Relative Timing Based Verification of Concurrent Systems

Marco A. Peña

Department of Computer Architecture

Technical University of Catalonia

Advisors: Jordi Cortadella and Enric Pastor

Outline

- PART I: Motivation and background
- PART II: Formal verification with relative timing
- PART III: Compositional verification
- PART IV: Conclusions and future work

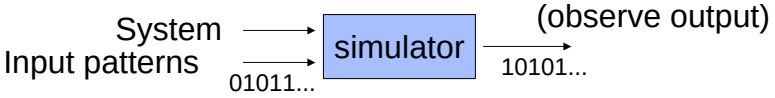
Motivation: the problem

- System's complexity: continuous growth in scale and functionality
- Probability to introduce design errors increases
- System failures are unacceptable:
  - 1994: Floating point divider unit of Pentium µP
  - 1996: Launch failure of Ariane 5 rocket
- **Simulation:** General bug-finding techniques
- **Formal verification**: Exhaustive coverage

Simulation

- Predominant verification method: intuitive idea
- Construction of test-cases: manually, randomly, etc.
- Cannot be exhaustive: "*Heisenbug*" paradigm, corner-cases, infinite cases, …

Simulation methodology:

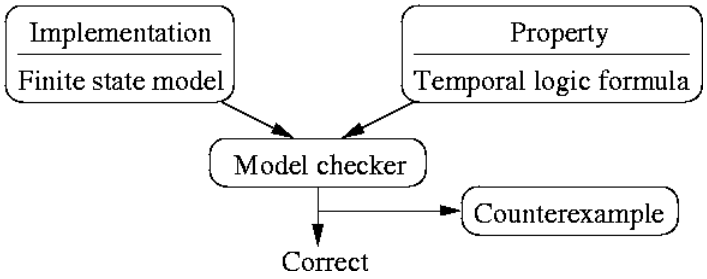


Formal verification

- **Mathematical/Algorithmic** methods to determine correctness:
  - Formal model of the system
  - Formal specification language (*e.g.* some logic)
  - Exhaustive reasoning method
- Ensures consistency with specification for **all** possible input patterns

Formal verification methodology...

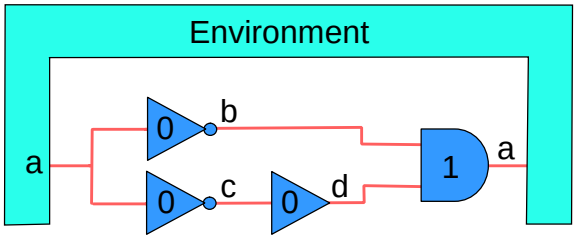
Model checking



- The checker enumerates **all** the states of the system
- Combinatorial **state explosion**: 32 bit register has 2^{32} states
- **Symbolic** methods, partial orders, abstractions, etc.
- Many automatic tools and success stories exist

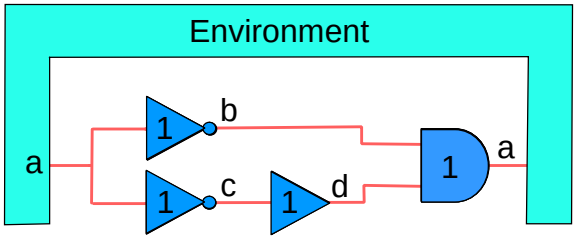
State space exploration

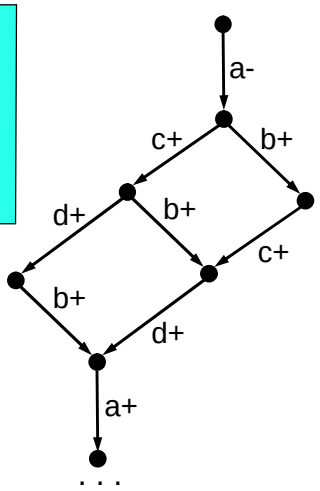
- Example:



- Initial state:
  - a = 1, b = c = d = 0
  - a is about to fall

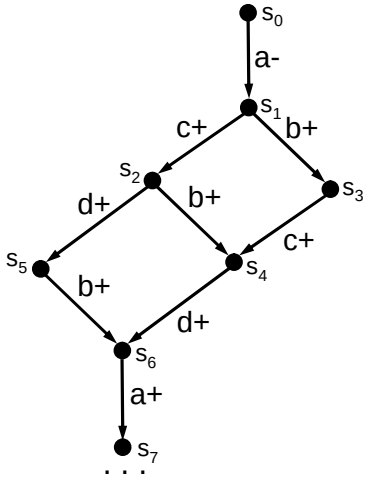
State space exploration





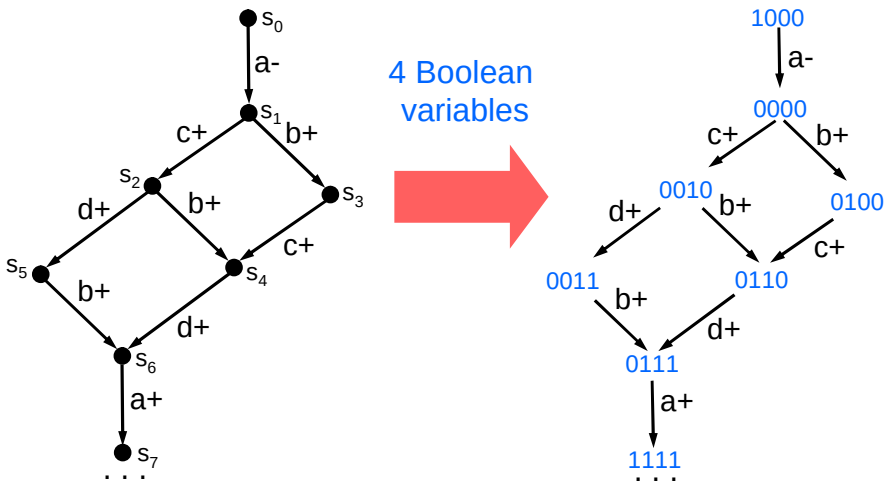
- FSM-like model: Transition system
  - State, transitions, events
- State space exploration
  - Initial state + transition relation
  - Iterative until fix-point
- Combinatorial state explosion problem

Symbolic methods

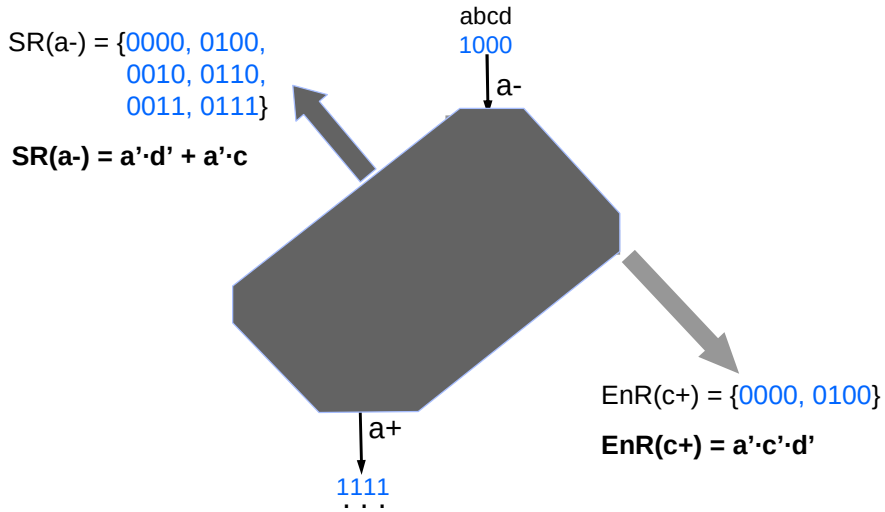


- Using $v_1 v_2 \ldots v_n$ Boolean variables, a state is a *minterm* in B^n
- Sets of states and transitions are Boolean functions in B^n
- Boolean functions represented as **Binary Decision Diagrams** (Bryant): worst-case exponential, but efficient in practice

Symbolic methods



Symbolic methods



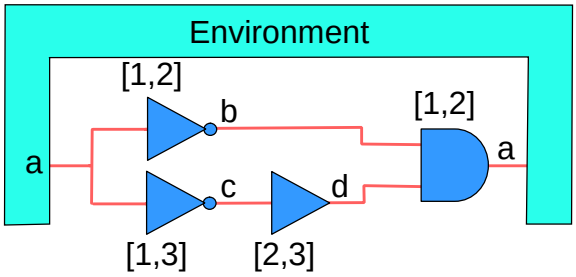
Appropriate for untimed systems…

Timed systems

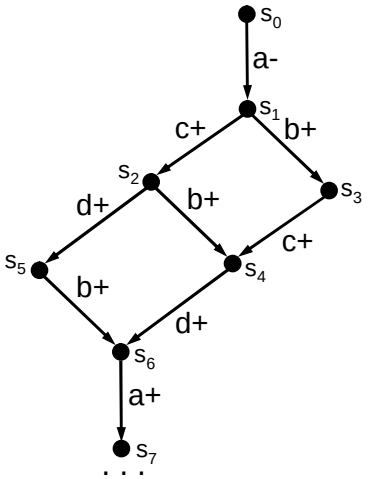
- Symbolic methods are mature for untimed systems
- Timed systems: correct behavior depends on timing characteristics of the system:
  - RTOSs: *e.g.* reaction to interrupts within a time frame
  - Communication protocols: *e.g.* time-out conditions
  - Aggressive circuit designs for performance
  - GALS-type systems: *e.g.* bridge of clock domains
- Time introduces new sources of exponentiallity
- Conventional verification techniques do not apply

