NKind: a model checker for liveness property verification on Lustre programs

Junjie Wei, Qin Li

SEKE 2022, KSIR Virtual Conference Center, Pittsburgh, USA
July 1 - July 10, 2022

Shanghai Key Laboratory of Trustworthy Computing,
East China Normal University, Shanghai, China

Introduction

- Model checking is getting greater concern in industrial field, like real-time reactive systems
- Existing tools mostly concentrate on proving safety properties
- We present NKind, the first model checker for Lustre supporting the verification of liveness properties

Preliminary

- Lustre
  - A synchronous dataflow language widely used in modeling reactive control systems
  - Focus on data and represent them as infinite sequences of values, i.e. dataflows
  - A Lustre program consist of nodes
    - Intputs/outputs as interface as well as local dataflow
    - Contain flow definitions
    - Nodes work synchronously, i.e. at the same speed

Preliminary

```
node main() returns (counter2:int);
var
    counter1 : int;
let
    counter1 = 0 -> 1 + pre counter1;
    counter2 = 100 -> (pre counter2 - counter1);
tel;
```

counter1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
counter2	100	99	97	94	90	85	79	72	64	55	45	34	22	9	-5

Preliminary

- L2SIA-WFR
  - Extended algorithm of liveness-to-safety for handling infinite state space using abstraction
  - Iterates and refines the abstraction by making use of spurious counterexamples
  - Try to prove the absence of a lasso-shape counterexample as an invariant

Architecture

- Parser
  - Accepts an extension to core Lustre
- Simplifier
  - Automata, Array iterator, etc
- Controller
  - Dispatches the proving task
- Parallel engines
  - BMC
  - k-Induction
  - Invariant Generation
  - PDR

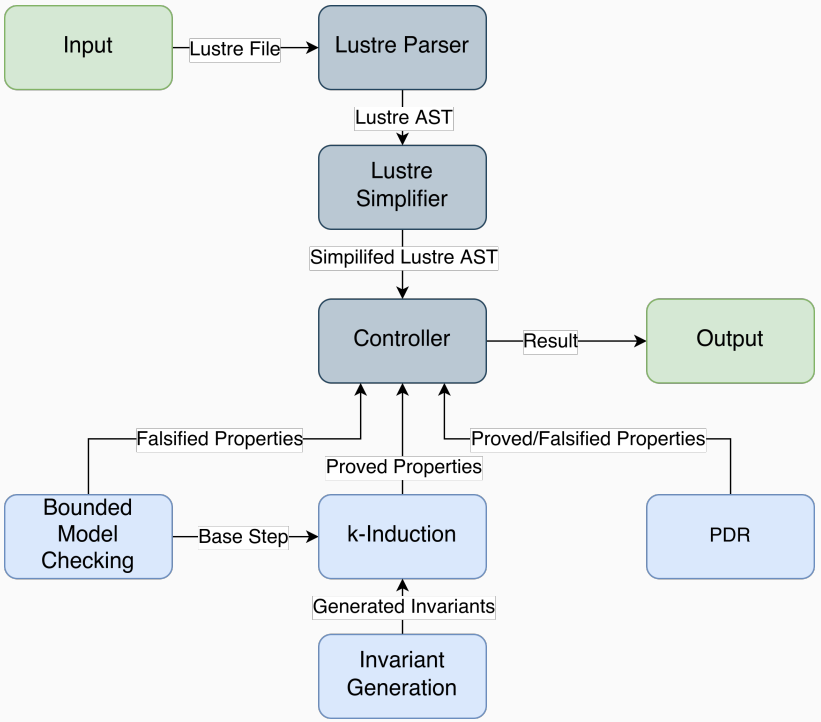

Figure 1: NKind architecture

Features

- Safety property
  - Attempts to prove that the given properties are invariants
  - Make use of the multiple engines mentioned before

```
node main() returns(x: int; y: int; f: bool);
var
    e1: bool;
    counter: int;
let
    x = 20 -> (pre x) + 1;
    y = 0 -> (pre y) + 2;
    f = false -> pre x < pre y;
    counter = 0 -> pre counter + 1;
    e1 = (counter > 21) => f;
    --%MAIN;
    --%PROPERTY Safe e1;
tel;
```

