Efficient Automata-Based Assertion-Checker Synthesis of SEREs for Hardware Emulation

Marc Boulé and Zeljko Zilic

January 24th
ASPDAC 2007

Outline

- Assertion-Based Verification and PSL
- Assertion-Checker Generation
- Automata Synthesis of Sequences (SEREs)
  - 3 Key Algorithms
- Experimental Results

Introduction

- Temporal sequences: crucial for temporal assertion languages such as SVA and PSL
- Need hardware implementation of sequences for resource-efficient assertion checker circuits
- Assertion checkers useful in
  - Hardware emulation
  - Post-fabrication silicon debug
  - On-line monitoring

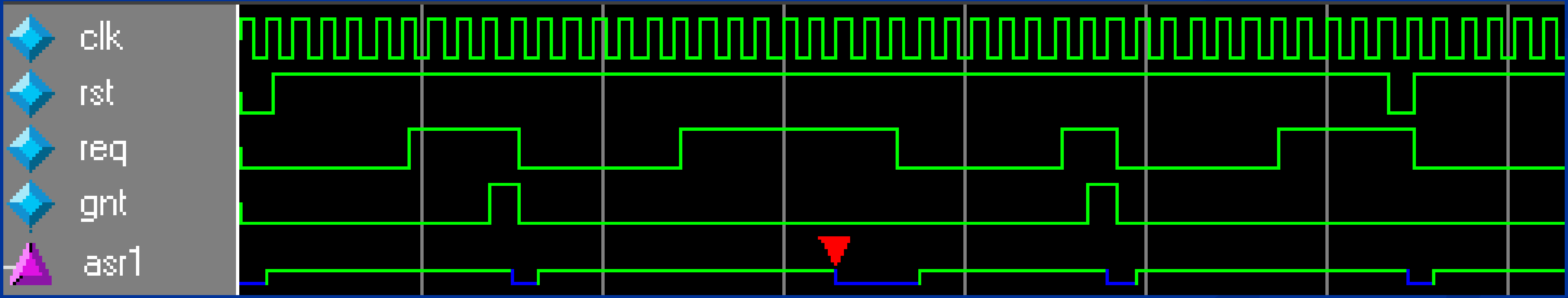
Assertions

- Assertion Based Verification: Assertion failures used to identify bugs in design
- Assertion: Formal statement for describing correct behavior of design
  - Formal (static) or simulation-based (dynamic) verification
  - IEEE 1850 Property Specification Language (PSL), SVA
- Sequential Extended Regular Expressions (SEREs) used to express temporal sequences of events
  - **How to implement automata-based SEREs for dynamic verification checkers?**

Verification With Assertions

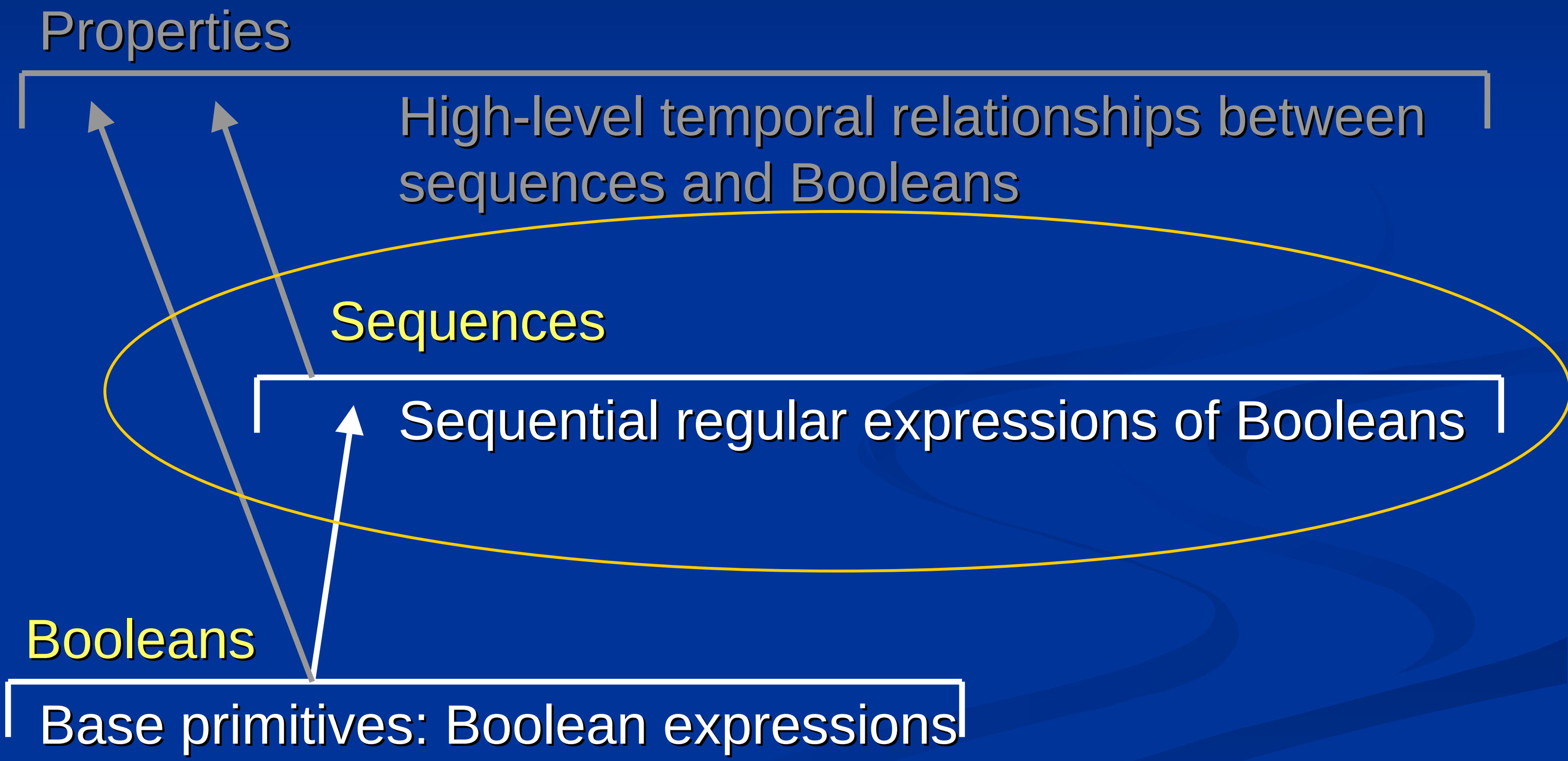
- **always** {~*req;req*} |-> ({ [*0:5] ; *gnt* } **abort** ~*rst*)
- Does the property hold?