Timed systems

- Example:



Timed systems



- **Timed transition system** (Manna, Pnueli)
  - Transition system
  - Min/Max delay bounds

  $d(a-) \in [1, 2]$

  $d(b+) \in [1, 2]$

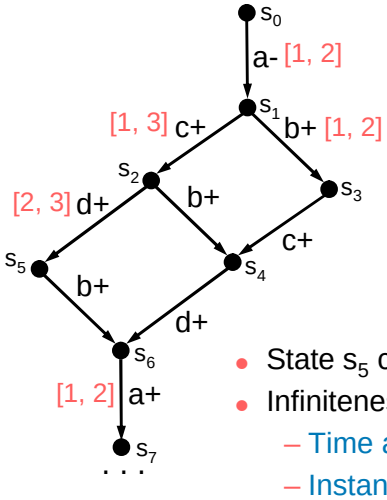
  $d(c+) \in [1, 3]$

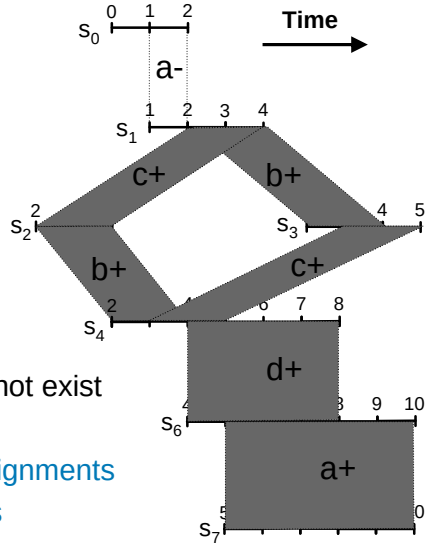
  $d(d+) \in [2, 3]$

  $d(a+) \in [1, 2]$

  . . .

Exact timed state space





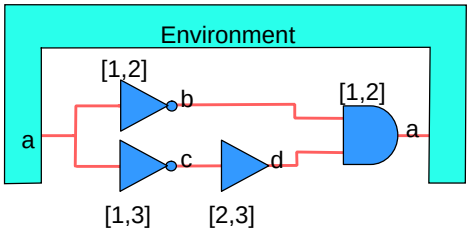
- State s_5 cannot exist
- Infiniteness:
  - Time assignments
  - Instances

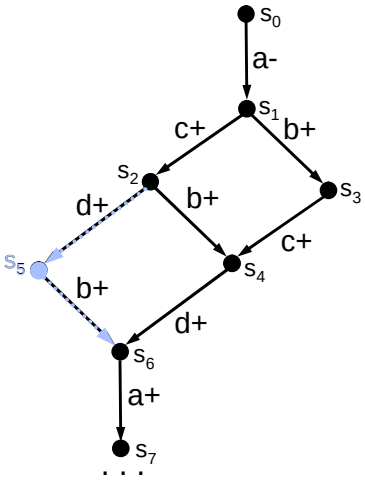
Exact timed state space

- Timed automata (Dill, Alur, Courcoubetis, …):
  - Finite automaton to describe the control
  - Real-valued ***clocks*** to model quantitative timing constraints
  - COSPAN, KRONOS, UPAAL, MOCHA, …
- *Clock* regions and region automata (Dill, Alur, …):
  - Equivalence class of *clock* valuations
  - Sensible to the number of clocks and the delay bounds
- Zone automata:
  - Collapse groups of regions into convex geometric regions
  - Difference Bound Matrices (Dill)
- **Symbolic methods not easily applicable:**
  - Difference Bound Matrices and BDDs (Dill, Balarin, …)
  - Discrete counters and NDDs: OpenKRONOS (Bozga, Maler, …)

Relative timing

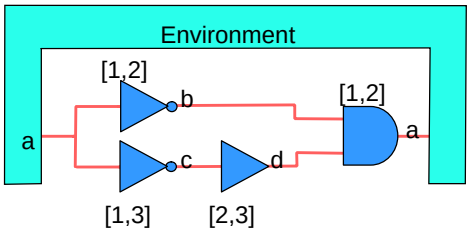
Relative timing

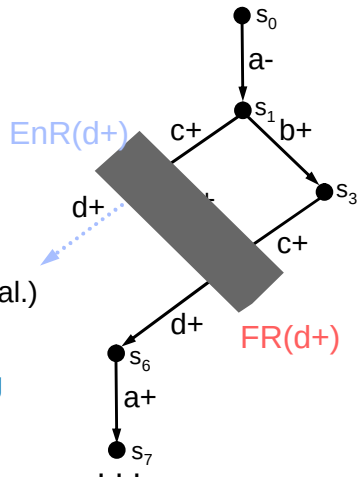




- Delays impose:
  - **d+** cannot fire in s_2
  - **d+** is only firable in s_4
  - **b+** always fires before **d+**
- Concurrency reduction by imposing relative orderings

Relative timing





- Lazy Transition System (Cortadella, et al.)
  - **d+** is **lazy** w.r.t. **b+**
  - Explicit distinction between enabling and firing
  - Conservative overestimation of the timed state space

GOAL of the thesis

GOAL of the thesis

- "Model-checking-like" approach for the verification of **safety** properties in **timed** systems
  - Safety properties (invariants): "an undesired behavior *never occurs*"
  - Satisfiability of the properties depends of timing characteristics of the system
  - Example: no spurious/undesired events in a circuit

- Some problems:
  - State explosion
  - Time dimension
  - Algorithmic complexity

- Some solutions:
  - Symbolic representation
  - Relative timing
  - Incremental approach

GOAL of the thesis

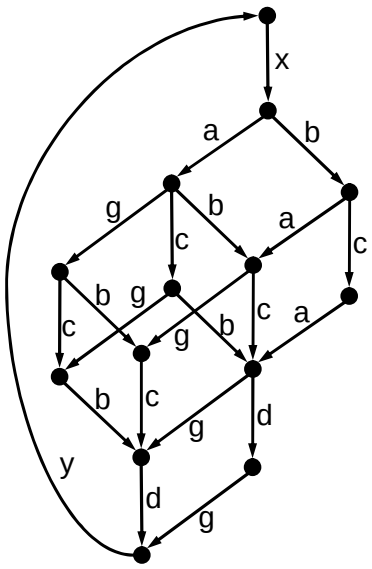
- System model: Timed Transition Systems (TTS)
- Safety properties (modeled as Boolean invariants)
- Relative timing: Lazy Transition Systems (LzTS)
- LzTSs can be represented using BDDs ⇒ large systems
- Iterative **incremental refinement** of the untimed domain:
  - Localized analysis of property violations,
  - Off-line absolute timing analysis, and
  - Incorporation of **Relative Timing** constraints
- **Back-annotation**: counterexample trace or sufficient relative timing constraints for correctness are reported

PART II

Formal verification with relative timing

- System model
- From absolute to relative timing
- State space refinement by timing constraints
- Overall verification approach
- Experimental results

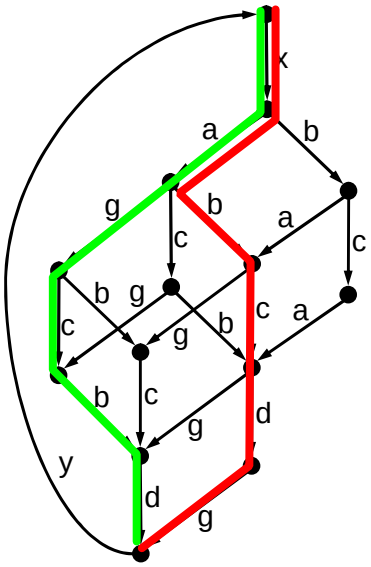
System model



- Timed Transition system
  - Transition system
  - Min/Max delay bounds

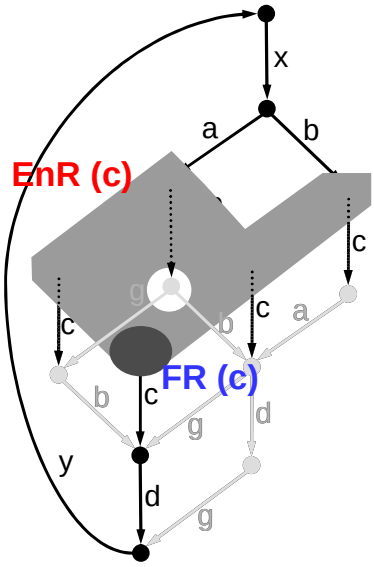
$d(a) \in [1,2]$
$d(b) \in [1,2]$
$d(c) \in [2.5,3]$
$d(g) \in [0.5,0.5]$
$d(d,x,y) \in [0,\infty)$

Safety properties



- Many properties can be expressed as relative orderings of events
- Example:
  - **g** must fire before **d** after having fired **x**
- "Good traces"
- "Bad traces"

System model



- Timed Transition system
  - Transition system
  - Min/Max delay bounds

$d(a) \in [1,2]$
$d(b) \in [1,2]$
$d(c) \in [2.5,3]$
$d(g) \in [0.5,0.5]$
$d(d,x,y) \in [0,\infty)$

