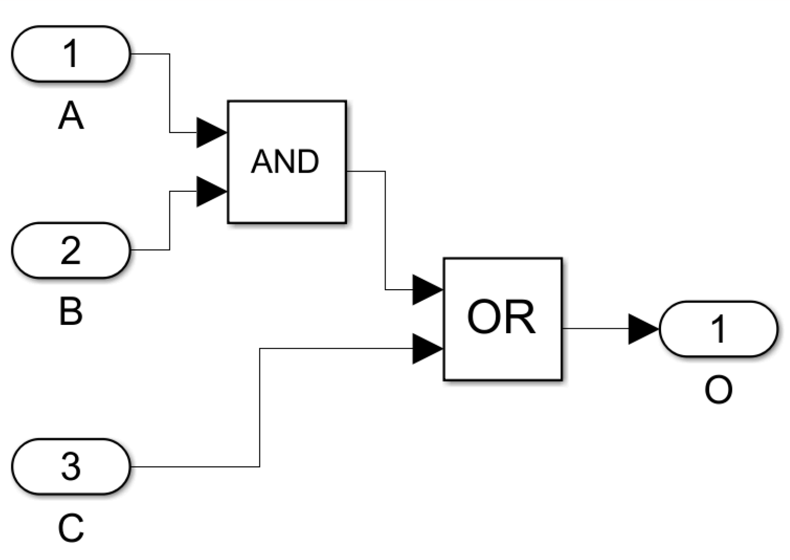
ATG: compute MC/DC

- MC/DC is a testing criterion defined in DO-178C, for critical software
- Decision: a sub-circuit with a Boolean output
- Condition: a Boolean net in the decision that needs to be observed
- Task: given a decision D, for each condition C find two tests T1, T2 such that
  - C has value true in T1
  - C has value false in T2
  - D evaluates differently in T1 and T2