Figure 2: Specifying property with safety property

```
================================================================================
VALID PROPERTIES: [Safety Property: e1] || KInduction || Step = 22 || Time = 479 ms
================================================================================

                    ---------------------------------------
                    ------        SUMMARY          ------
                    ---------------------------------------

VALID PROPERTIES: [Safety Property: e1]

------------------------------------------------------------
```

Figure 3: Safety verification result

Features

- Liveness property
  - Support properties which can have finite-time violations and will finally holds forever
  - Integrate L2SIA-WFR to PDR process

```
node main() returns(x: int; y: int; f: bool);
var
    e1: bool;
let
    x = 20 -> (pre x) + 1;
    y = 0 -> (pre y) + 2;
    f = false -> pre x < pre y;
    e1 = f;
    --%MAIN;
    --%PROPERTY Live e1;
tel;
```

Figure 4: Specifying property with liveness property

```
================================================================================
VALID PROPERTIES: [Liveness Property: e1] || Pdr || Step = 1 || Time = 2 s 482 ms
================================================================================

                    -------------------------------------
                    ------        SUMMARY          ------
                    -------------------------------------

VALID PROPERTIES: [Liveness Property: e1]

--------------------------------------------------------
```

Figure 5: Liveness verification result

Features

- Liveness property
  - Structure referred to ic3ia
  - Monotonically strengthens the original transition system
  - Refine abstraction from spurious counterexamples

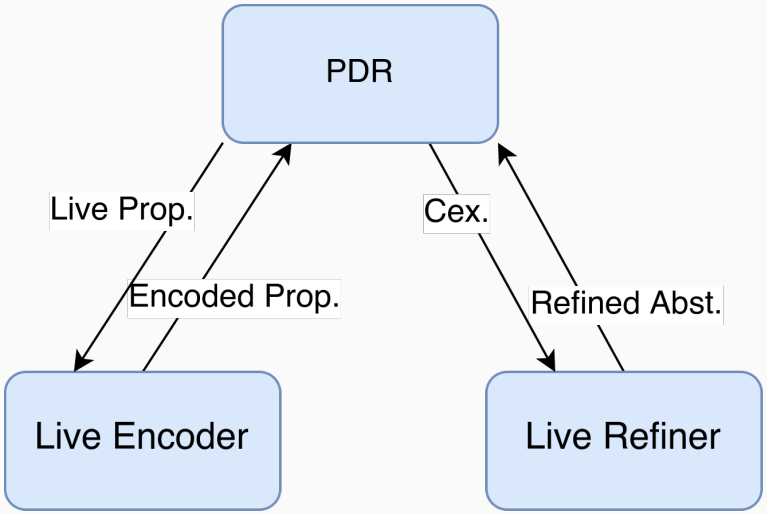

Figure 6: Embedding liveness extension to PDR process

Evaluation

- Safety benchmark from Kind (864)