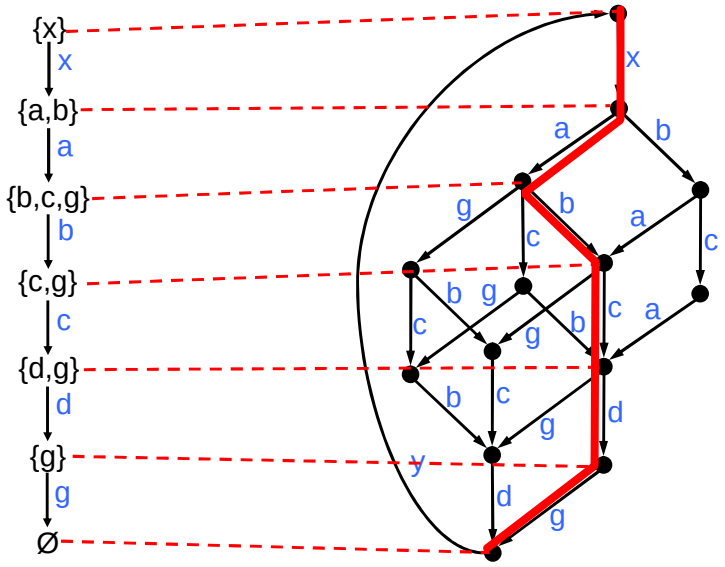
- Lazy Transition system

From absolute to relative timing

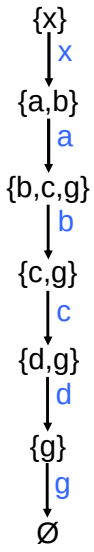
From absolute to relative timing

- Firing times depend on the enabling and the delays
- Enabling information can be extracted from a trace:
  - Event Structure (ES)
  - An ES covers other traces *modulo* concurrency
  - Key notion: **enabling compatibility**
- Timing analysis on an ES:
  - Relative timing relations $\Rightarrow$ Timed ES
- Main result: **only traces enabling compatible with the timed ES exist in the timed domain**

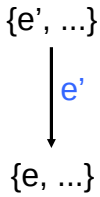
Enabling information



Enabling information

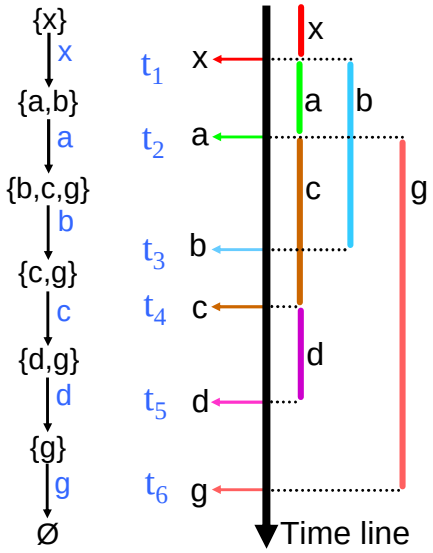


- An event **e** can only become enabled at the time another event **e'** fires (**e'** triggers **e**)



- The order of the enabling implies the firing times of the events

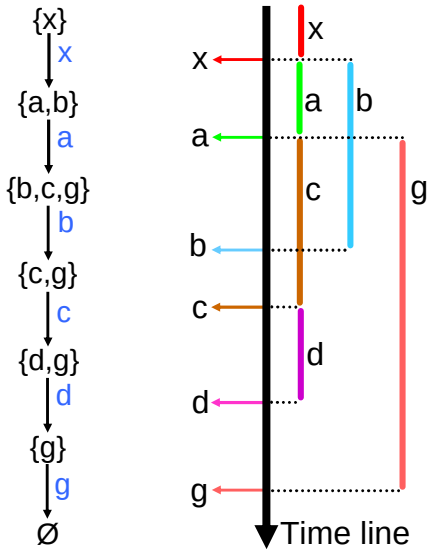
Timing consistent trace

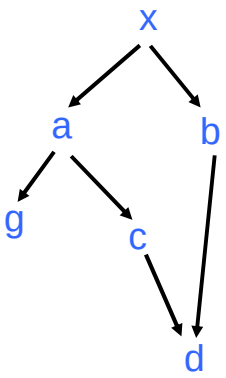


Time assignment to event firings such that:

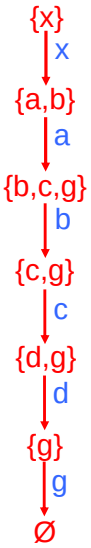
$$d_{min}(g) \leq t_6 - t_2 \leq d_{max}(g)$$

Event structure from a trace

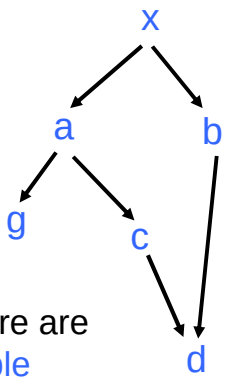




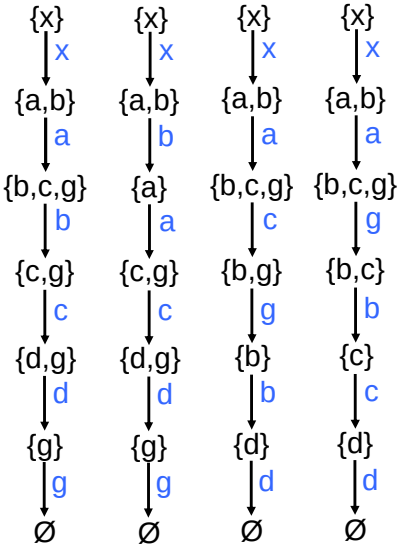
Enabling compatibility

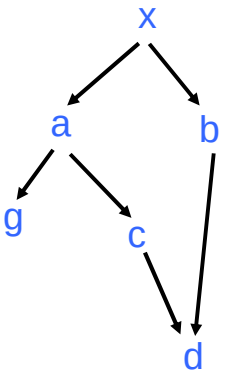


Trace and event structure are enabling compatible



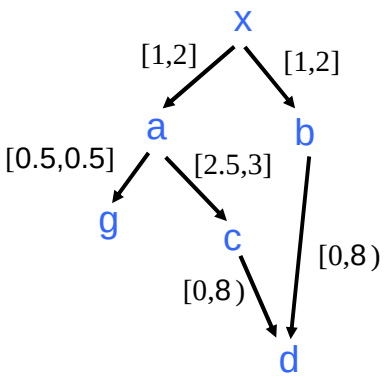
Enabling compatibility





Timing analysis (time separation)

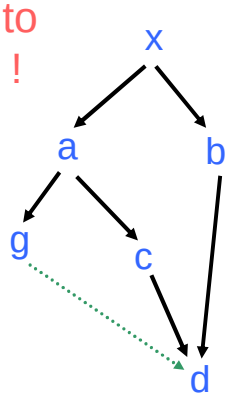
- McMillan, Dill (1992): min/max constraints in acyclic graphs
- Hulgaard, Burns (1994): max constraints for cyclic graphs with choice
- Chakraborty (1997): linear constraints and approximate methods



max $\tau(g) - \tau(d) = -2$

Timing analysis (time separation)

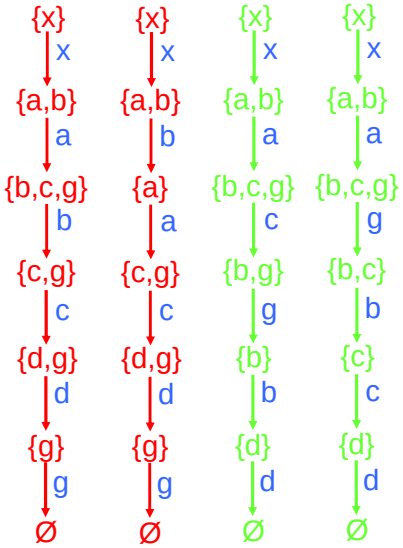
From absolute to relative timing !

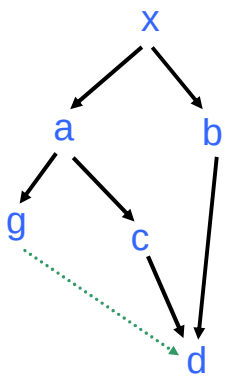


Timed Event Structure

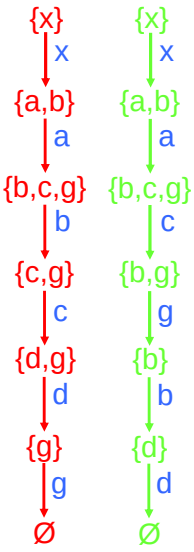
max $\tau(g) - \tau(d) = -2$

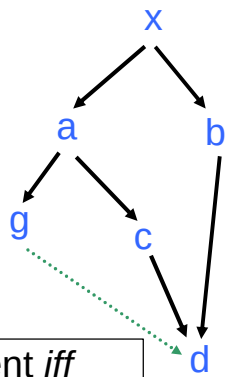
Enabling compatibility & Timing consistency