ATG: compute MC/DC

- Each row is a test
- Tests 0 and 1 show MC/DC for A

	A	B	C	O
0	T	T	F	T
1	F	T	F	F
2	T	F	F	F
3	F	F	T	T
4	F	F	F	F

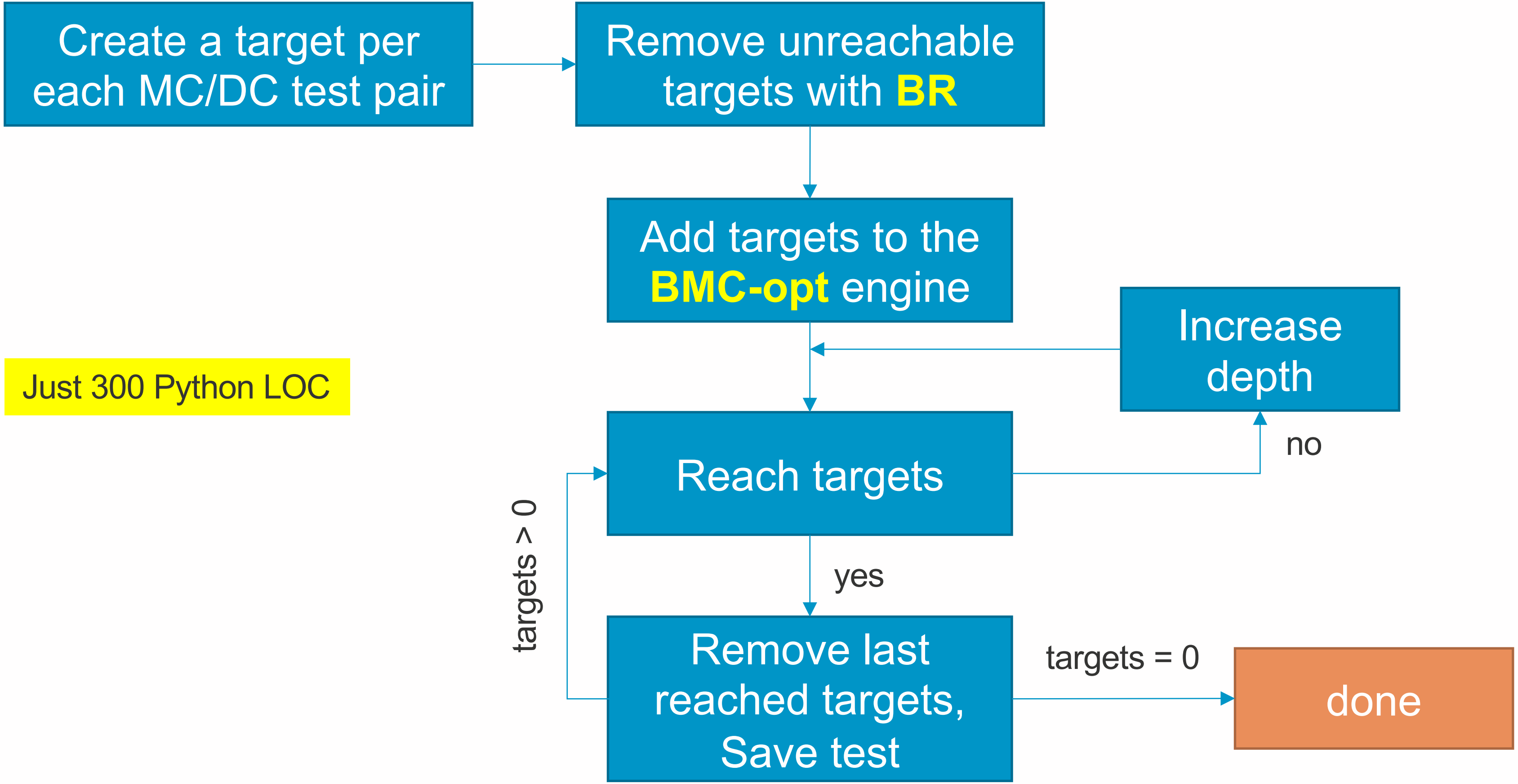


ATG: compute MC/DC

- To come up with suitable tests (the table) is easy
- … but, the less tests are produced, the better
  - Tests are to be written down on tables and reviewed by the FAA (no kidding)

- It is not so easy, it is an optimization problem
- Also, not merely combinational, sequential part plays a role too
- Need for an optimizing-BMC

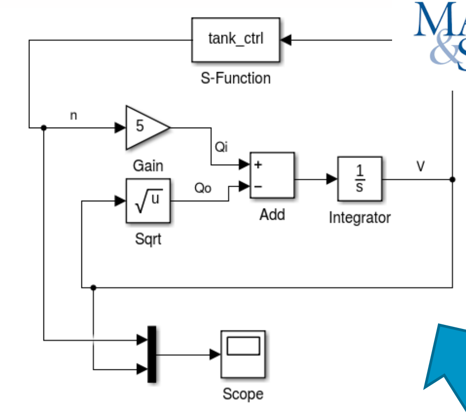
ATG: compute MC/DC



Just 300 Python LOC

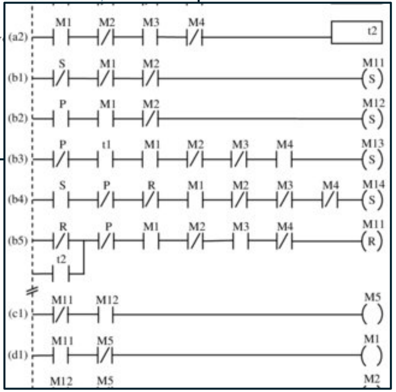
Parsers for real-world industrial languages

