Embedded Systems 24

REVIEW: Testing

Testing includes

- the application of test patterns to the inputs of the device under test (DUT) and
- the observation of the results.

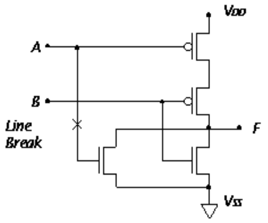
More precisely, testing requires the following steps:

1. test pattern generation,
2. test pattern application,
3. response observation, and
4. result comparison (okay, not okay, *inconclusive*).

REVIEW: Hardware Fault models

Hardware fault models include:

- **stuck-at fault** model (net permanently connected to ground or V_{dd})
- **stuck-open** faults: for CMOS, open transistors can behave like memories
- **delay** faults: circuit is functionally correct, but the delay is not.



- *Break above results in a "memory-effect" in the behavior of the circuit*
- *With AB=10, there is not path from either VDD or VSS to the output*
  - *F retains the previous value for some undetermined discharge time*

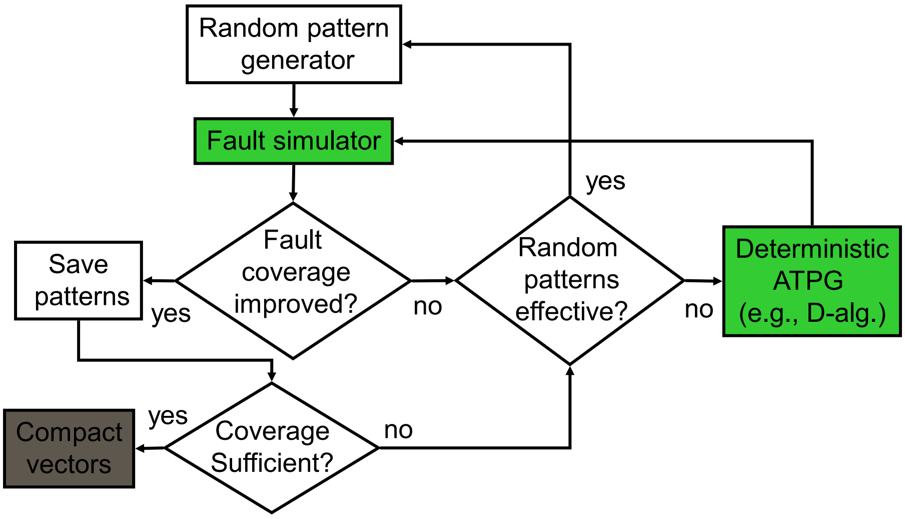
www.cedcc.psu.edu/ee497f

/rassp_43/sld022.htm

REVIEW: D-Algorithm

- Activate fault
  - **Place a D or $\overline{D}$** at fault site
  - Do **justification, forward implication,** and **consistency check** for all signals
- D-Drive
  - **Propagate D/$\overline{D}$** toward outputs
  - Do **justification, forward implication,** and **consistency check** for all signals
- Backtrack if
  - **A conflict** occurs, or
  - **D-frontier** becomes **empty**
- Stop when
  - **D** or **$\overline{D}$** at an output, i.e., **test found**, or
  - If search exhausted without a test, then **no test possible**

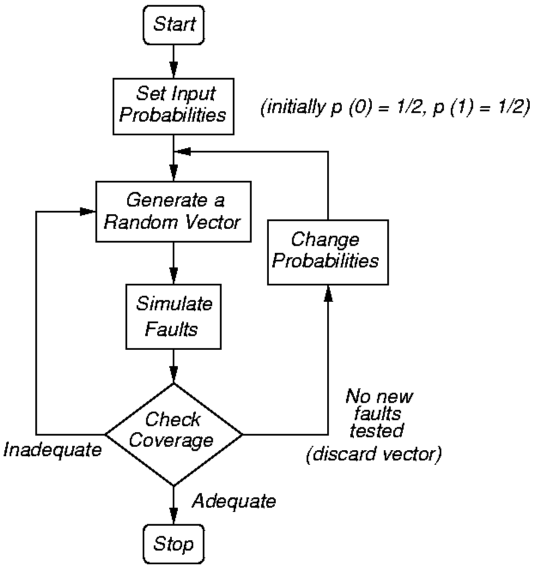
REVIEW: An ATPG System



Copyright Agrawal & Bushnell

Random Pattern Generation

- **Typically gets tests for 60-80% of faults**
- **Then switch to D-algorithm or other ATPG method**

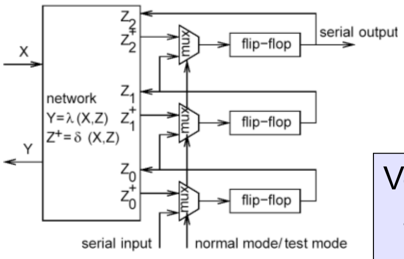


Copyright Agrawal & Bushnell

REVIEW: Fault injection

- Intentional activation of faults by HW or/and SW means
  - Establish faults in a predictable and reproducible way
  - Trigger error-handling routines
- **Two purposes:**
  - **Testing and Debugging**
    - During normal operation faults are *rare events*
    - May be much too rare to achieve meaningful data from std. testing
  - **Dependability Forecasting**
    - Used for deriving data about the likely dependability of the system
    - Need to know the types and frequencies of different faults in the intended operational environment

REVIEW: Scan design



Verifying a transition requires

- Shifting-in the state to be tested
- Application of the input pattern
- Checking if output is correct
- Shifting-out the successor state and comparing it.