Enabling compatibility & Timing consistency



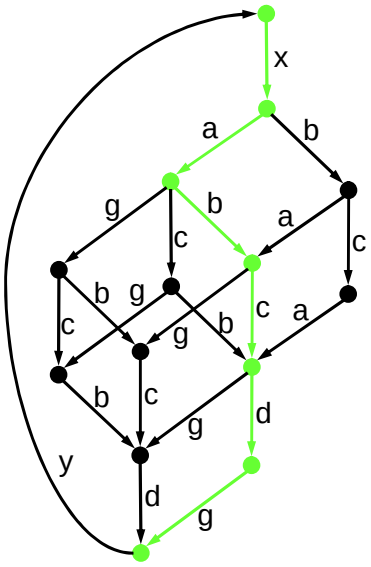


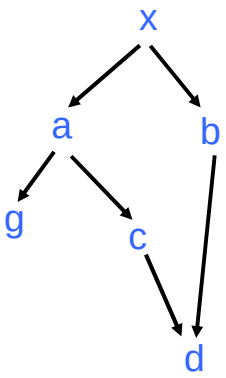
Theorem:

A trace is timing consistent *iff* it is an enabling compatible trace of the **timed** event structure

State space refinement by
timing constraints:
enabling compatible product

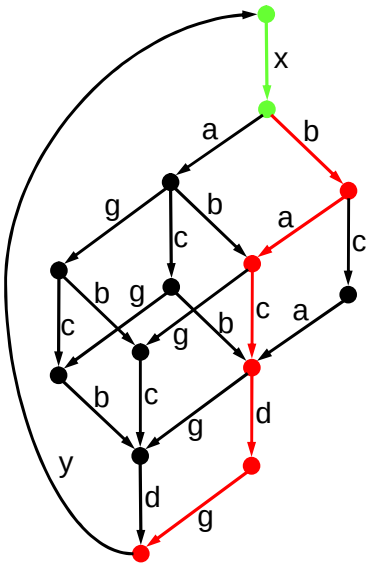
Enabling compatibility

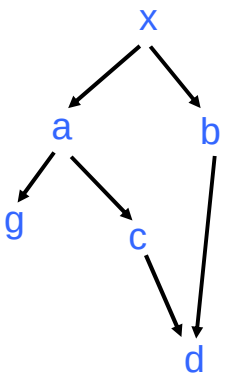




Enabling compatible

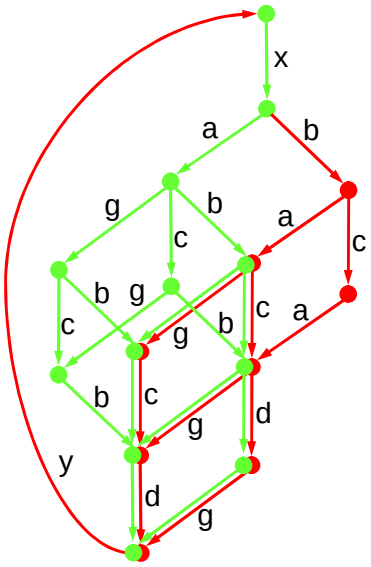
Enabling compatibility

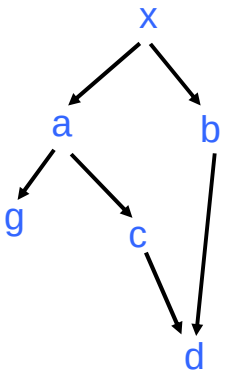




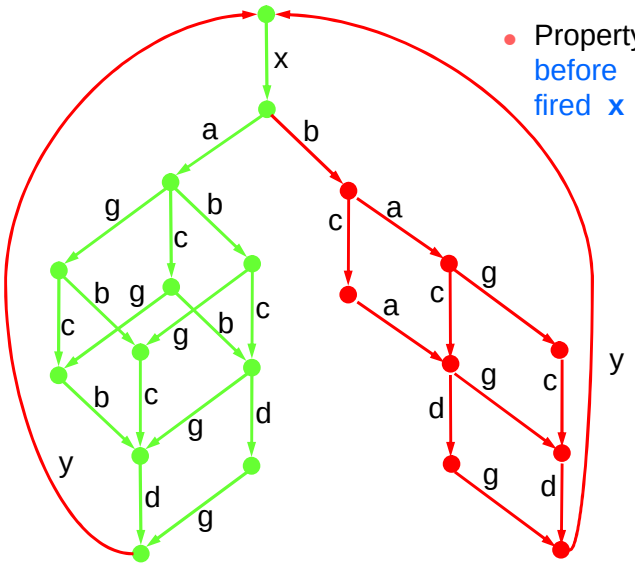
Not enabling compatible

State space partition



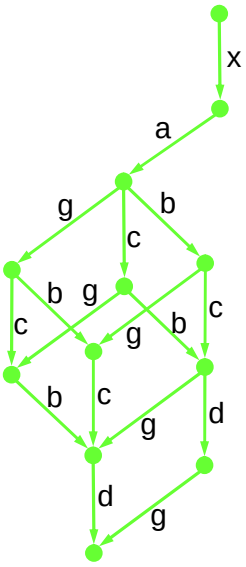


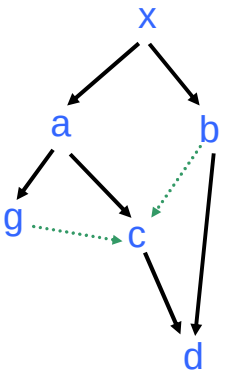
State space partition



- Property: **g** must fire before **d** after having fired **x**

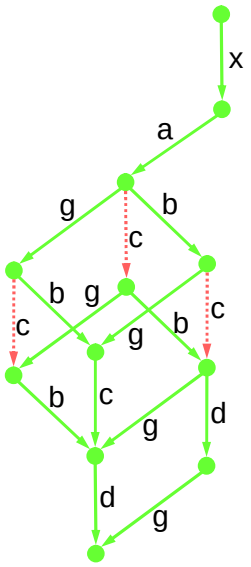
State space partition & Timing analysis

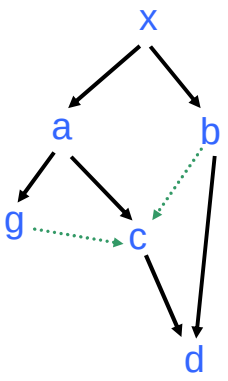




Timing analysis

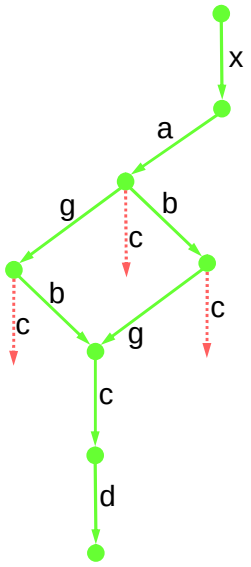
State space partition & Timing analysis

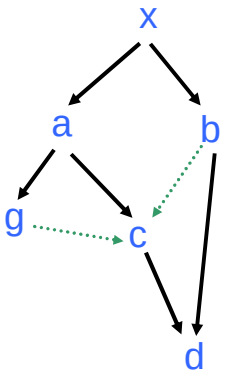




Timing analysis

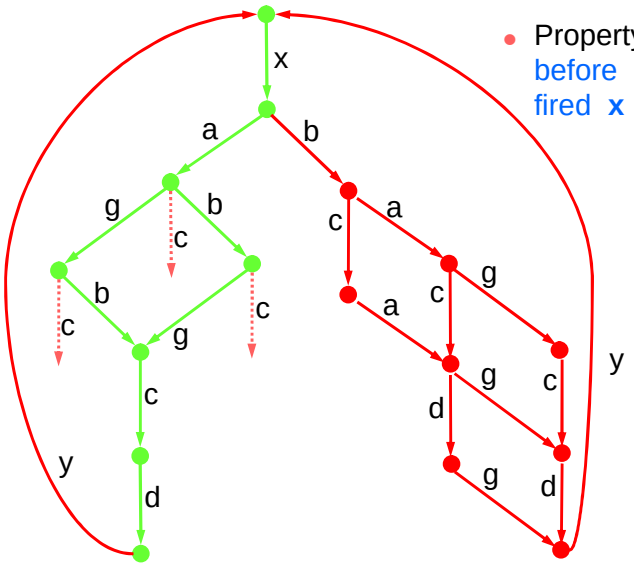
State space partition & Timing analysis





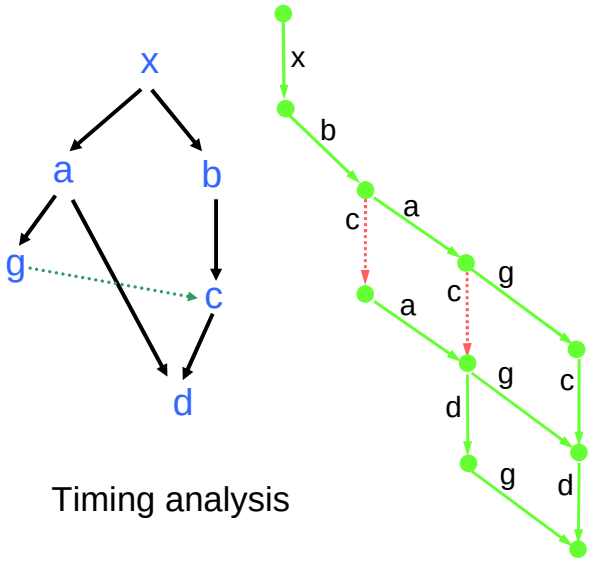
Timing analysis

State space partition & Timing analysis



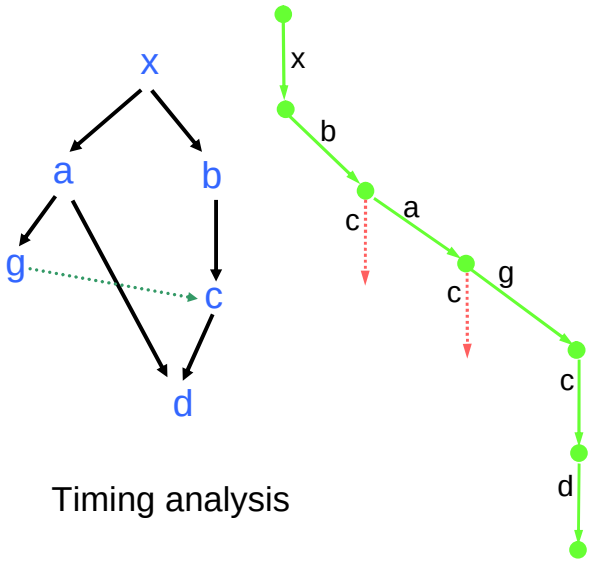
- Property: **g** must fire before **d** after having fired **x**