Control Engineering Languages



IEC 61131-3

```
    END_IF;
Setpoint_IN_STAGE_1_FAILED:
    (* During 'STAGE_1_FAILED': '<S1>:119' *)
    IF (stage3_sensor <= 0) OR (stage2_sensor <= 0) THEN
        (* Transition: '(<S1>:150' *)
        (* Transition: '(<S1>:152' *)
        IF stage2_sensor > 0 THEN
            (* Transition: '<S1>:155' *)
            is_c2_Setpoint := Setpoint_IN_STAGES_1_3_FAILED;
            (* Entry 'STAGES_1_3_FAILED': '<S1>:120' *)
            rtb_stage1_setpoint := L0;
            rtb_stage2_setpoint := L0 - overall_target;
            distributed_target := rtb_stage2_setpoint;
        ELSE
            (* Transition: '<S1>:154' *)
            IF stage3_sensor > 0 THEN
                (* Transition: '<S1>:159' *)
                is_c2_Setpoint := Setpoint_IN_STAGES_1_2_FAILED;
                (* Entry 'STAGES_1_2_FAILED': '<S1>:121' *)
                rtb_stage1_setpoint := L0;
                rtb_stage2_setpoint := L0;
                distributed_target := L0 - ov
            ELSE
                guard_0 := TRUE;
            END_IF;
        END_IF;
    ELSE
        guard_0 := TRUE;
    END_IF;
```



Lustre

```
node Sofar( X : bool ) returns ( Sofar : bool );
let
    Sofar = X -> X and pre Sofar;
tel
```

Lustre to Python

- Parser written in Python using ANTLR

- Takes Lustre, dumps Intrepyd’s Python API
  - benchmark.lus => benchmark.py

- Good collection of benchmarks (Kind2), thanks for the effort of collecting them

Simulink/Stateflow to Python

- Simulink to Python: initial translation implemented on top of ConQAT Java libraries
  - Very fast but
  - A pain to implement in detail and to maintain
  - Need to infer data types
- Stateflow to Python: a real nightmare
  - No available specification of the language!
  - Need to guess behavior via simulation

FAILURE

Simulink/Stateflow to IEC-61131 ST to Python

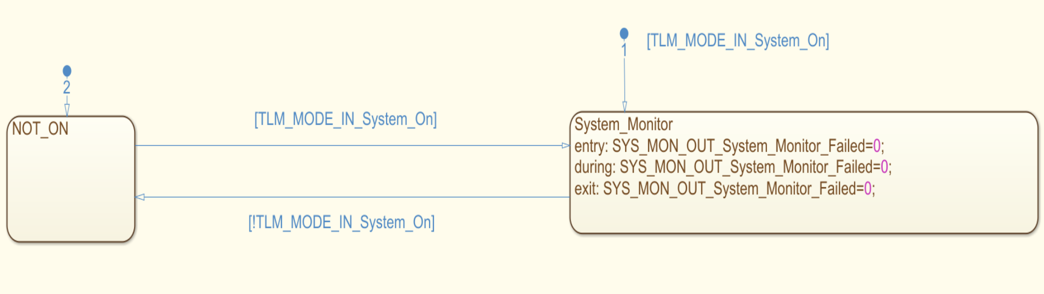
- Matlab provides a toolkit called Simulink PLC Coder that generates IEC-61131 ST

- Two birds with one stone:
  - We can indirectly handle Simulink/Stateflow
  - We can set foot in the Industrial Automation world

- No need to parse the “whole” ST language, but only a subset (i.e., no loops)

