Conservative approach in conformance checking

Frederic Beal
Tokyo Institute of Technology

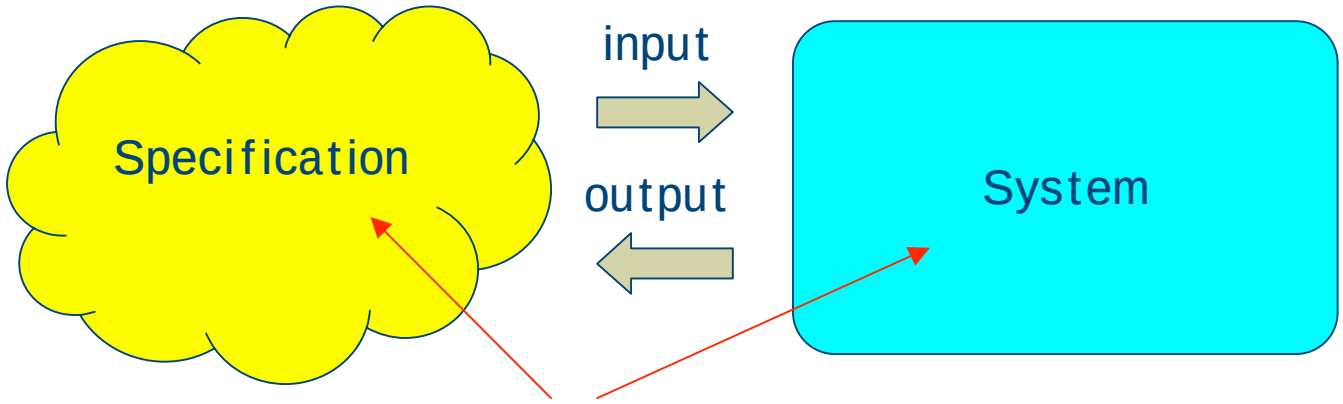
Tomohiro Yoneda
National Institute of Informatics

Background and Goal

- Formal verification of hardware or software is important
  - Its cost is too high
  - Developing some approach to reducing its cost is important
- Goal
  - Use conservative approach in conformance checking
    - allowing false negatives, but guaranteeing that false positives never appear

Conformance Checking

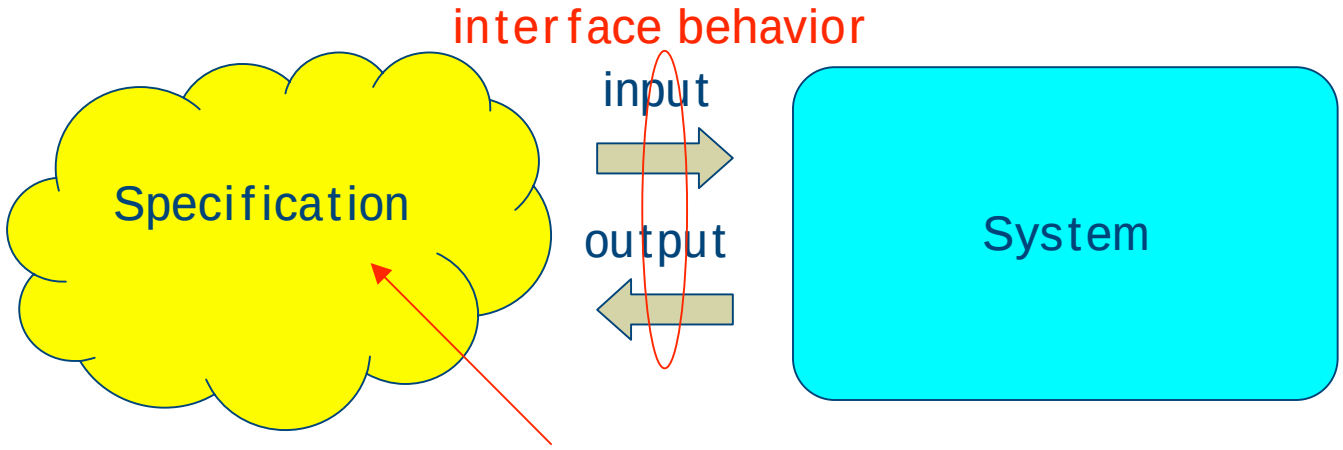
- Specification and system are expressed by the same model
- Safety, some restricted liveness, and some internal properties are checked