State space partition & Timing analysis



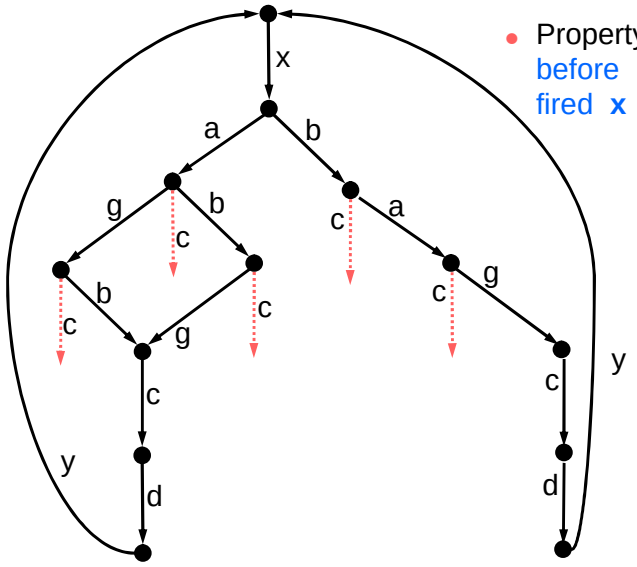
Timing analysis

State space partition & Timing analysis



Timing analysis

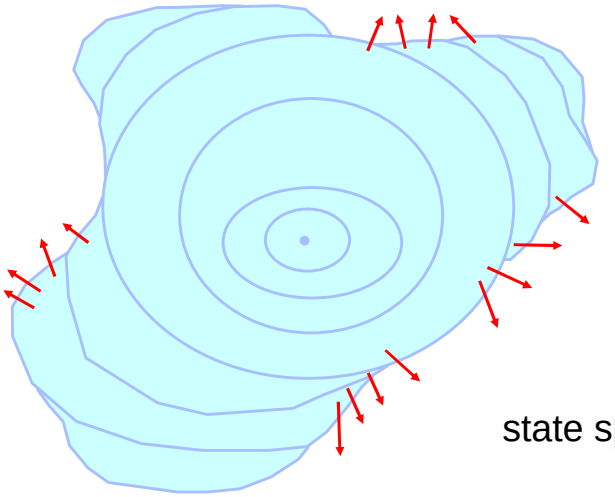
State space partition & Timing analysis



- Property: **g** must fire before **d** after having fired **x**

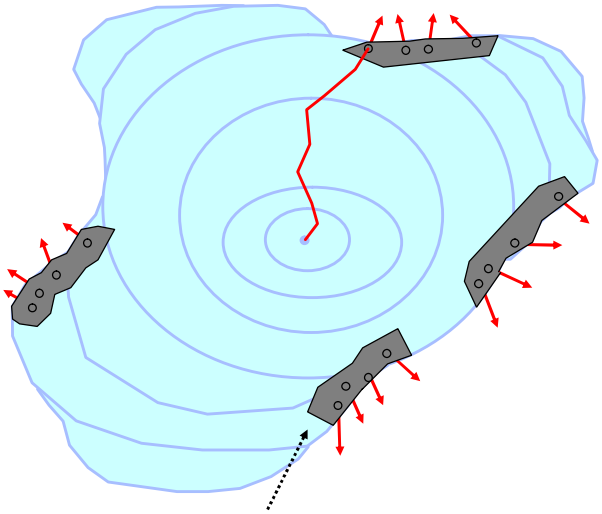
Overall verification approach

Verification approach

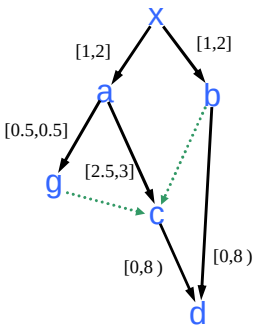


Symbolic
state space exploration
and
failure detection

Verification approach: iterative refinement

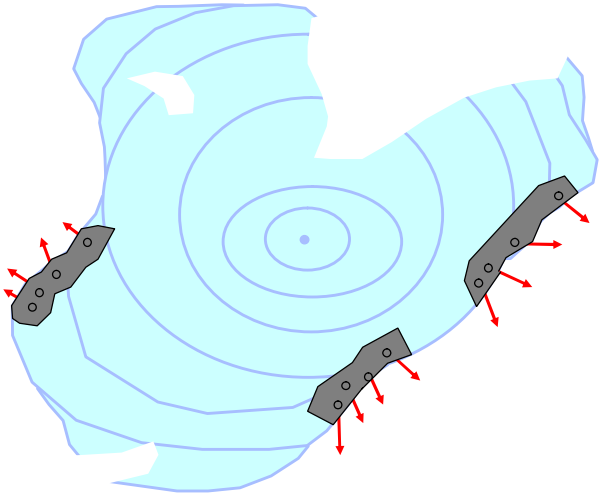


- Failure trace
- Event structure

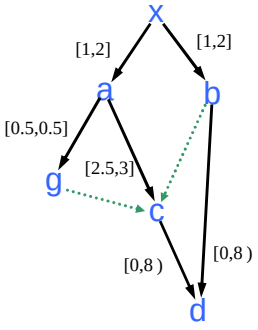


- Timing analysis
- Composition

Verification approach: iterative refinement

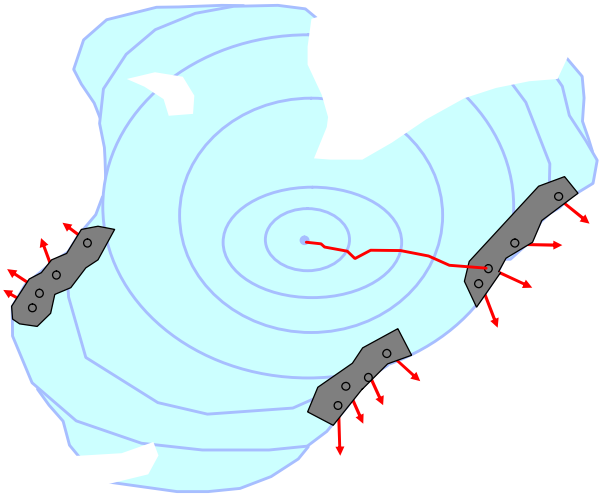


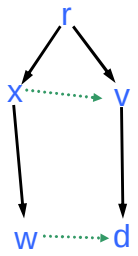
- Failure trace
- Event structure



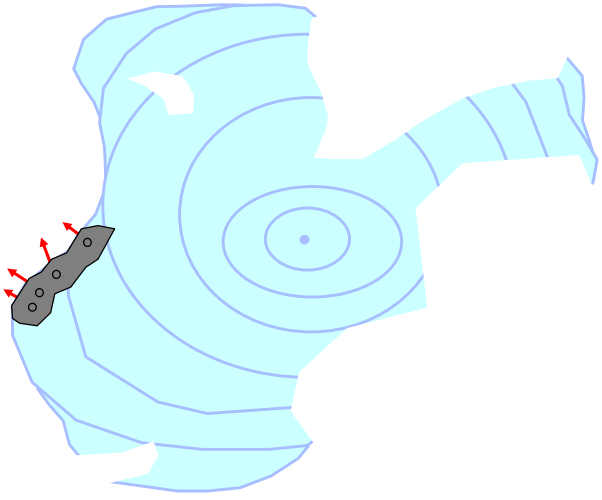
- Timing analysis
- Composition

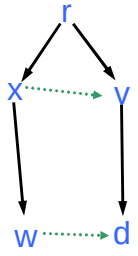
Verification approach: iterative refinement



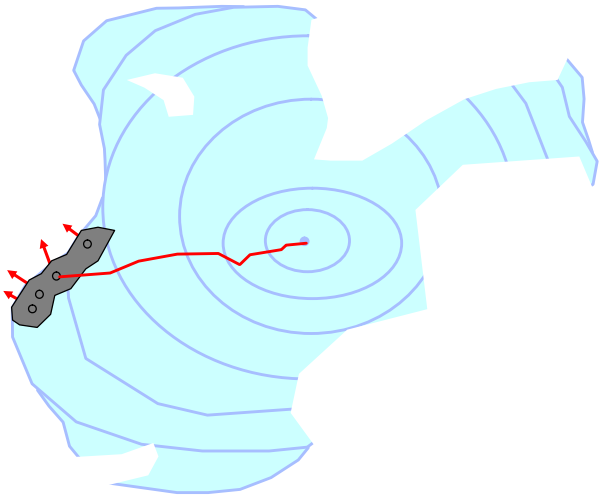


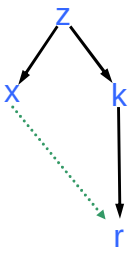
Verification approach: iterative refinement