– Inspect waveform or write code to check the property (often tedious!)



+ Assertion does the monitoring for us

Property Specification Language



Properties

High-level temporal relationships between sequences and Booleans

Sequences

Sequential regular expressions of Booleans

Booleans

Base primitives: Boolean expressions

Property Specification Language

Properties

…, Sequences used in conditional and obligation contexts in properties

Sequences

Concatenation, Disjunction, Fusion, Repetition, Goto repetition, Non-consecutive repetition, Intersection

Booleans

HDL Boolean expressions, implication and equivalence, PSL built-in functions

PSL SEREs – Base Operators

- Sequential regular expressions *r* composed of:

b	Boolean expressions
$\{r\}$	Grouping (like parentheses in RE)
$r_1 ; r_2$	Concatenation
$r_1 : r_2$	Fusion (overlapped concatenation)
$r_1 \mid r_2$	Disjunction
$r_1 \&\& r_2$	Intersection, length-matching
[*0]	Empty SERE (like ε in RE)
r[*]	Kleene closure (like * in RE)

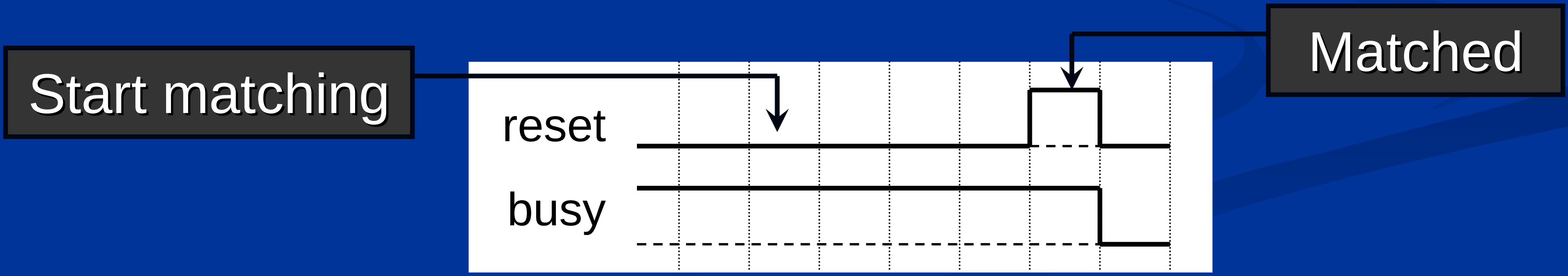
- Example: {{*busy*[*2] ; *ack* } | {*busy*[*5] ; *error* }}

PSL SEREs – Sugaring

- SERE sugaring rules in PSL (non-exhaustive):

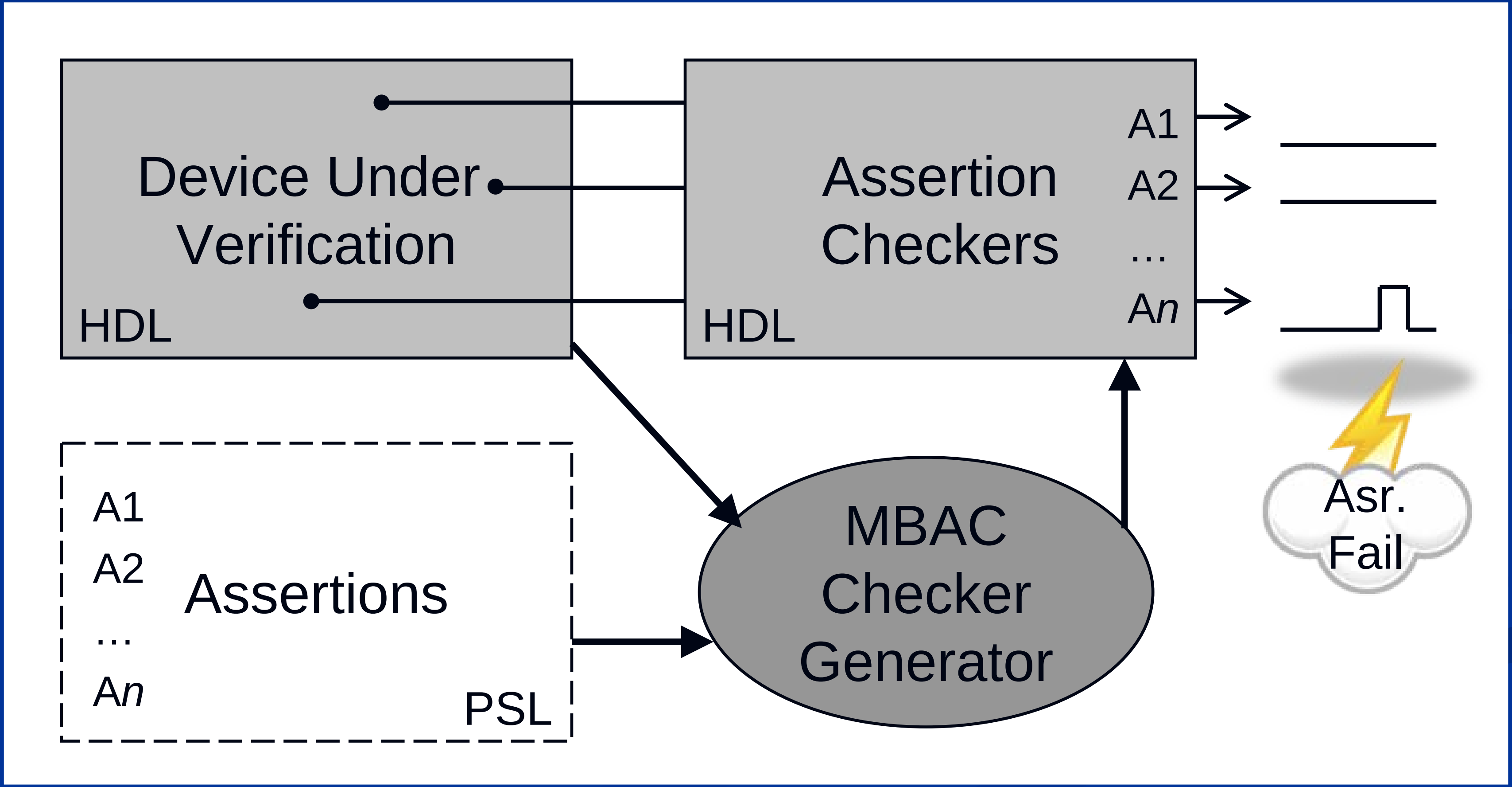
r[*c] = r ; r ; … ; r	Fixed count repetition, c>0
r[*l:h] = r[*l] | … | r[*h]	Bounded repetition
b[->] = (~b)[*] ; b	Goto repetition
b[->c] = {b[->]}[*c]	
b[=c] = b[->] ; (~b)[*]	Non-consecutive repetition