- Parser implemented again with ANTLR in Python

Simulink/Stateflow to IEC-61131 ST

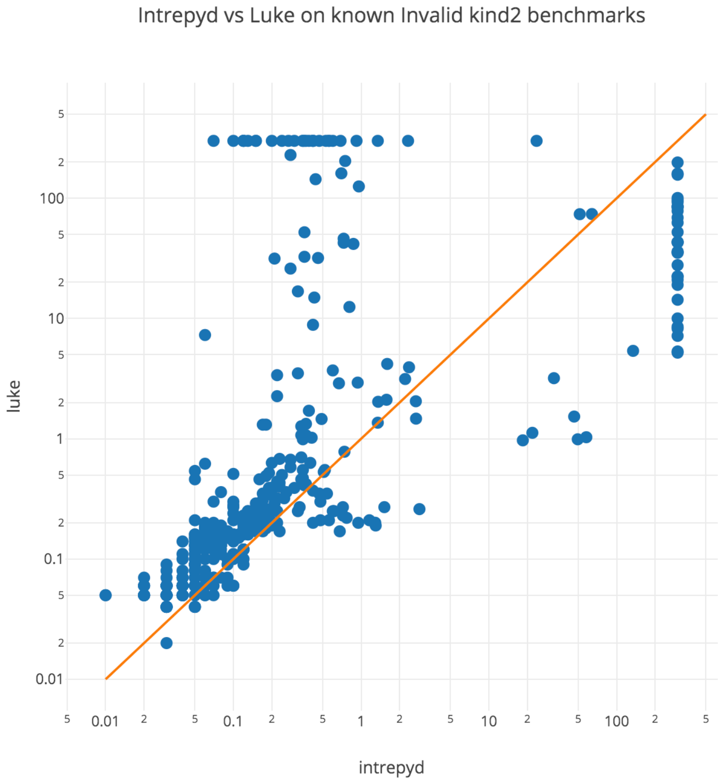



```
CASE ssMethodType OF
    0:
        is_active_c1_Subsystem := 0;
        is_c1_Subsystem := 0;
        Failed := FALSE;
    1:
        IF is_active_c1_Subsystem = 0 THEN
            is_active_c1_Subsystem := 1;
            IF Sys_On THEN
                is_c1_Subsystem := 2;
                Failed := FALSE;
            ELSE
                is_c1_Subsystem := 1;
            END_IF;
        ELSE
            CASE is_c1_Subsystem OF
                1:
                    IF Sys_On THEN
                        is_c1_Subsystem := 2;
                        Failed := FALSE;
                    END_IF;
                ELSE
                    IF  NOT Sys_On THEN
                        Failed := FALSE;
                        is_c1_Subsystem := 1;
                    ELSE
                        Failed := FALSE;
                    END_IF;
            END_CASE;
        END_IF;
END_CASE;
```

Experiments

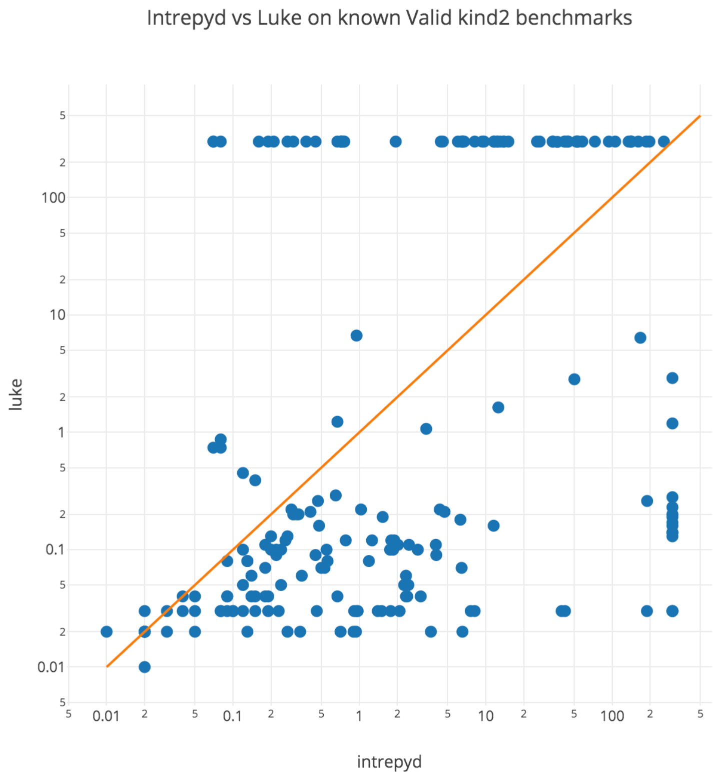
Intrepid vs Luke on Invalid benchmarks

- Basically two different implementation of BMC
- Solved by Intrepid: 341 in 589 s
- Solved by Luke: 342 in 3219 s
- https://plot.ly/create/?fid=robertobruttomesso:30#/