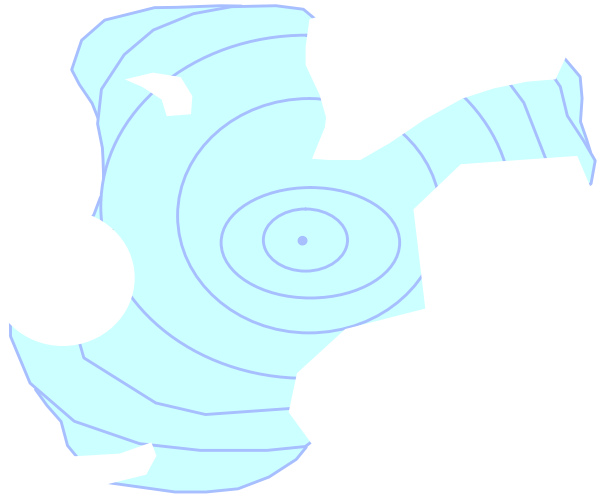


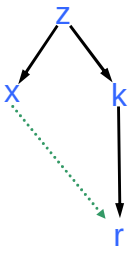
Verification approach: iterative refinement



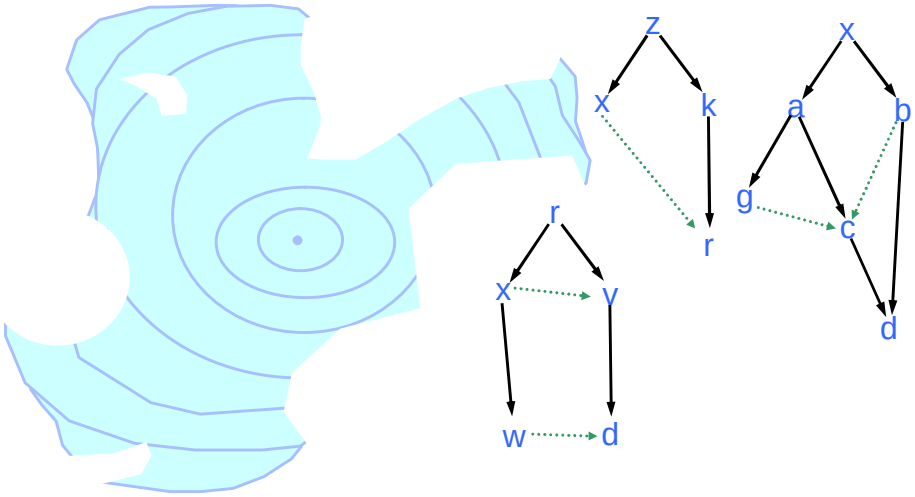


Verification approach: iterative refinement



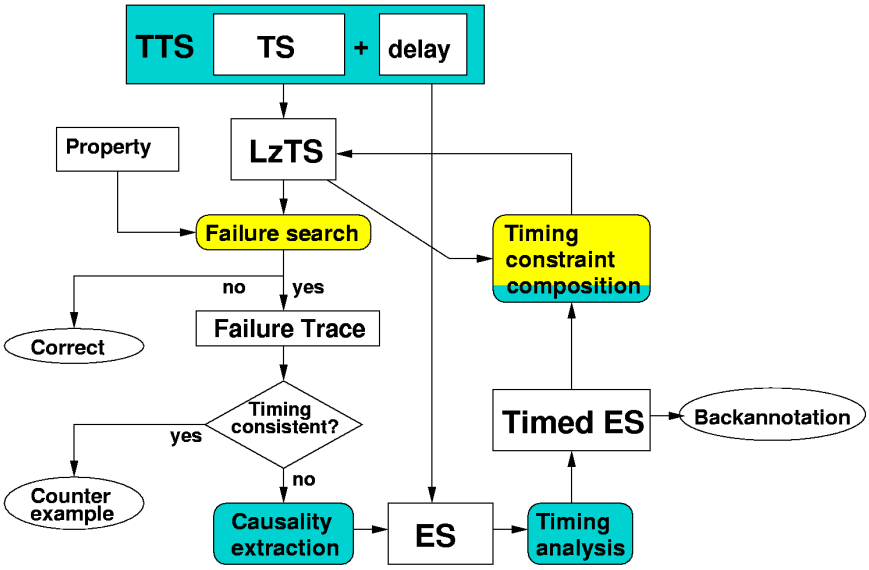


Verification approach: back-annotation

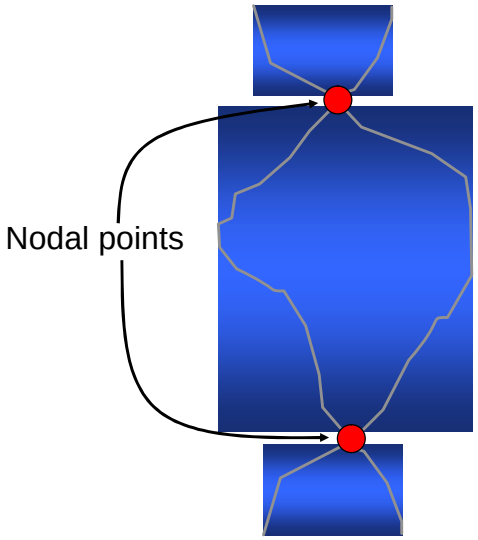


Back-annotation: sufficient timing constraints

Verification approach: flow



Verification approach: convergence



All cycles cut by nodal points

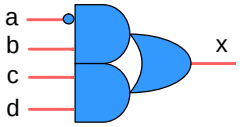
Finite number of traces between nodal points

Convergence and exact results guaranteed

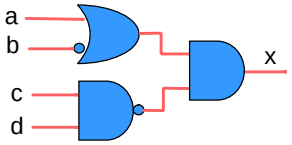
Experimental results

Experiments: gate decompositions

- Complex-gate decompositions in *Speed-Independent* asynchronous circuits :
  - Robust hazard-free operation regardless of gate delays



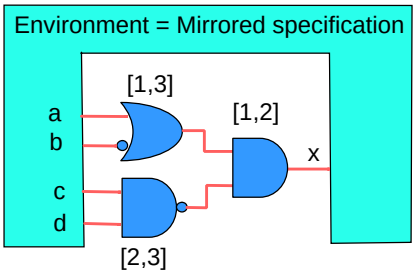
Hazard-free



Non hazard-free
Unless timing assumptions

Experiments: set-up

- Verification of asynchronous control circuits
- Library delays assigned to gates
- Closed system: circuit vs. specification



- Verification:
  - Input-output conformance w.r.t. the specification
  - Absence of internal hazards

Experiments: results

name	gates	untimed	failure	iters.	correct ?	CPU
sbuf-read-ctl	10	74	16	3	Y	1
rcv-setup	6	78	34	2	N	1
alloc-outbound	11	82	20	4	Y	3
ebergen	9	83	22	1	N	1
mp-forward-pkt	8	186	70	8	Y	5
dff	6	255	164	6	N	2
half	7	227	133	1	N	1
chu133	9	288	204	2	N	1
converta	12	408	244	10	N	12
nowick	10	510	292	4	Y	3
chu150	8	520	339	3	N	1
sbuf-send-ctl	13	1592	1081	18	N	54
rpdtf	8	2612	1841	2	N	2
sbuf-send-pkt2	13	4544	4044	[illegible]	[illegible]	103
vme	15	10568	8655	[illegible]	[illegible]	30
sbuf-ram-write	15	14016	12362	[illegible]	[illegible]	415
ram-read-sbuf	16	19328	17488	[illegible]	[illegible]	550
mr1	16	[illegible]	11574	[illegible]	[illegible]	317
mr0	20	[illegible]	642291	2	N	48
tsend-bm	12	[illegible]	717561	3	N	185
trimos-send	24	[illegible]	1.8 E6	1	N	127
mmu	22	[illegible]	5.2 E6	3	N	480

TranSyT

- PIII 866MHz
- 1GB RAM
- 256MB peak
- No specific strategy
- Back-annotation

Experiments: results comparison

name	untimed	iters.	CPU	CPU
sbuf-read-ctl	74	3	1	1
rcv-setup	78	2	1	1
alloc-outbound	82	4	3	1
ebergen	83	1	1	1
mp-forward-pkt	186	8	5	1
dff	255	6	2	1
half	227	1	1	1
chu133	288	2	1	1
converta	408	10	12	1
nowick	510	4	3	1
chu150	520	3	1	1
sbuf-send-ctl	1592	18	54	1
rpdtf	2612	2	2	1
sbuf-send-pkt2	4544	19	103	3
vme	10568	21	30	1
sbuf-ram-write	14016	34	415	32
ram-read-sbuf	19328	36	550	826
mr1	21076	29	317	Memory
mr0	727304	2	48	Memory
tsend-bm	763608	3	185	Memory
trimos-send	2.1 E6	1	127	Memory
mmu	5.6 E6	3	480	Memory

OpenKronos
(Bozga et al. 2002):

- Timed automata
- Discrete *clocks* as counters
- NDDs
- UltraSparc
- 2GB RAM !
- No back-annotation