- Example: {{busy[*]} && {reset[->]}}



MBAC Checker Generator

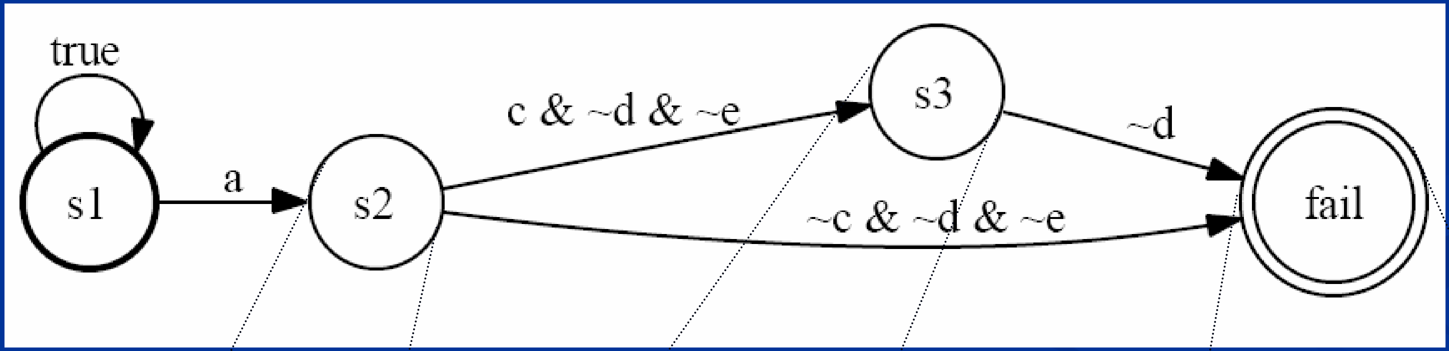
- Circuit-level checkers from assertion statements

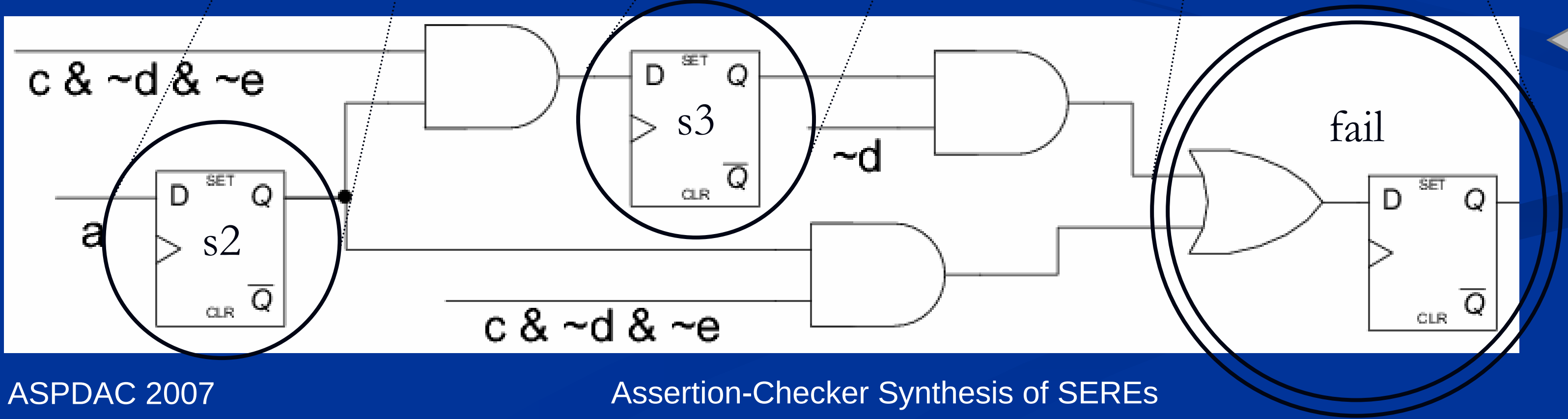


Checker Generation Process

- Assertion → Finite Automaton → HDL

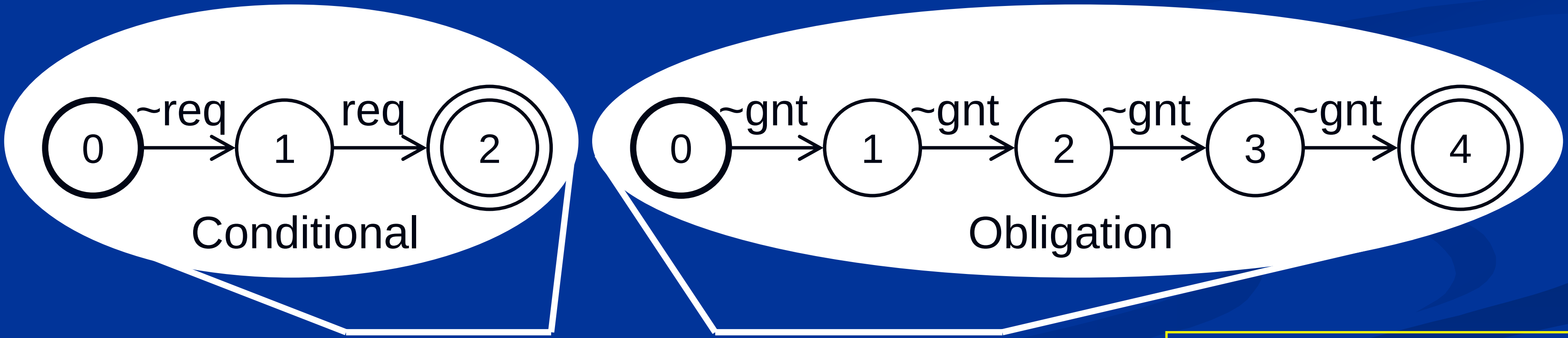
assert always ({a} |=> { {c[*0:1]; d} | {e} })