Product of both state spaces should be explored

Idea (1)

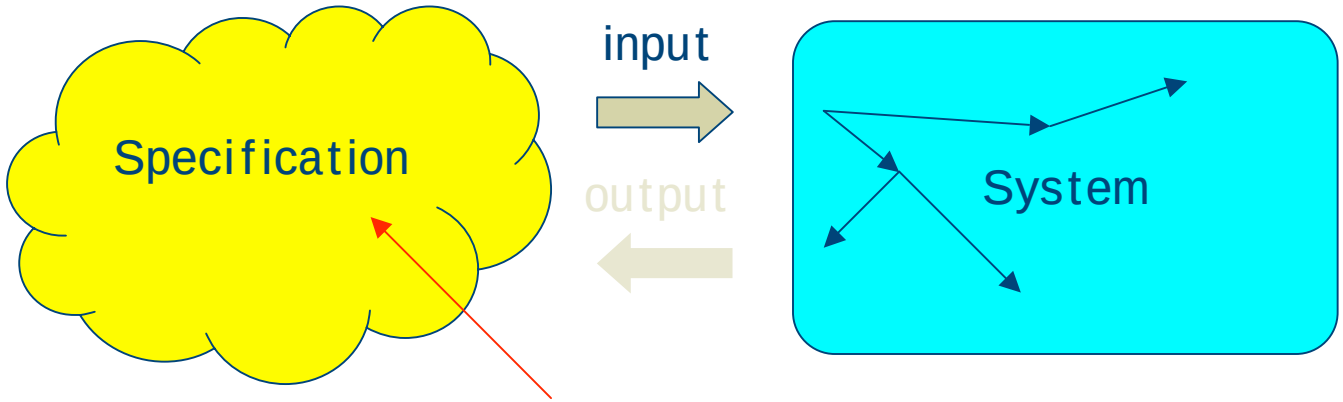
- Use only the state space of the specification
- Guess the internal states of the system from its interface behavior



Only the state space of Specification is explored

Idea (2)

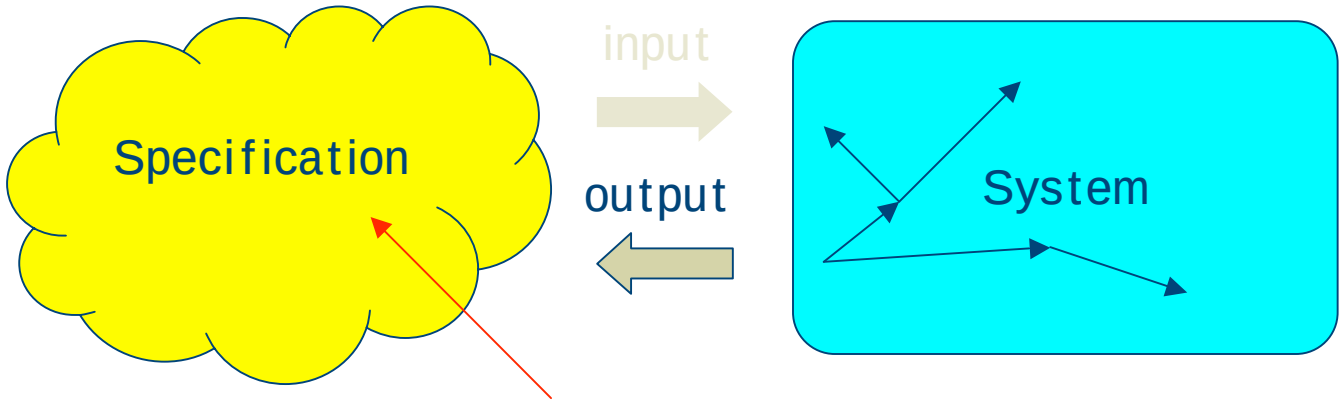
- How to guess the internal states
  - When an input is given
    - use forward implication to propagate enabling condition of events from input to output
    - eg.) A is given → B is enabled if C is enabled



Only the state space of Specification is explored

Idea (3)

- How to guess the internal states
  - When an output is observed
    - use backward implication to decide the actually fired events
    - eg.) A is observed → B must have fired



Only the state space of Specification is explored

Why conservative? (1)

- Decision
  - Safety failure
    - Illegal output is produced
  - Strong conformance failure
    - Expected output is not produced
- Internal states cannot be determined exactly
  - Use a symbol to represent uncertainty to capture all possible behavior of system