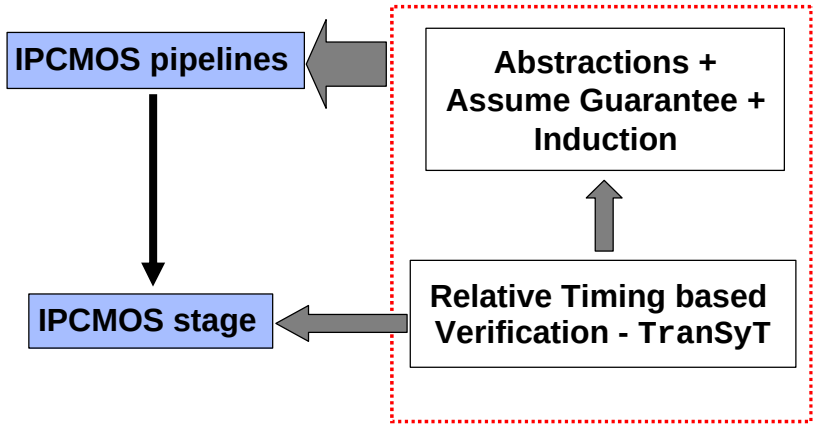
PART III

Compositional verification

- IPCMOS architecture
- Verification of IPCMOS pipelines
  - Strategy
  - Abstractions
  - Assume-Guarantee and induction
  - Results

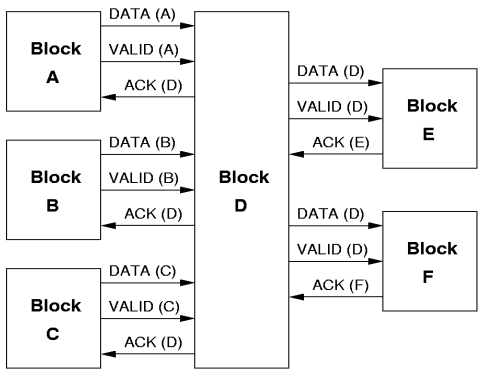
Goal

GOAL: Formal verification of a complex timed design (IPCMOS architecture)



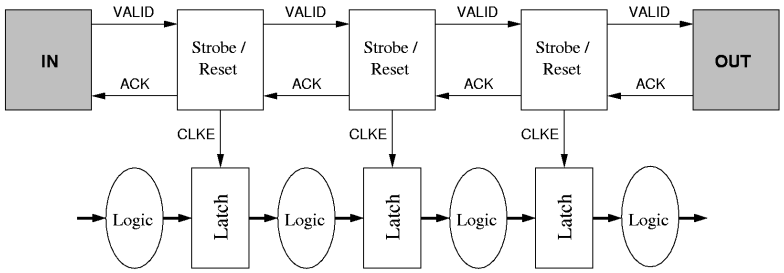
IPCMOS architecture

General IPCMOS architecture



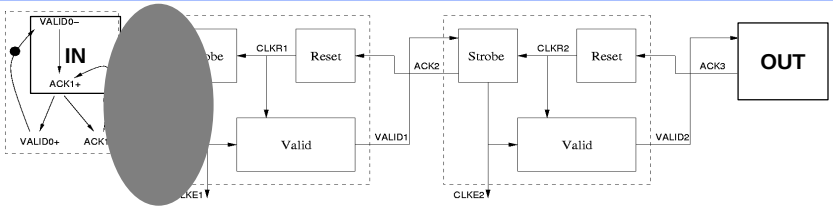
- **Pulse-based** asynchronous clocking technique for large devices operating at GHz frequencies
- Block-level interlocking scheme ⇒ scalable
- Schuster, et al. (IBM) ISSCC 2000

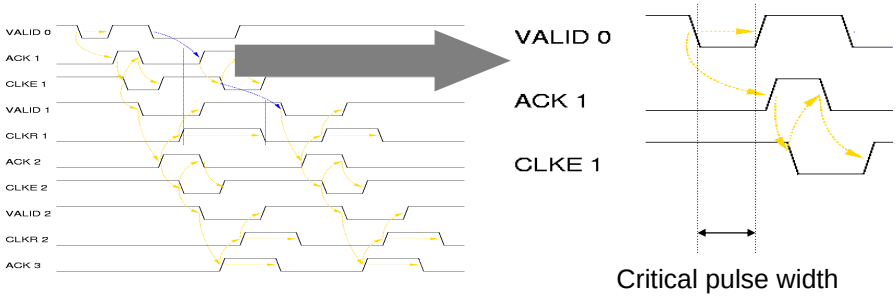
Linear IPCMOS pipeline architecture



- 2-phase request (**VALID**) and acknowledge (**ACK**) protocol
- **VALID** : data is available to the next block, mimics delay of the logic
- **ACK** : data received by the next block
- Performance up to 4GHz (year 2000 technology)
- Correctness depends on pulse widths

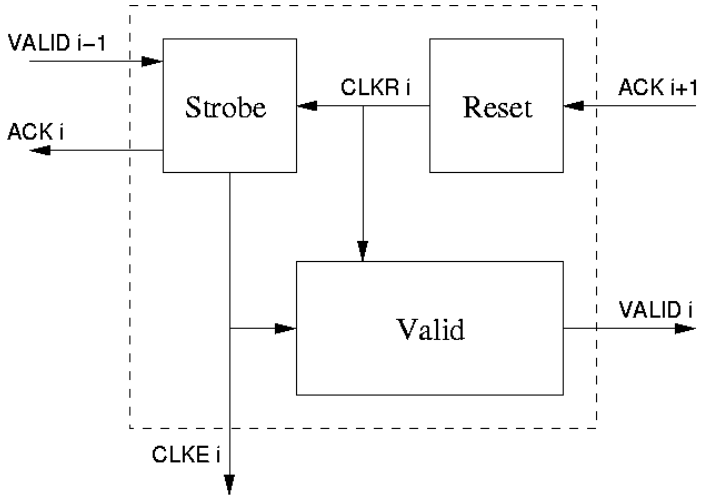
Two-stage pipeline at work



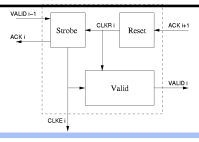


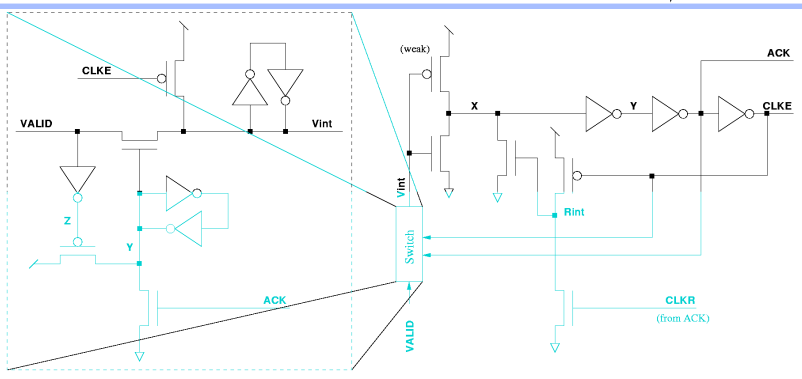
Critical pulse width

Stage building blocks



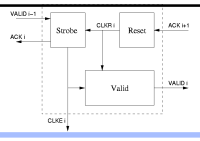
Strobe circuit

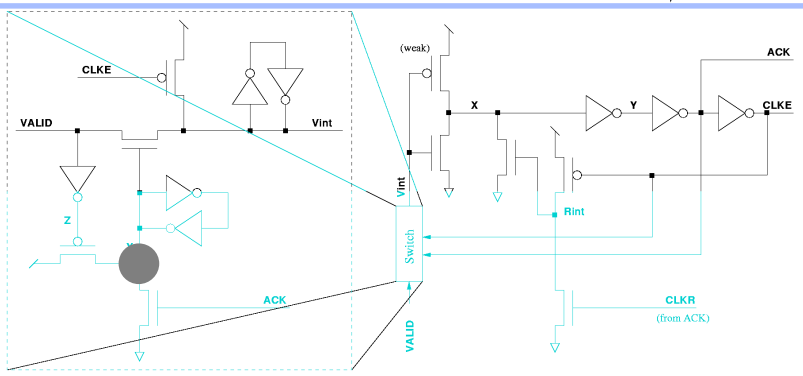




- Capture a negative pulse at **VALID** from the previous stage and produce a positive pulse at **ACK** to the next stage and a negative pulse at **CLKE** to the functional unit

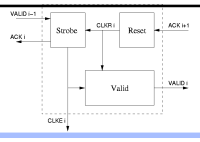
Strobe circuit

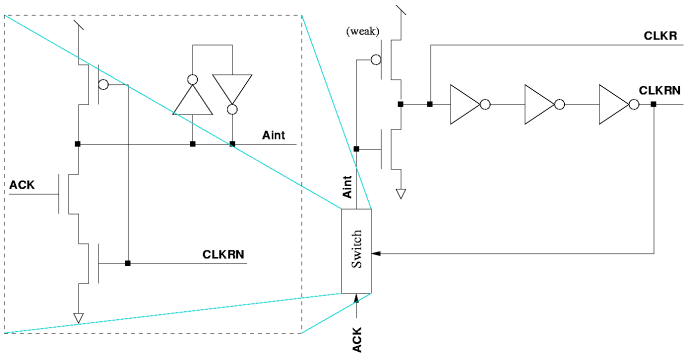




- Transition relations:
  - En(Y+) : ¬Y · ¬Z
  - En(Y-) : Y · ACK
  - . . .
- Failure conditions:
  - Shortcut at Y : ¬Z · ACK
  - . . .