SERE Modes in Properties

- Conditional Mode: Identify all occurrences of expression for a given start condition
- Obligation Mode: Identify the first failure of expression for a given start condition

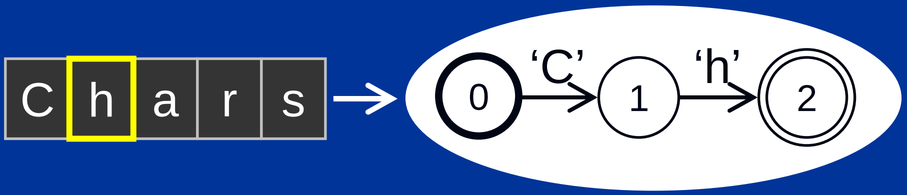


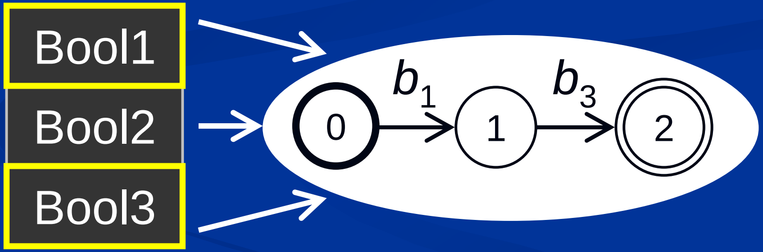
- **always** *{~req;req}* |-> ({ [*0:3] ; *gnt* })

“|->” is temporal implication

SEREs vs. Regular Expressions

- SEREs extend traditional REs with:
  - Length-matching intersection
  - Fusion (overlapped concatenation)
  - Based on Boolean expressions (<u>not mutually exclusive symbols as in REs</u>)

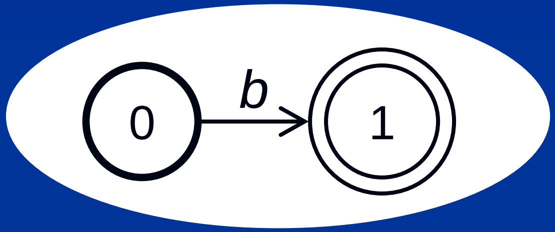




- SEREs in properties → failure detection also
  - Obligation mode needed, not only occurrence detection

SERE FA Construction (Conditional mode)

- Inductive construction [Hopcroft'00]
- Base case: Top level Boolean Expressions b_i