Why conservative? (2)

- In the guessed system state
  - if an output event is enabled with some consistent conditions
    - it can occur eventually
  - if uncertainty symbol is propagated to an output
    - it may or may not occur

If the output is not ready to occur in specification, safety failure is reported in both cases

Why conservative? (3)

- In the guessed system state
  - if an output event is not enabled
    - it actually never occurs
  - if uncertainty symbol is propagated to an output
    - it may or may not occur

If the output is enabled in specification,
strong conformance failure is reported in both cases

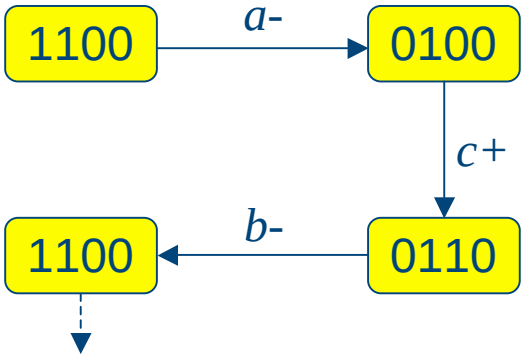
Questions

- Does the proposed idea really reduce the cost of conformance checking?
- How often are false negatives produced?

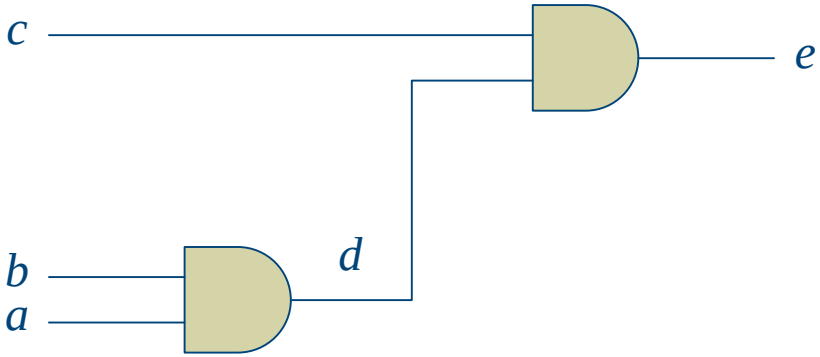
- Case study
  - Verification of asynchronous circuits