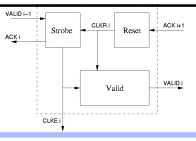
Reset circuit

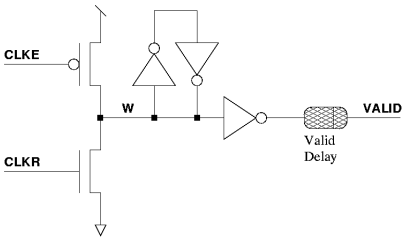




- Capture a positive pulse at **ACK** from the next stage and produce a positive pulse at **CLKR** and a negative pulse at **CLKRN** to reset both the strobe and valid circuits

Valid circuit

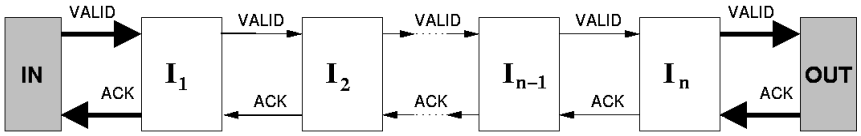




- Capture negative pulses at **CLKE** and produce a negative pulse at **VALID**. The pulse is reset by a positive pulse at **CLKR**
- Delay after the inverter mimics the delay of the functional block controlled by the stage

Verification of IPCMOS pipelines

Verification goal



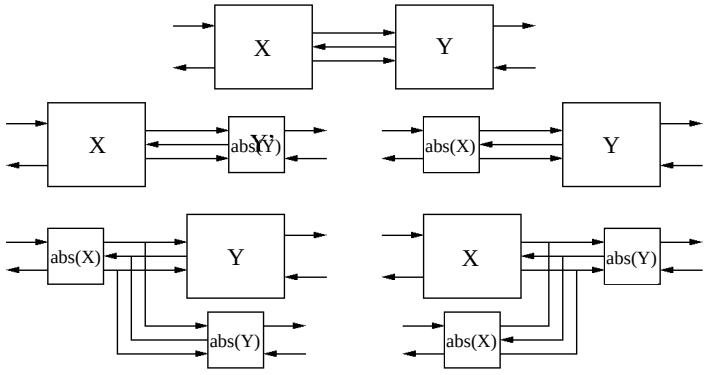
- Assuming data-path is correct, the pipeline is correct (spec.):

  S = “Every data fed into de pipeline is acknowledged once and only once at every stage”

- *S* is modeled by a deadlock condition plus correctness of CMOS circuits: no short-circuits, etc.
- Correctness regardless of the length of the pipeline

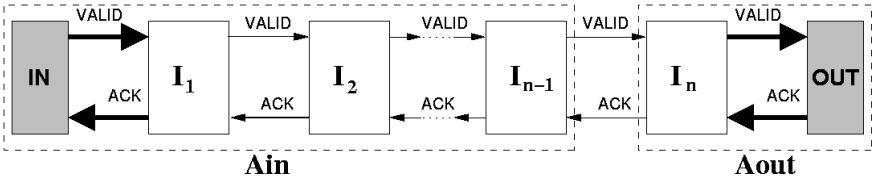
$$IN \parallel I_1 \parallel \ldots \parallel I_n \parallel OUT \leq S \ , \ n \geq 1$$

Assume-Guarantee verification



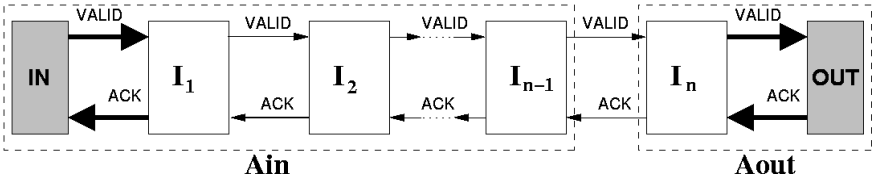
- Pnueli 1984, Clarke et al. 1989, etc.
- Abstractions to overcome complexity: preserve the input/output behavior and the properties of interest
- ***Assume*** the abstractions are correct
- Prove that the abstractions are correct to ***guarantee*** a sound analysis

Verification strategy



- Key observation:
  - Pulse-based communication only at the extremes
  - Internal communication is time-independent, *i.e.* 2-phase handshaking
- Allows:
  - Untimed abstractions
  - Assume-guarantee

Verification strategy

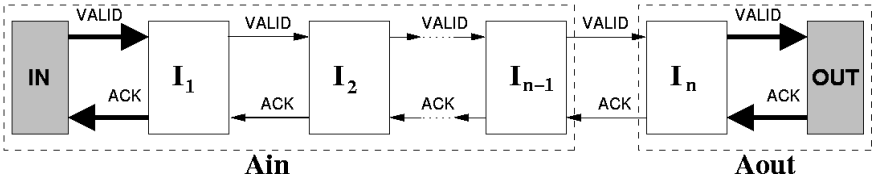


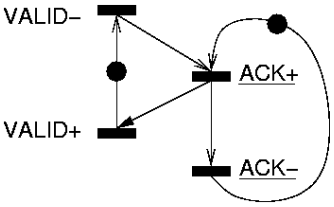
- A_{in} and A_{out} are untimed abstractions that hide the pulse-based behavior
- **Assume**: pose verification in terms of: $A_{in} \parallel A_{out} \leq S$
- **Guarantee** soundness of the abstractions:

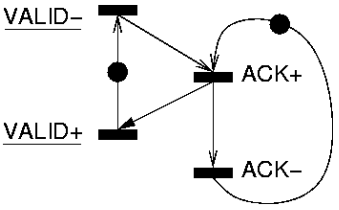
$$IN \parallel I_1 \parallel \ldots \parallel I_n \parallel OUT \leq A_{in} \parallel A_{out}$$

- Prove correctness of a one-stage pipeline

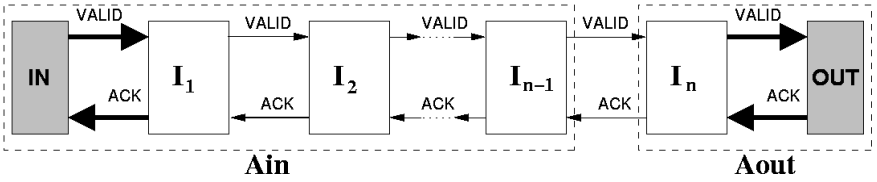
Abstractions



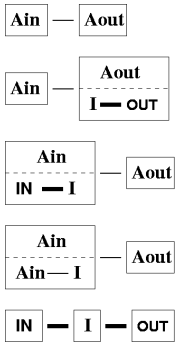




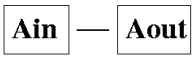
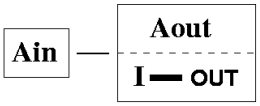
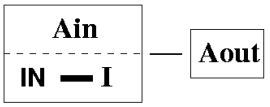
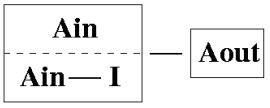
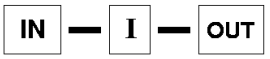
Assume-guarantee strategy



- Assume: $A_{in} \parallel A_{out} \leq S$
- Guarantee correctness of A_{out}
- Guarantee correctness of A_{in}
- Guarantee correctness of A_{in} (induction)
- Guarantee correctness of 1-stage: IN || I || OUT $\leq S$



Assume-guarantee: results

	CPU time	Refinements	
$A_{in} \parallel A_{out} \leq S$	*1m*	*--*	
$A_{in} \parallel I \parallel OUT \leq A_{in} \parallel A_{out}$	*28m*	*7*	
$IN \parallel I \parallel A_{out} \leq A_{in} \parallel A_{out}$	*9m*	*3*	
$A_{in} \parallel I \parallel A_{out} \leq A_{in} \parallel A_{out}$	*10m*	*3*	
$IN \parallel I \parallel OUT \leq S$	*35m*	*40*	

PART IV

Conclusions and future work

Conclusions

- New technique for the verification of timed systems
  - Relative timing
  - Avoids generating the exact timed state space
  - Symbolic methods ⇒ bigger systems
- Key notion: enabling compatibility
  - Timing added on-demand: Incremental refinement
  - Back-annotation: sufficient conditions for correctness
- Combined with compositional reasoning to overcome complexity: IPCMOS case study

Publications

- "Formal Verification of Safety Properties in Timed Circuits", Int. Conf. on Advanced Research in Asynchronous Circuits and Systems (ASYNC), 2000.
- "A Case Study for the Verification of Complex Timed Circuits: IPCMOS", Design Automation and Test Europe (DATE), 2002.
- "Relative timing based verification of concurrent systems", Ph.D. Forum at Design Automation and Test Europe (DATE), 2003.

Future work

- Improvements:
  - Study sensibility to selection of traces, ESs, …
  - Consider OR-causality in ESs, timing analysis, …
  - Faster convergence and better back-annotation
  - Enabling compatible product: critical for performance
- Long-term:
  - Beyond safety properties, temporal logic, …
  - Automatic derivation of symbolic timing constraints
  - Hierarchical verification and compositional reasoning

Questions?