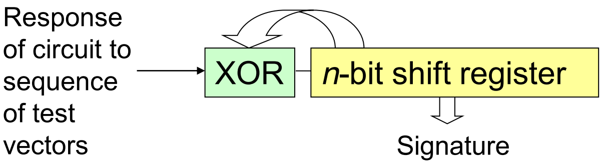
Essentially reduced to testing combinatorial logic

REVIEW: Limitations of a single serial scan chain

- For chips with a large number of flip-flops, serial shifts can take a quite long time.
- Hence, it becomes necessary to provide several scan chains.
  - ☞ Trying to avoid serial shifts by generating test patterns internally and by also storing the results internally.
  - ☞ Compaction of circuit response in a **signature**. Shifting the entire result out becomes obsolete, we just shift out the signature.

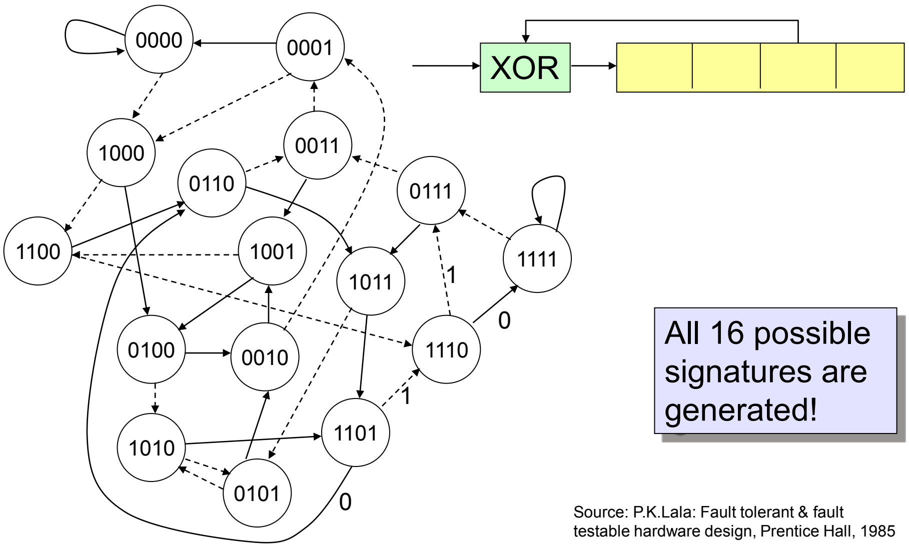
REVIEW: Signature analysis

- Response of circuit to sequence of test patterns compacted in a signature. Only this signature is compared to the golden reference.
- In order to exploit an *n*-bit signature register as well as possible, we try to use all possible values.
- In practice, we use shift-registers with linear feedback:



- Using proper feedback bits, all possible values for the register can be generated.

Example: 4-bit signature generator



All 16 possible signatures are generated!

Source: P.K.Lala: Fault tolerant & fault testable hardware design, Prentice Hall, 1985

Aliasing for signatures

Consider aliasing for some current pattern

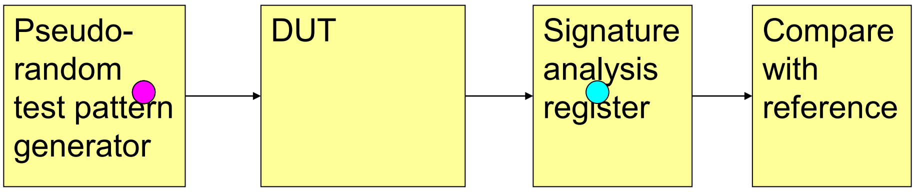
- An n-bit signature generator can generate 2^n signatures.
- For an m-bit input sequence, the best that we can get is to evenly map $2^{(m-n)}$ patterns to the same signature.
- Hence, there are $2^{(m-n)}$-1 sequences that map to the same signature as the pattern currently considered.
- In total, there are 2^m-1 sequences different from the current one.

☞ $$P = \text{Probability}\left(\frac{\text{other patterns map to same signature}}{\text{total number of other patterns}}\right) = \frac{2^{(m-n)} - 1}{2^m - 1}$$

$$P = \frac{1}{2^n} \text{ for } m >> n \text{ provided that we evenly map patterns to signatures}$$

Replacing serially shifted test pattern by pseudo-random test patterns

Shifting in test patterns can be avoided if we generate (more or less) all possible test patterns internally with a pseudo-random test pattern generator.