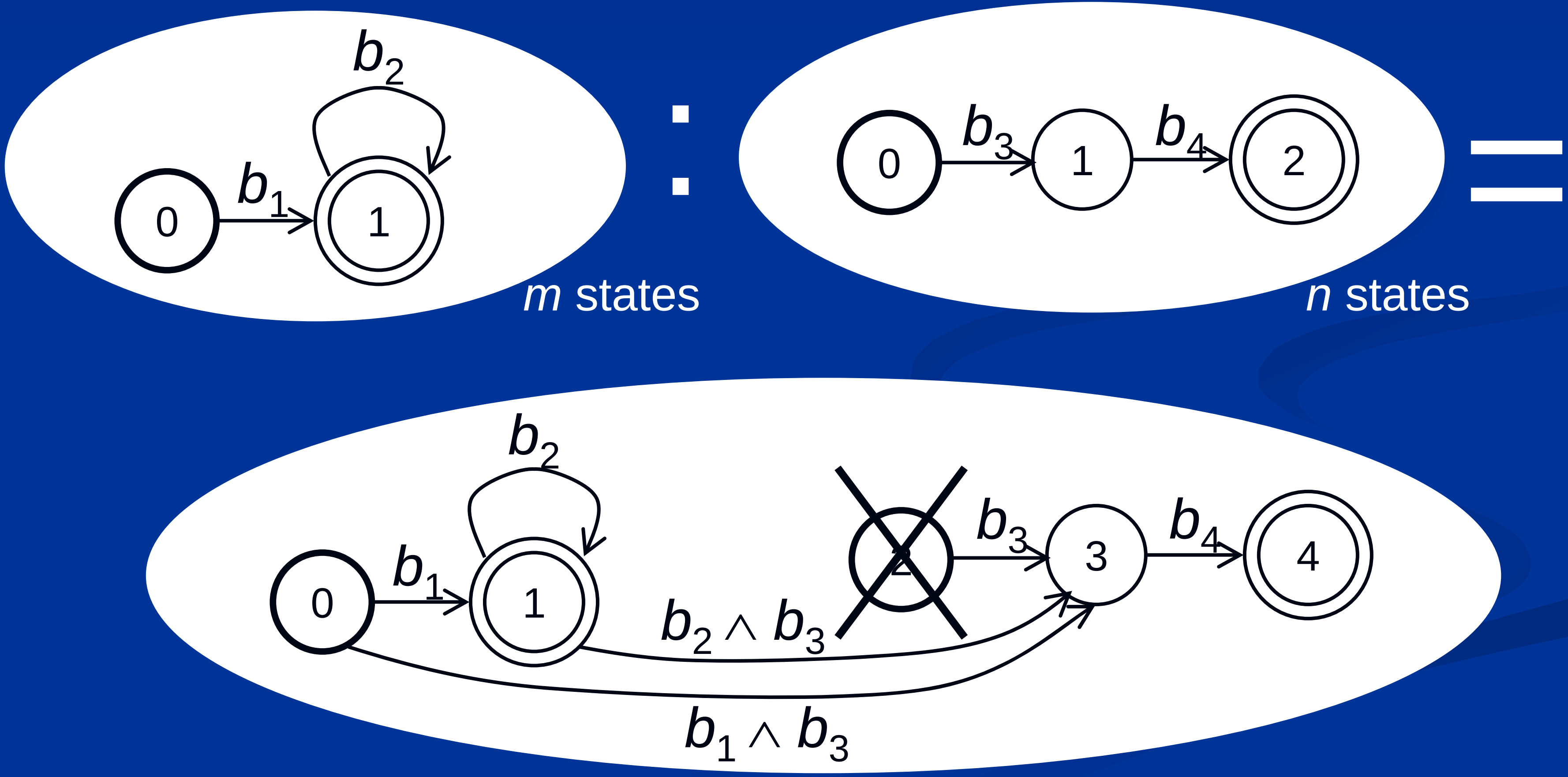


- Inductive cases:
  - Disjunction
  - Concatenation
  - Kleene closure
  - Fusion
  - Intersection (length matching)

$\}$ Custom algorithms $\rightarrow$ same effect as: NFA Construction [Hopcroft'00] + ε Removal