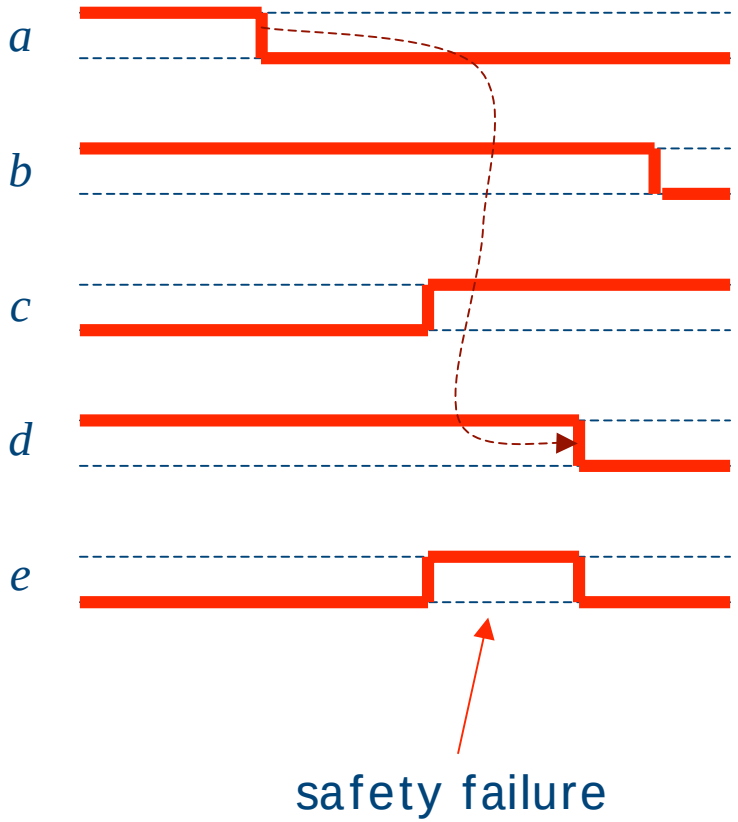
Verification of asynchronous circuits

Specification



System



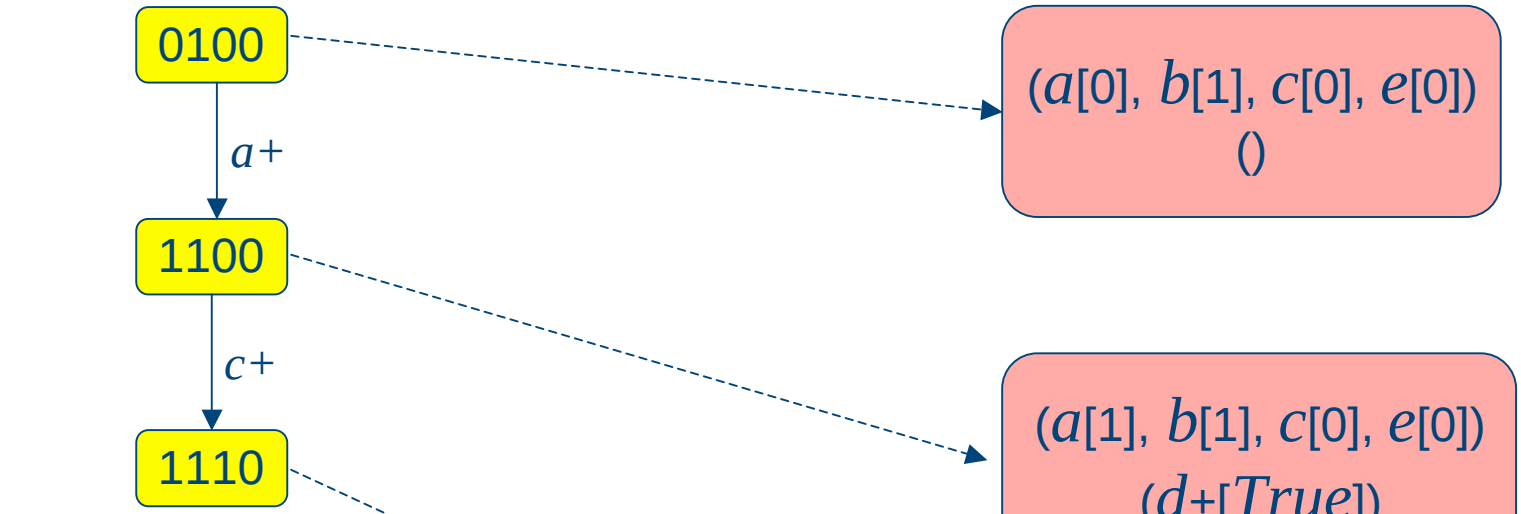


State representation

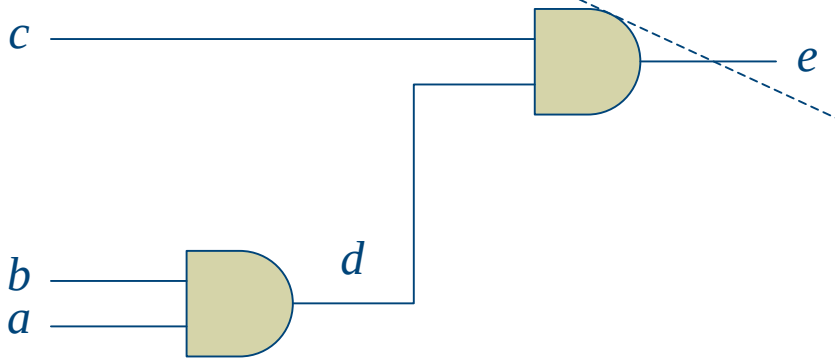
- State
  - Interface signal vector
    - $(a[0], b[1], c[1], e[0])$
  - Enabling conditions for pending transitions
    - $(d+[\text{True}], e+[d+ \vee f+])$
      - $d+$ is already enabled
      - $e+$ will be enabled if either $d+$ or $f+$ fires
- State change
  - only when specification state changes