Pseudo-random test pattern generator

DUT

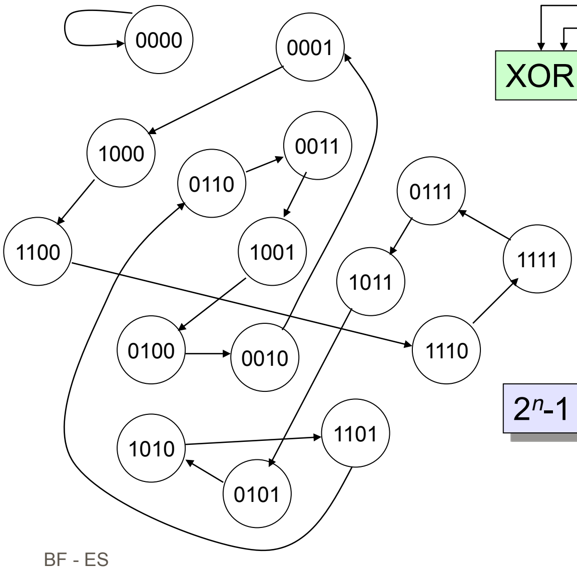
Signature analysis register

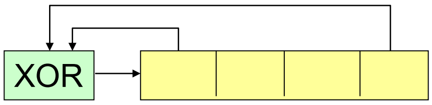
Compare with reference

- Effect of pseudo random numbers on coverage to be analyzed.
- Signature analysis register shifted-out at the end of the test.

Comparison possible because test pattern generator is a deterministic source!

Pseudo random test pattern generation





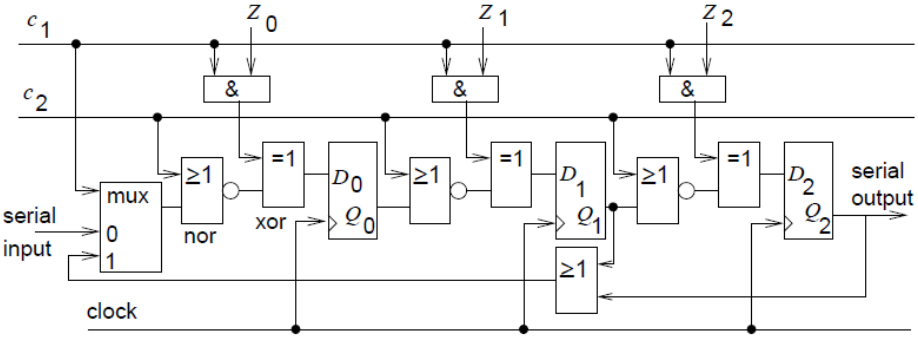
Linear feedback shift register (LFSR)

2^n-1 patterns generated!

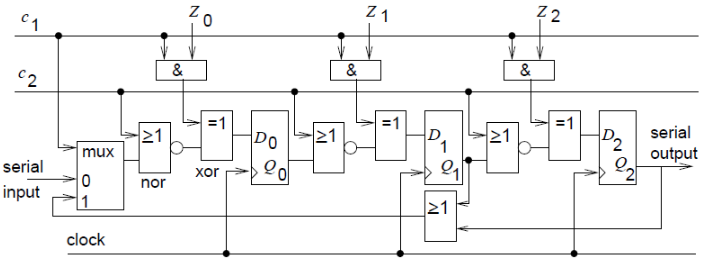
Combining signature analysis with pseudo-random test patterns: Built-in logic block observer (BILBO)

Könemann & Mucha

Uses *parallel* inputs to compress circuit response

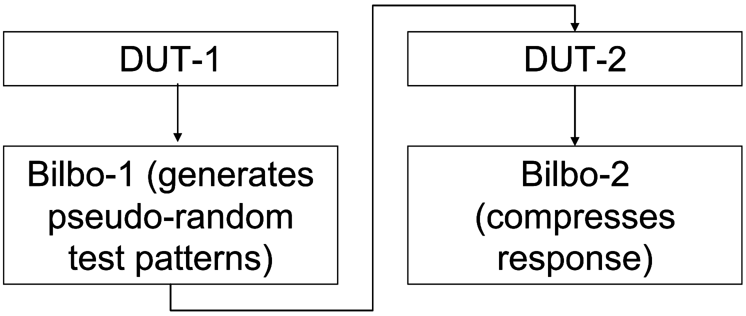


Built-in logic block observer (BILBO)
Modes of operation



c_1	c_2	D_i	
’0’	’0’	$'0' \oplus \overline{Q_{i-1}} = \overline{Q_{i-1}}$	scan path mode
’0’	’1’	$'0' \oplus \overline{'1'} =' 0'$	reset
’1’	’0’	$Z_i \oplus \overline{Q_{i-1}}$	LFSR mode
’1’	’1’	$Z_i \oplus \overline{'1'} = Z_i$	normal mode

Typical application



DUT-1

Bilbo-1 (generates pseudo-random test patterns)

DUT-2

Bilbo-2 (compresses response)

Compressed response shifted out of Bilbo-2 & compared to known „golden“ reference response.