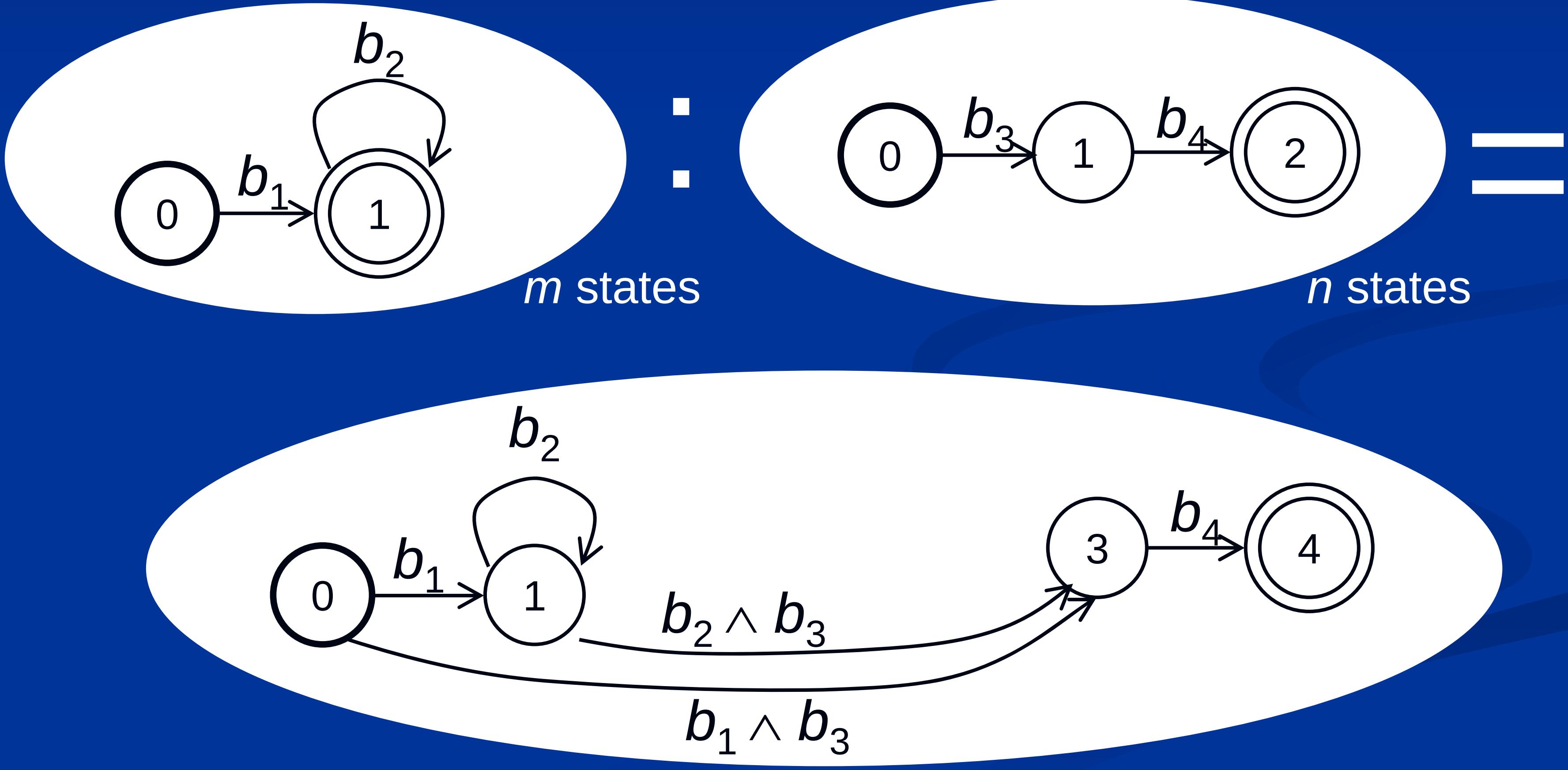
SERE FA Construction – Fusion

- Example for $\{b_1; b_2[*]\}:\{b_3; b_4\}$ $O(m+n)$



SERE FA Construction – Fusion

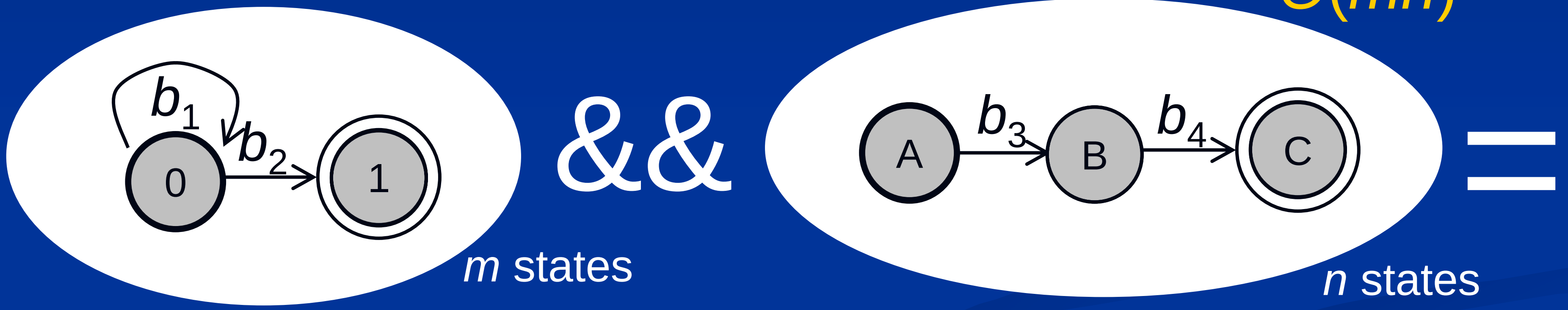
- Example for $\{b_1; b_2[*]\}:\{b_3; b_4\}$ $O(m+n)$



SERE FA Construction – Intersection

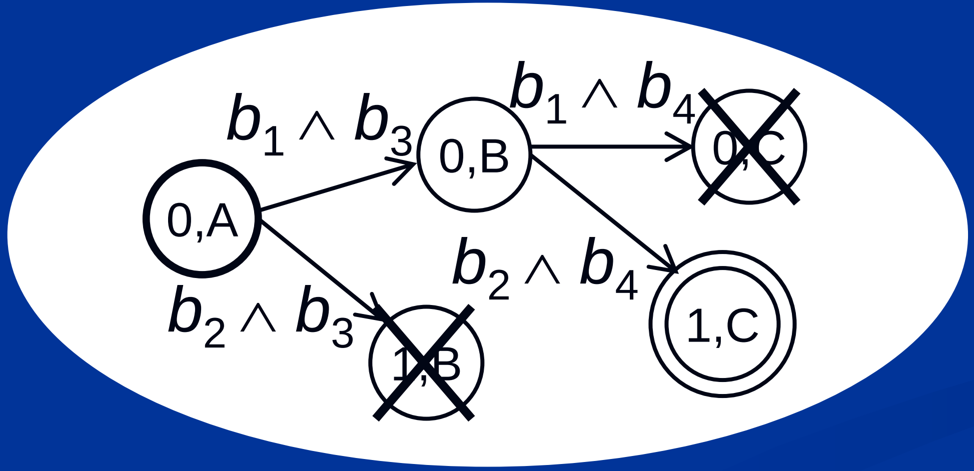
- Example for {b_1[*];b2}&&{b_3; b_4}

Worst case
$O(mn)$



m states

n states



State construction stack:

→ 1,C: *no edges in* C. (1 & C final)

→ 0,C: *no edges in* C.

→ 0,B:“b1 $\wedge$ b4” 0,C; “b2 $\wedge$ b4” 1,C

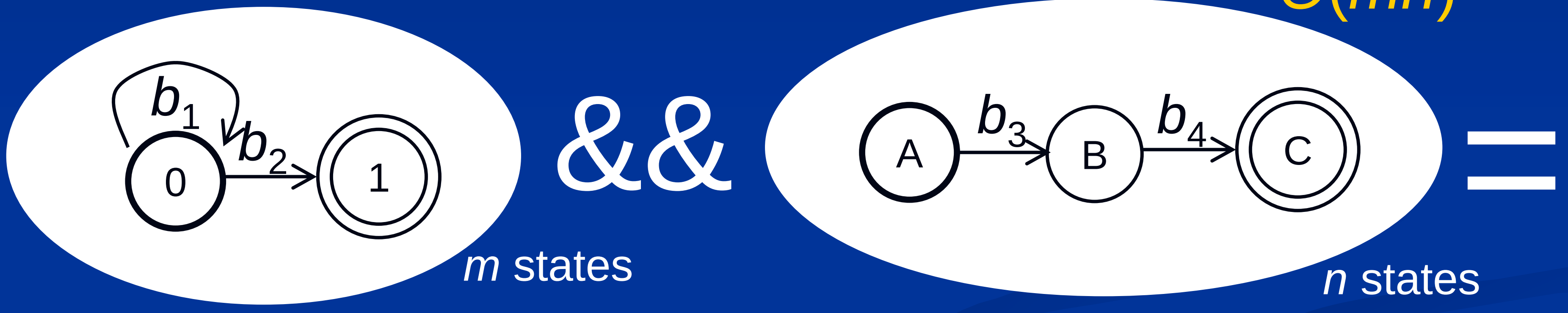
→ 1,B: *no edges in* 1.

→ 0,A:“b2 $\wedge$ b3” 1,B; “b1 $\wedge$ b3” 0,B

SERE FA Construction – Intersection

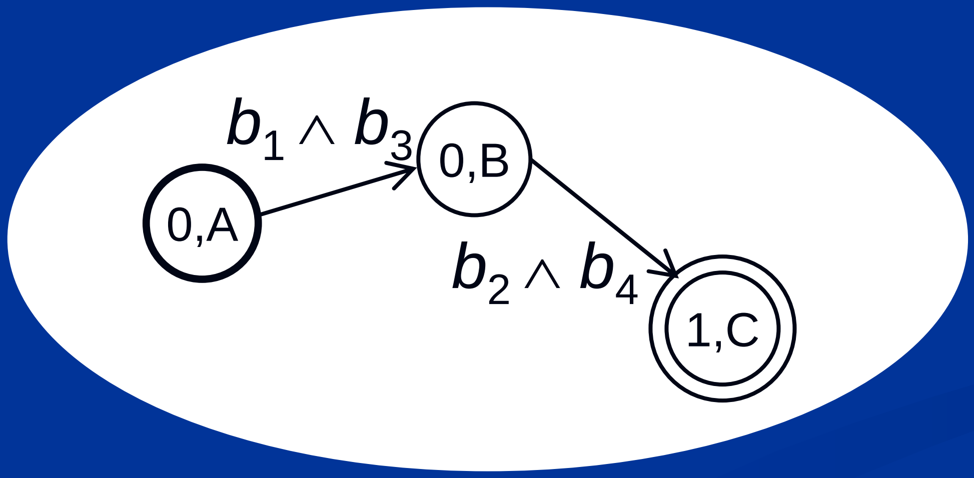
- Example for $\{b_1[*];b2\}\&\&\{b_3;\ b_4\}$

Worst case $O(mn)$



m states

n states



State construction stack:

1,C: *no edges in* C. (1 & C final)

0,C: *no edges in* C.