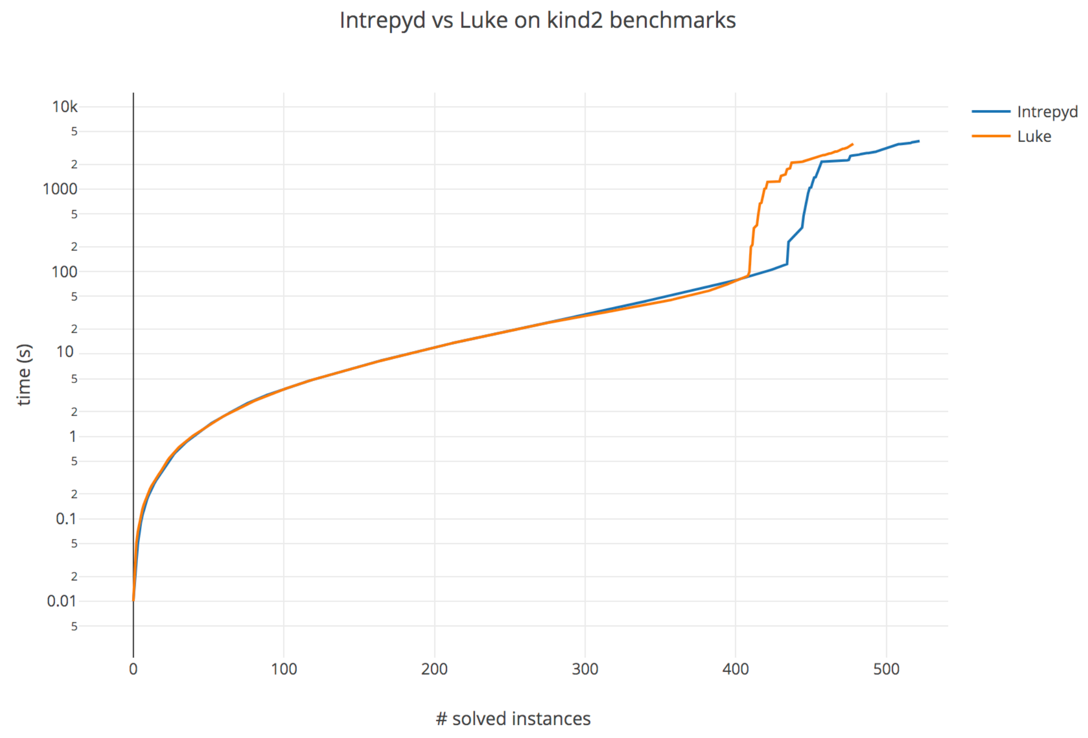
Intrepid vs Luke on Valid benchmarks

- Basically TI vs Backward Reach
- Solved by Intrepid: 182 in 3242 s
- Solved by Luke: 137 in 335 s
- https://plot.ly/create/?fid=robertobruttomesso:32#/



Intrepid vs Luke on Valid benchmarks

- Solved by Intrepid overall: 523 in 3831 s
- Solved by Luke overall: 479 in 3557 s
- https://plot.ly/create/?fid=robertobruttomesso:36#/



Preliminary experiments: GPCA Simulink/Stateflow

- Benchmark from the CocoSim suite (https://coco-team.github.io/cocosim/)
- Simulink/Stateflow model of an infusion pump
- Translated into IEC-61131 ST with Matlab and then into Python with our frontend (takes a few seconds)
- Out of 8 properties, 4 can be solved in about 50 seconds (14 seconds for parsing)

Conclusion

How to get intrepid

- Intrepid is open-source, BSD-3 licensed
- Works on Windows and Linux “officially”

- repo = https://github.com/formalmethods
- Backend: repo/intrepid
- Python API: repo/intrepyd
- `pip install intrepyd`
- Blog: https://formalmethods.github.io

NOZOMI
NETWORKS

Thank You

www.nozominetworks.com