Roles of Bilbo-1 and 2 swapped for testing DUT-1

Summary

- **Testing**
  - Fault model
  - ATPG: D-Algorithm
  - Fault coverage
  - Fault simulation for computing coverage
  - Fault injection
  - Model-based testing
- **Design for testability**
  - Scan path, Boundary scan
  - Signature analysis
  - Pseudo random patterns, BILBO

Formal Methods

The term refers to a broad set of notions and tools for

1. mathematically rigorous documentation of requirements
2. mathematically rigorous models of designs
3. verification of consistency
   - correctness of a design relative to requirements
   - replacability of one design by another

Automated Formal Methods

- Model Checking: automatically verify whether certain properties are guaranteed by the model; determine safe parameters
- Controller Synthesis: automatically construct control strategies that keep the system safe

Overview:

1. Intro: Analyzing FlexRay
2. Timed automata
3. Regions & zones
4. Model checking and controller synthesis
5. Hybrid automata

FlexRay Bus Protocol

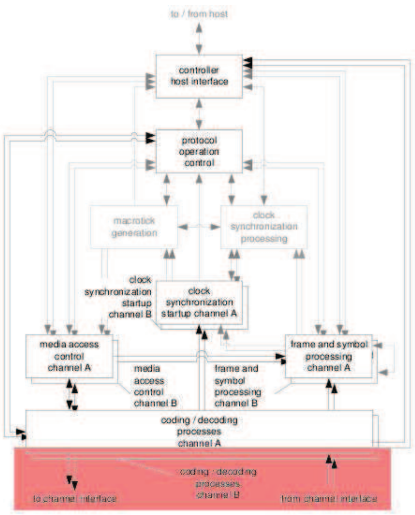
FlexRay

- communication protocol for distributed components in cars
- used in BMW X 5 and BMW's 7 series for X-by-wire
- developed by: BMW, Bosch, Daimler, Freescale, General Motors, NXP Semiconductors, Volkswagen, et al.

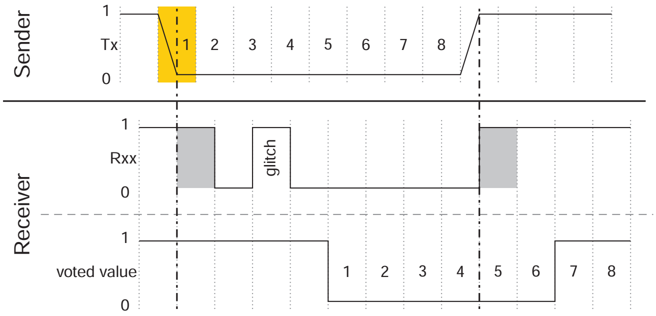
FlexRay as the Future Drive-by-Wire Standard

⇒ **Safety**-critical!

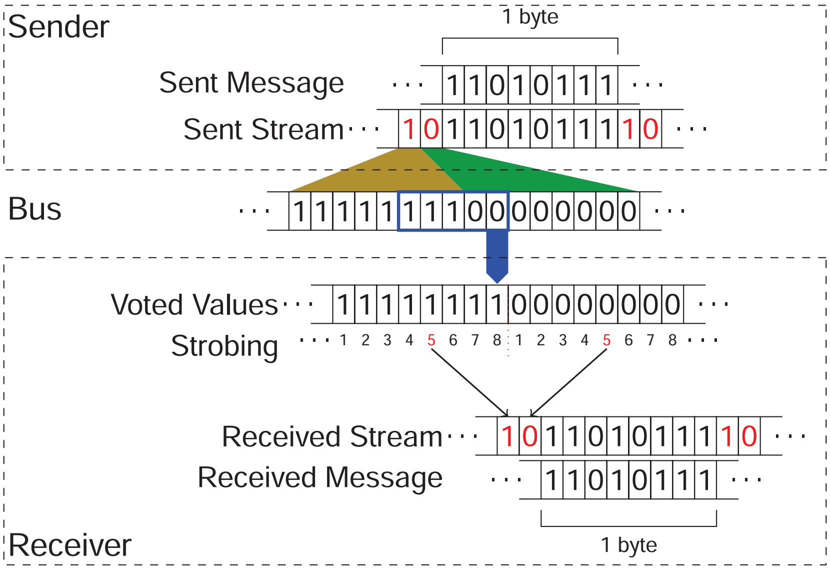
FlexRay Physical Layer


to / from host
controller host interface
protocol operation control
macrotick generation
clock synchronization processing
clock synchronization startup channel B
clock synchronization startup channel A
media access control channel A
media access control channel B
frame and symbol processing channel A
frame and symbol processing channel B
coding / decoding processes channel A
coding / decoding processes channel B
to channel interface
from channel interface