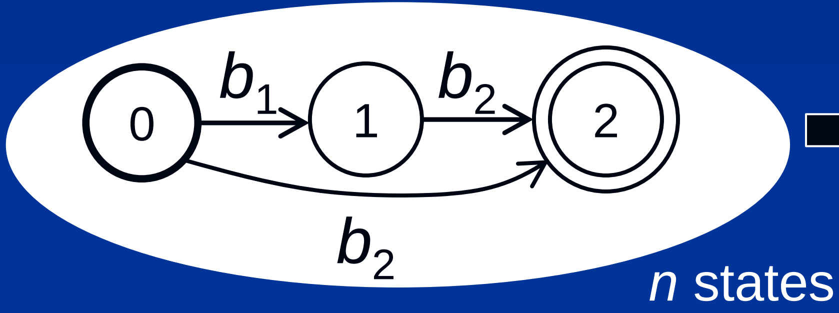
0,B:"b1 $\wedge$ b4" 0,C; "b2 $\wedge$ b4" 1,C

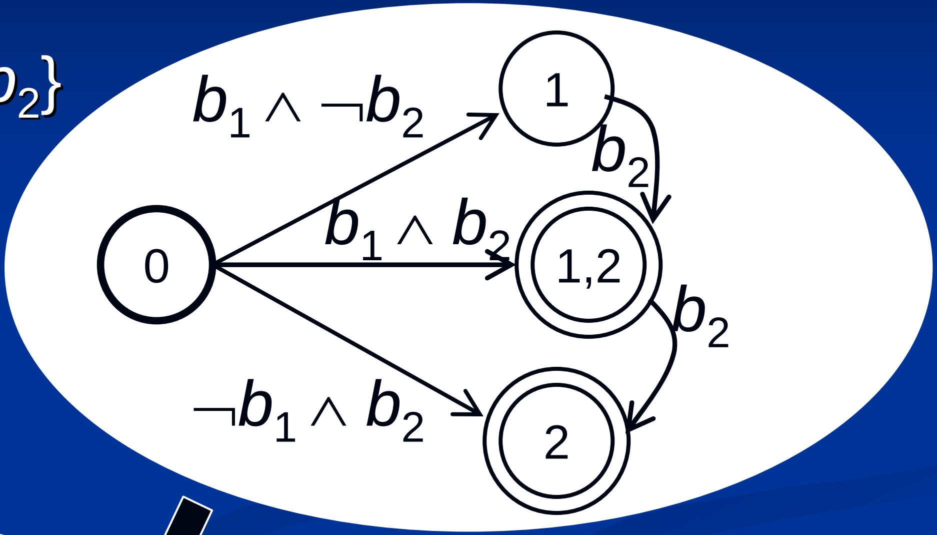
1,B: *no edges in* 1.

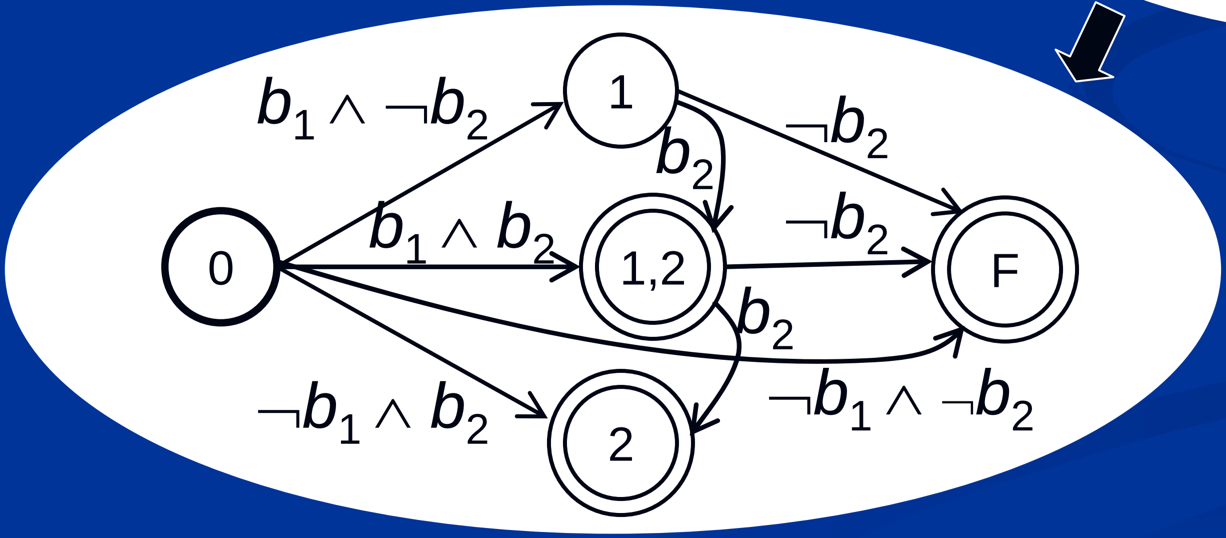
0,A:"b2 $\wedge$ b3" 1,B; "b1 $\wedge$ b3" 0,B

Obligation Mode SEREs in Properties – FirstFail()