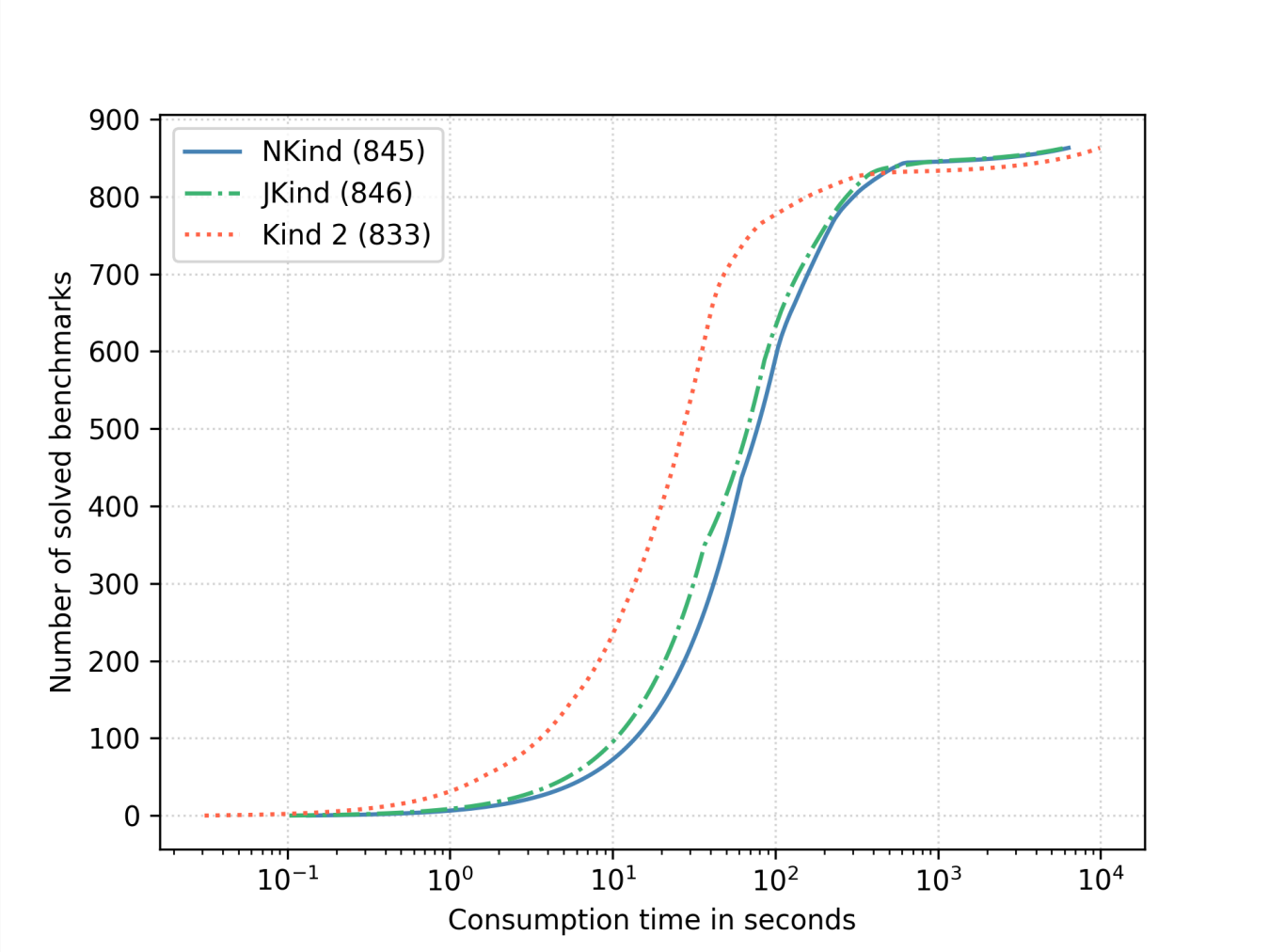

Figure 7: Verification results on safety property benchmark

Evaluation

- Liveness benchmark from ic3ia

Test name	ic3ia	NKind
any-down-live	35.77	1.88
parallel-live	51.68	12.61
binary-live	0.09	0.86
piecewise-live	0.11	1.13
count-nested-live	0.37	1.87
stabilize-live	5.16	3.99
count-down-live	1.05	1.00
swap-dec-live	1.53	2.73
count-up-to-sym-live	5.84	1.11
refine_disj_problem	Timeout	Timeout

Table 1: Representative verification results on liveness property benchmark

Evaluation

- Liveness benchmark transformed from Kind (225)