Jitter and Glitch Correction



Protocol Operation



Sender

1 byte

Sent Message ··· 1 1 0 1 0 1 1 1 ···

Sent Stream ··· 1 0 1 1 0 1 0 1 1 1 1 0 ···

Bus ··· 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 ···

Voted Values ··· 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 ···

Strobing ··· 1 2 3 4 5 6 7 8 1 2 3 4 5 6 7 8 ···

Received Stream ··· 1 0 1 1 0 1 0 1 1 1 1 0 ···

Received Message ··· 1 1 0 1 0 1 1 1 ···

1 byte

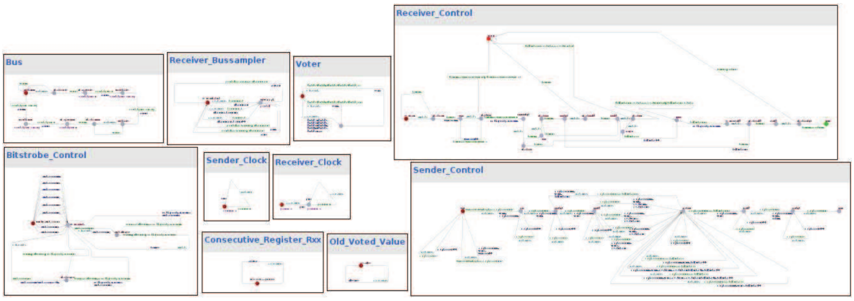
Receiver

Guaranteed Error Resilience?

Newest FlexRay Specification, Version 2.1, Revision A:

> *"[FlexRay] attempts to enable tolerance of the physical layer against presence of one glitch in a bit cell [. . .]. There are specific cases where a single glitch cannot be tolerated and others where two glitches can be tolerated."*

Michael Gerke's Model of the Protocol



- protocol
- jitter (parameterized)
- glitches

Automated Analysis: Glitch Tolerance

The protocol tolerates

- 1 glitch in every sequence of 4 consecutive samples (1 out of 4)
- 2 arbitrarily placed glitches in the complete message (at most 2)

Note: one message $\approx$ 21.000 samples

The protocol does not tolerate:
2 arbitr. placed glitches in every seq. of 82 consec. samples (2 out of 82)

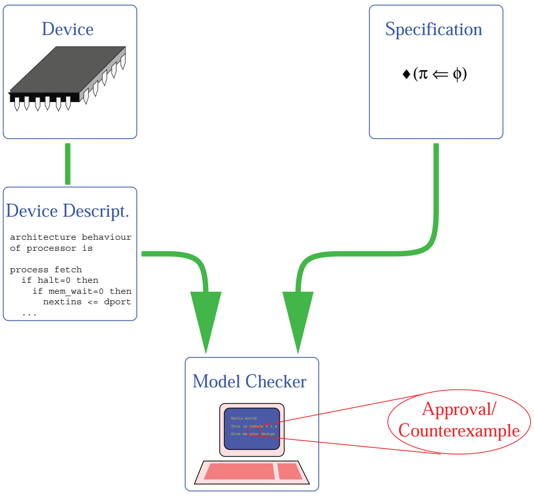
Automated Analysis: Glitch Tolerance vs. Delay Variance

Parameter exploration using binary search:
boundaries for variation of a single parameter

glitch tolerance	delay variance
(1 out of 4)	$1.435ns \rightarrow 7.6075ns$
(2 at most)	$1.435ns \rightarrow 7.6075ns$
(1 at most)	$1.435ns \rightarrow 12.020ns$

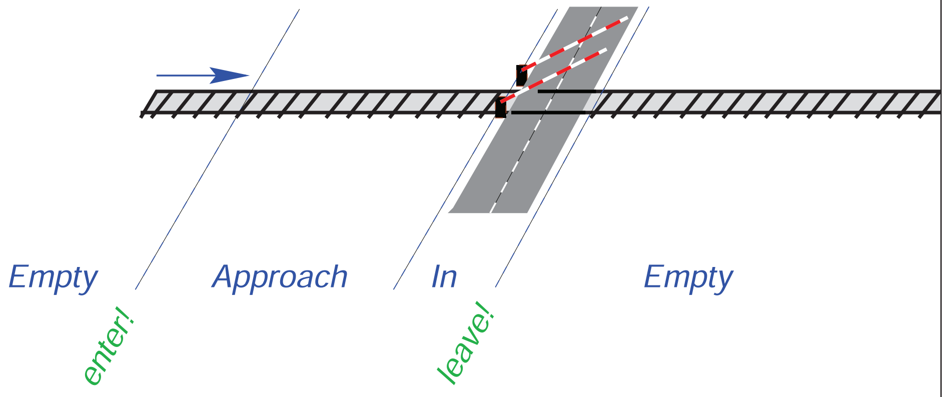
glitch tolerance	deviation of clock from standard rate
(1 out of 4)	0.15% → 0.46%
(2 at most)	0.15% → 0.46%
(1 at most)	0.15% → 1.09%
(no glitches)	0.15% → 1.74%