Forward implication

Specification

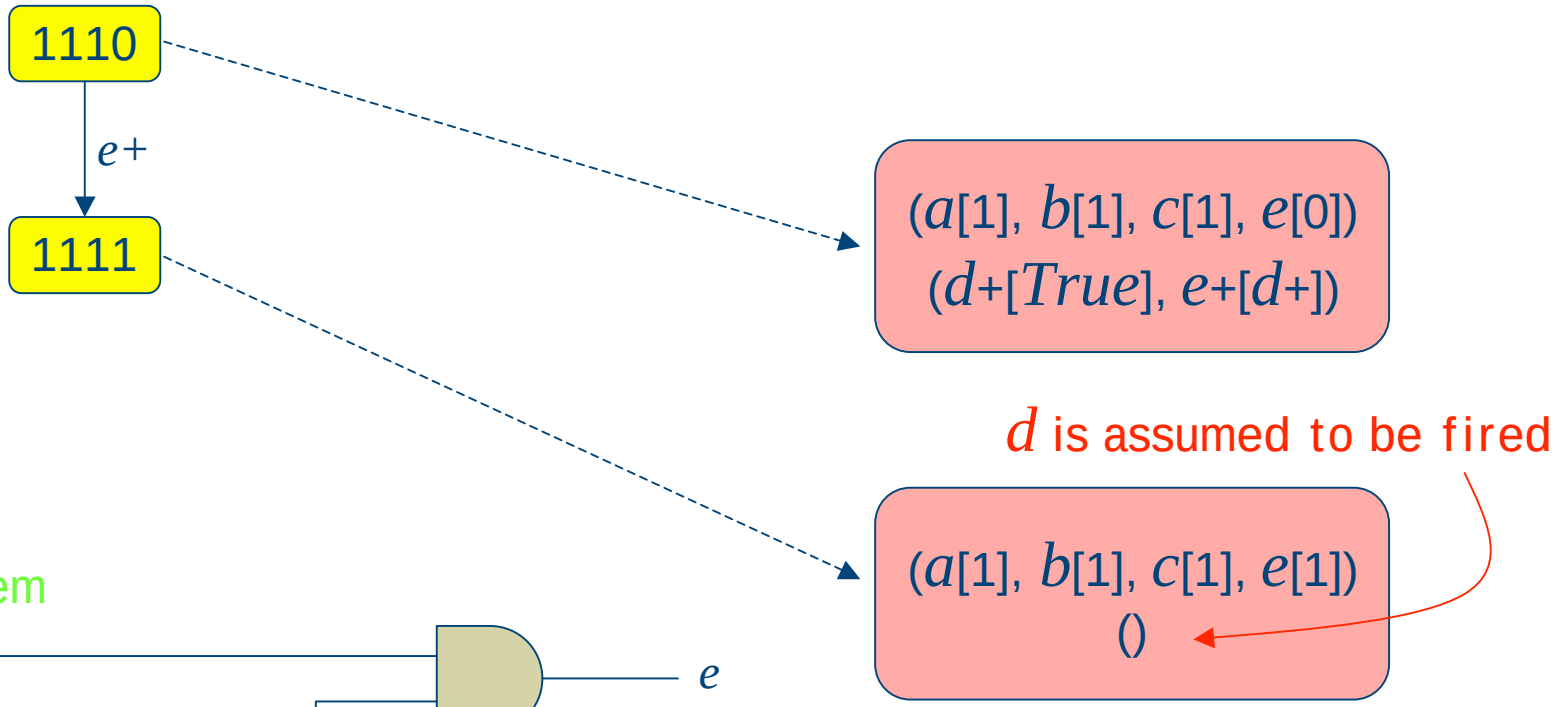


System

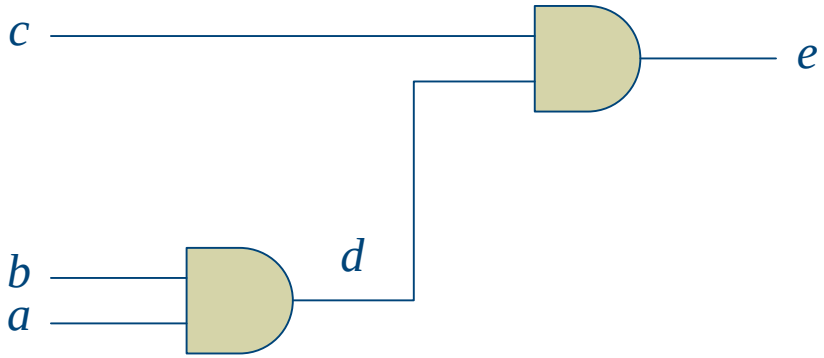


Backward implication

Specification

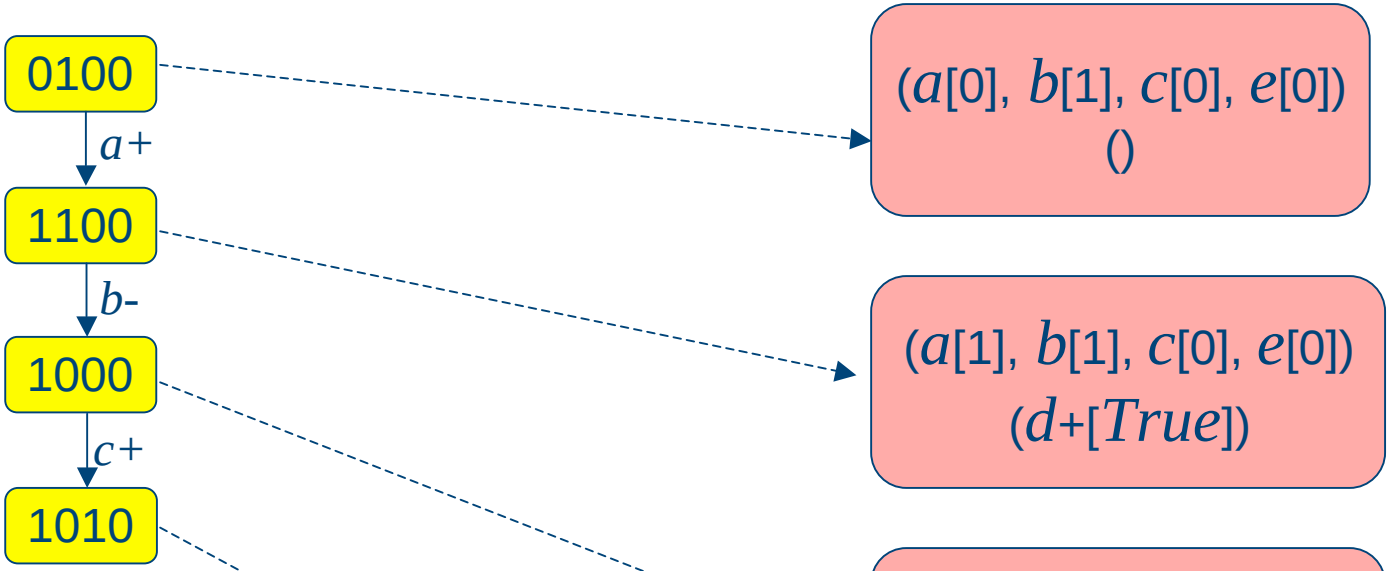


System



Failure detection

Specification



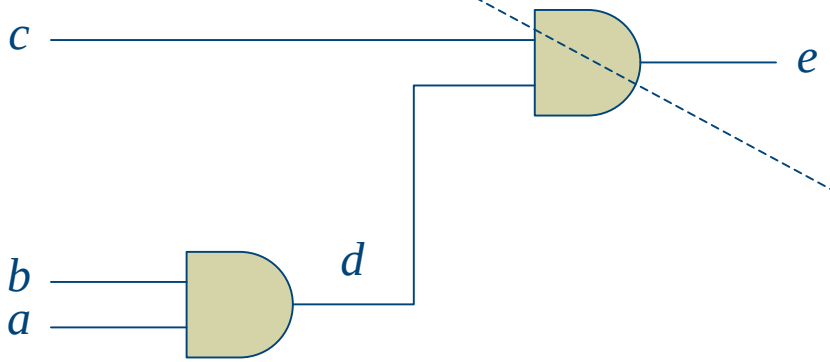
(a[0], b[1], c[0], e[0])
()

(a[1], b[1], c[0], e[0])
(d+[True])

(a[1], b[0], c[0], e[0])
(d_U[True])

(a[1], b[1], c[1], e[0])
(d_U [True], e_U[d_U])

System



Preliminary experimental results

- versify: BDD based conformance checker
- No false negatives found in proposed method

Circuits	#(I+O)	#(internal)	#(spec. states)	CPU time (s)	
				versify	proposed
LMS4_pr11	19	8	538	44.5	9.6
FIR5_2mul	19	24	4730	214.8	45.7
IIR_2mul_2	10	12	291	1.65	0.15
FIR3_1mul2	13	26	1753	3.13	4.3

Future work

- Apply it to timed system verification
  - Forward/Backward implication handling timing information (DBMs)
  - Comparison with verifiers based on timed automata and time Petri nets