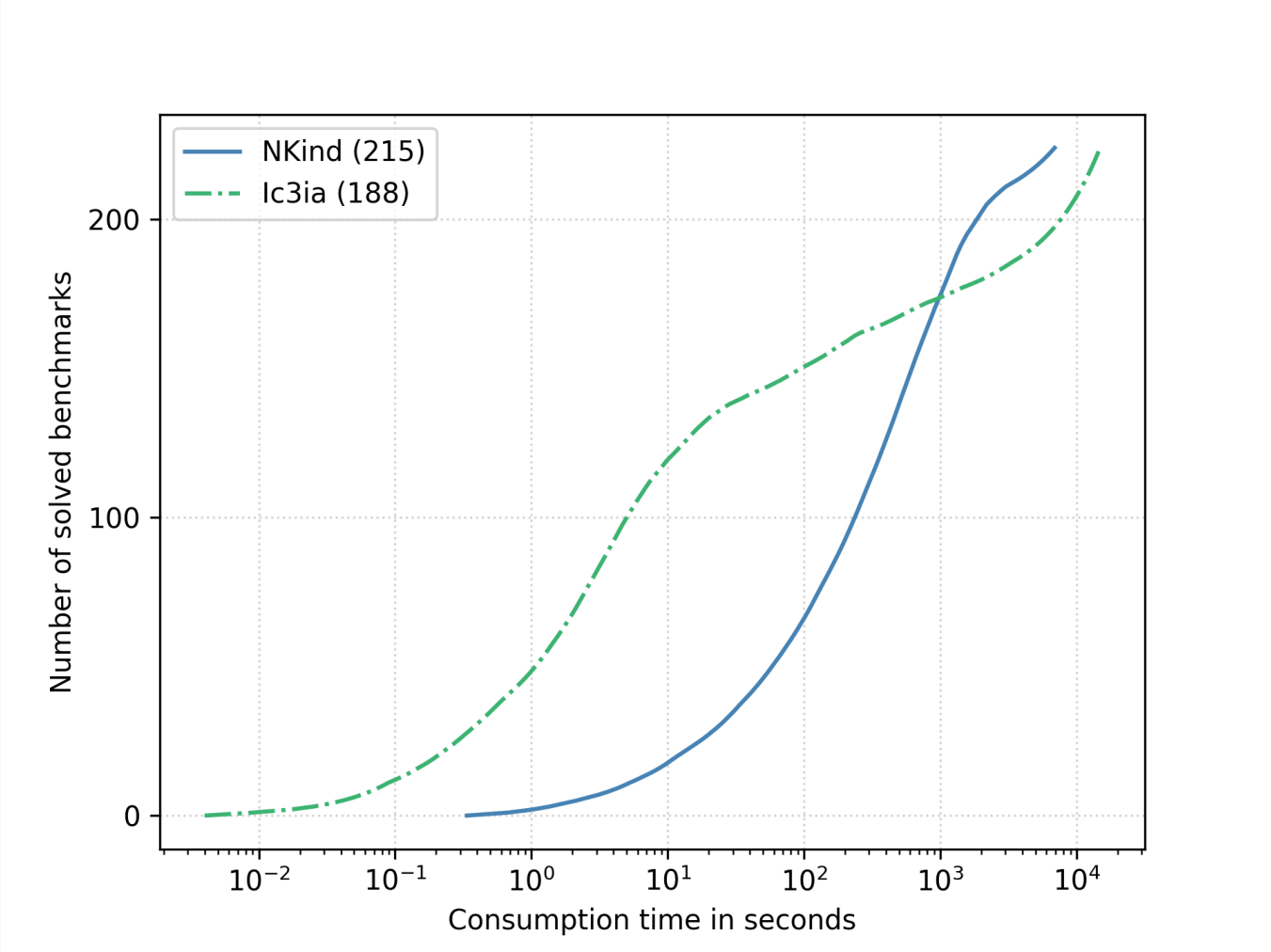

Figure 8: Verification results on more complex liveness property benchmark

Conclusion

- We present NKind, the first model checker for Lustre supporting the verification of liveness properties
- NKind has a rather competitive performance comparing to mainstream tools
- More in-depth optimizations for performance is left for future work

Thank you!