Model Checking



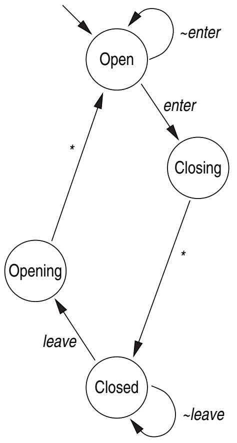
```
architecture behaviour
of processor is

process fetch
  if halt=0 then
    if mem_wait=0 then
      nextins <= dport
  ...
```

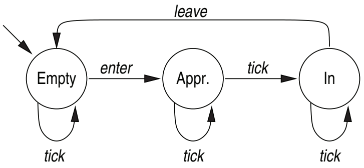
Finite-State Model-Checking



Safety requirement: Gate has to be closed whenever a train is in “In”.

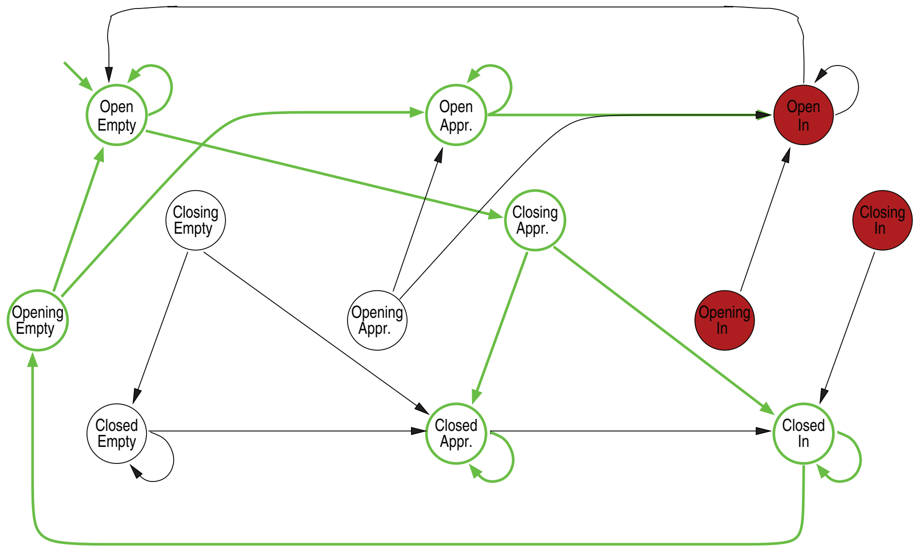
Finite-State Automata



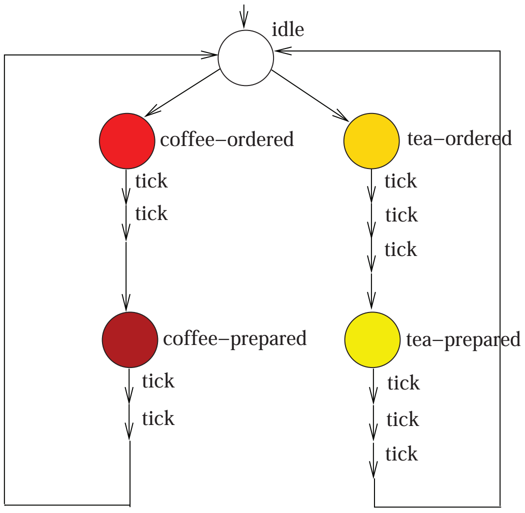


* = leave,enter,tick
~leave = enter, tick
~enter = leave, tick

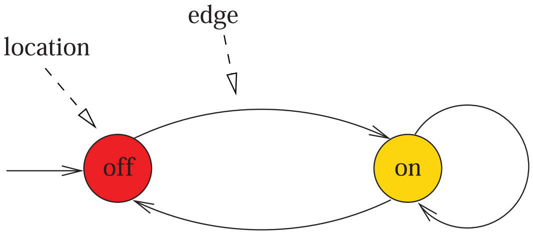
Model Checking



A Discrete-Time Coffee Machine

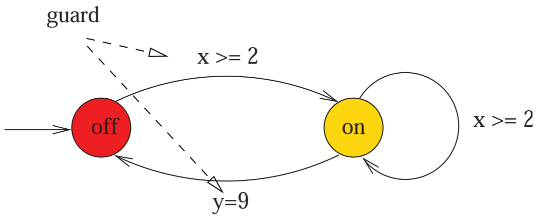


REVIEW: Timed Automata



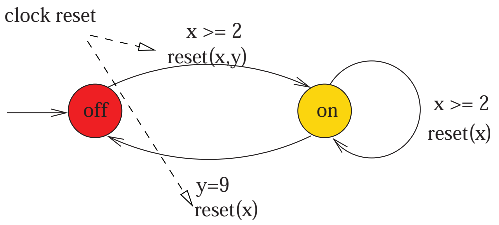
- a graph with *locations* and *edges*
- a location is labeled with the valid *atomic propositions*
- *taking an edge is instantaneous*, i.e, consumes no time