- Example for $\{b_1[*0{:}1]; b_2\}$





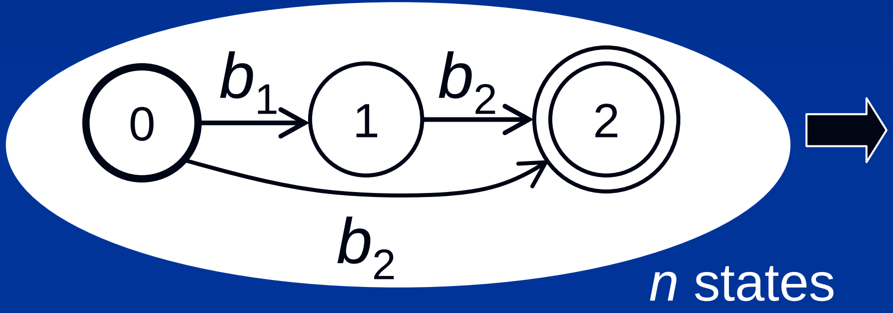


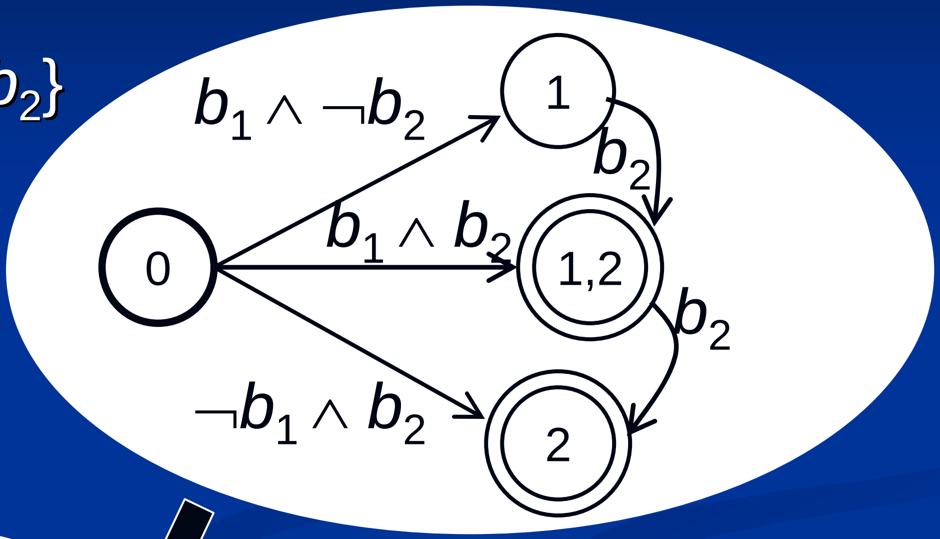
Strong Determinization
Worst case $O(e^n)$

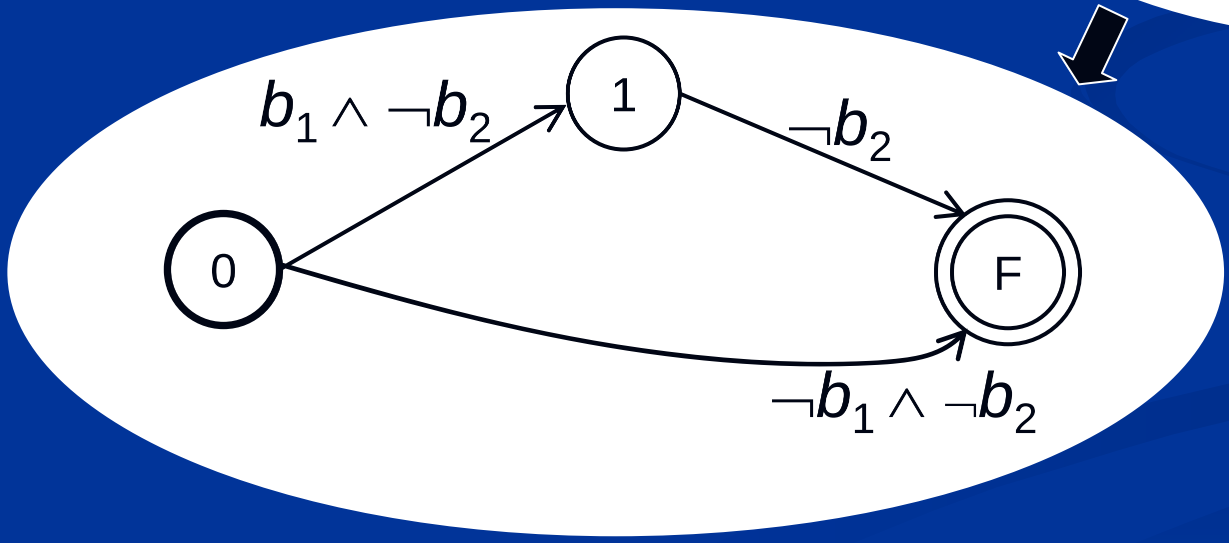
Pseudo Negation:
- Failure Conditions

Obligation Mode SEREs in Properties – FirstFail()

- Example for $\{b_1[*0:1]; b_2\}$







Strong Determinization

Worst case $O(e^n)$

Pseudo Negation:

- Failure Conditions
- Remove old final states

Experimental Results

Properties (Xilinx 8.1.03i for XC2V1500-6)	MBAC			IBM FoCs 2.03		
	FF	LUT	MHz	FF	LUT	MHz
never { a;d;{b;a}[*2:4];c;d }	**12**	**12**	622	25	24	622
never { {a[*];b[*1:3]} | {c;d[*1:2];e} }	**4**	**4**	**622**	24	23	454
never { {[*];a} && {b[=0]} }	**1**	**2**	**N.A.**	6	4	622
never { a ; {b;c;d} & {e;b;a;d} ; a }	**6**	**6**	**680**	13	12	622
never { {a[*]} : {b[*]} }	**1**	**2**	**680**	7	7	483
always {a} |=> { {b;c;d} & {e;d;b} }	**4**	**6**	**483**	No Output		
always {a} |=> { e;d;{b;e}[*2:4];c;d }	**15**	**21**	**378**	No Output		
always {a} |=> { b ; {c[*0:4]} & {d} ; e }	7	11	**487**	7	**10**	359
always {a} |=> { b ; {c[*0:6]} & {d} ; e }	**9**	**15**	**428**	No Output		
always {a} |=> {{{c;d}[+]} && {e[–>2]} }	**5**	**7**	**517**	6	10	425

Conclusion

- Introduced an efficient <u>automaton-based</u> implementation of SEREs for creating checkers for dynamic verification and silicon debug
  - Boolean-expressions in automata symbols
  - Fusion and intersection algorithms
  - First failure detection algorithm for use in properties
- These techniques for SEREs + property implementation from [HLDVT'06] = efficient assertion checking circuits for PSL