REVIEW: Timed Automata



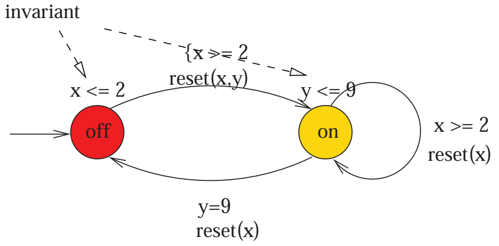
- equipped with real-valued *clocks* $x, y, z, \ldots$
- clocks advance implicitly, all at the *same speed*
- logical constraints on clocks can be used as *guards* of actions

Timed Automata with nondeterminstic delays [Alur/Dill]



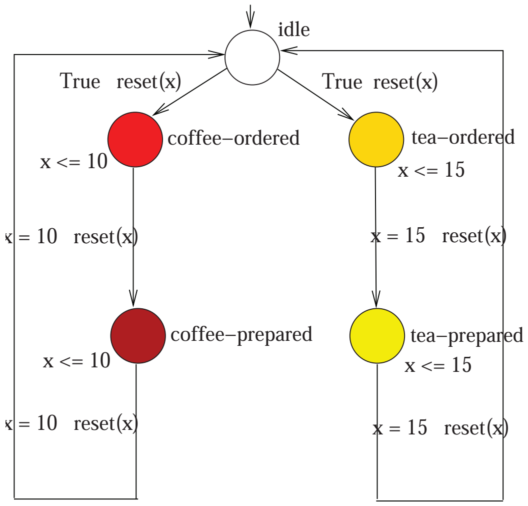
- clocks can be *reset* when taking an edge
- assumption:
  all clocks are zero when entering the initial location initially

Timed Automata



- guards indicate when an edge *may* be taken
- a location invariant specifies the *amount of time that may be spent in a location*
  - before a *location invariant* becomes false, an edge must be taken

A Real-Time Coffee Machine



Timed Automata with Nondeterministic Delays [Alur/Dill]

A *timed automaton* is a tuple

$$TA \;=\; \big(Loc, Act, C, \rightsquigarrow, Loc_0, inv, AP, L\big) \quad \text{where:}$$

- Loc is a finite set of locations.
- $Loc_0 \subseteq Loc$ is a set of initial locations
- C is a finite set of clocks
- $L : Loc \rightarrow 2^{AP}$ is a labeling function for the locations
- $\rightsquigarrow \subseteq \; Loc \times CC(C) \times Act \times 2^C \times Loc$ is a transition relation, and
- $inv : Loc \rightarrow CC(C)$ is an invariant-assignment function

Clock Constraints

Clock constraints over set C of clocks are defined by:

$$g ::= \; \mathit{True} \;\big|\; x < c \;\big|\; x \leq c \;\big|\; \neg g \;\big|\; g \wedge g$$

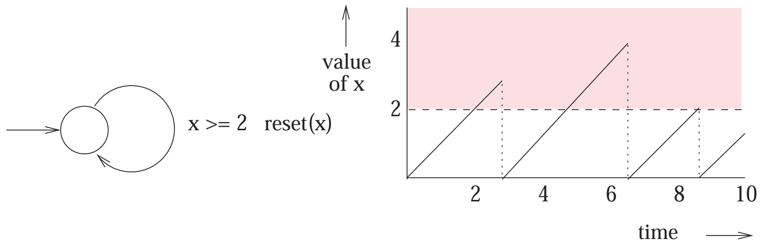
- where $c \in \mathbb{N}$ and clocks $x, y \in C$
- rational constants would do; neither reals nor addition of clocks!
- let $CC(C)$ denote the set of clock constraints over C
- shorthands: $x \geq c$ denotes $\neg(x < c)$
  and $x \in [c_1, c_2)$ or $c_1 \leq x < c_2$ denotes $\neg(x < c_1) \wedge (x < c_2)$

Intuitive Interpretation

- Edge $\ell \xrightarrow{g:\alpha,C'} \ell'$ means:
  - action α is enabled once guard g holds
  - when moving from location ℓ to ℓ', any clock in C' will be reset to zero
- $inv(\ell)$ constrains the amount of time that may be spent in location ℓ
  - the location ℓ must be left before the invariant $inv(\ell)$ becomes false

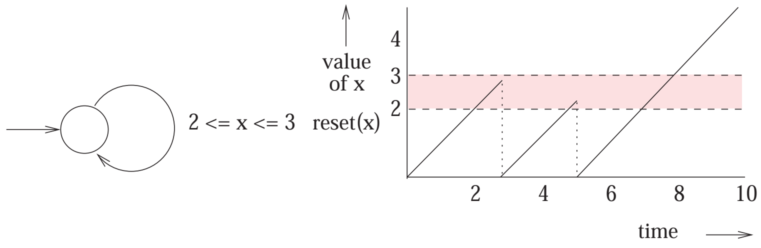
Guards vs. Location Invariants

The effect of a lowerbound guard:



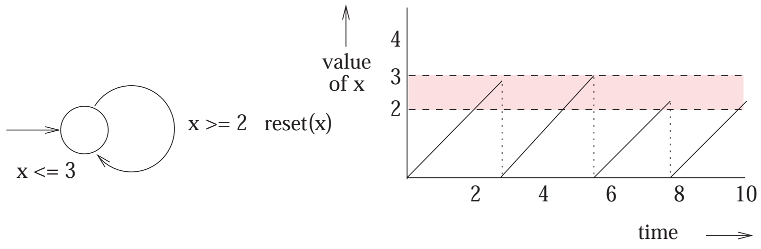
Guards vs. Location Invariants

The effect of a lowerbound and upperbound guard:

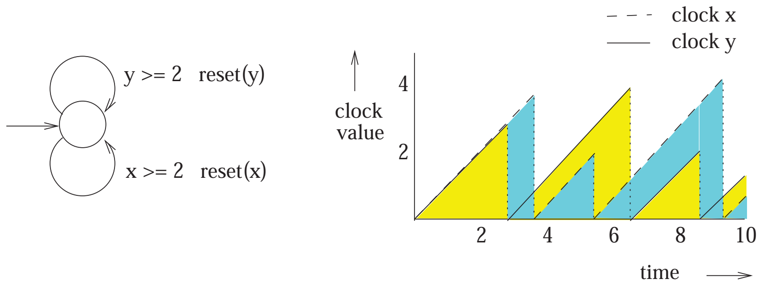


Guards vs. Location Invariants

The effect of a guard and an invariant:



Arbitrary Clock Differences



Composing Timed Automata

Let $TA_i \;=\; \big(Loc_i, Act_i, C_i, \rightsquigarrow_i, Loc_{0,i}, inv_i, AP, L_i\big)$ and H an action-set

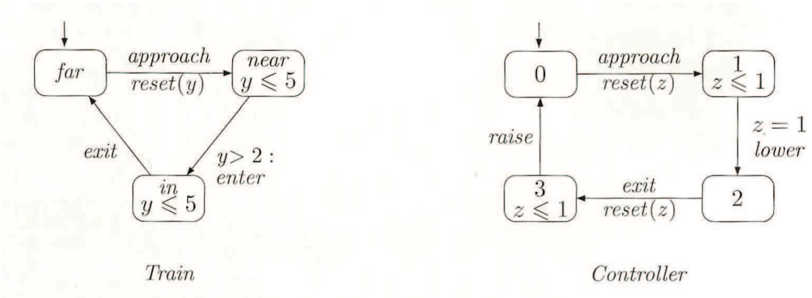
$TA_1 \;||_H\; TA_2 = \big(Loc, Act_1 \cup Act_2, C, \rightsquigarrow, Loc_0, inv, AP, L\big)$ where:

- $Loc = Loc_1 \times Loc_2$ and $Loc_0 = Loc_{0,1} \times Loc_{0,2}$ and $C = C_1 \cup C_2$
- $inv(\langle \ell_1, \ell_2 \rangle) = inv_1(\ell_1) \;\wedge\; inv_2(\ell_2)$ and $L(\langle \ell_1, \ell_2 \rangle) = L_1(\ell_1) \cup L_2(\ell_2)$
- $\rightsquigarrow$ is defined by the inference rules:

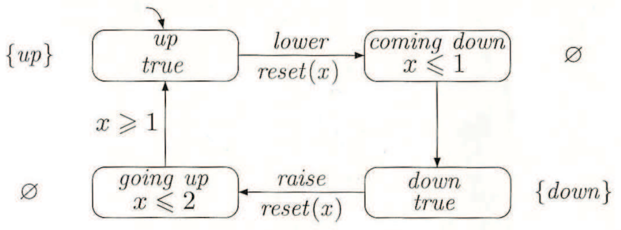
$$\text{for } \alpha \in H \qquad \frac{\ell_1 \overset{g_1:\alpha,D_1}{\rightsquigarrow_1} \ell_1' \;\wedge\; \ell_2 \overset{g_2:\alpha,D_2}{\rightsquigarrow_2} \ell_2'}{\langle \ell_1, \ell_2 \rangle \overset{g_1 \wedge g_2:\alpha,D_1 \cup D_2}{\rightsquigarrow} \langle \ell_1', \ell_2' \rangle}$$

$$\text{for } \alpha \notin H\text{:} \frac{\ell_1 \overset{g:\alpha,D}{\rightsquigarrow_1} \ell_1'}{\langle \ell_1, \ell_2 \rangle \overset{g:\alpha,D}{\rightsquigarrow} \langle \ell_1', \ell_2 \rangle} \quad \text{and} \quad \frac{\ell_2 \overset{g:\alpha,D}{\rightsquigarrow_2} \ell_2'}{\langle \ell_1, \ell_2 \rangle \overset{g:\alpha,D}{\rightsquigarrow} \langle \ell_1, \ell_2' \rangle}$$

Example: Railroad Crossing



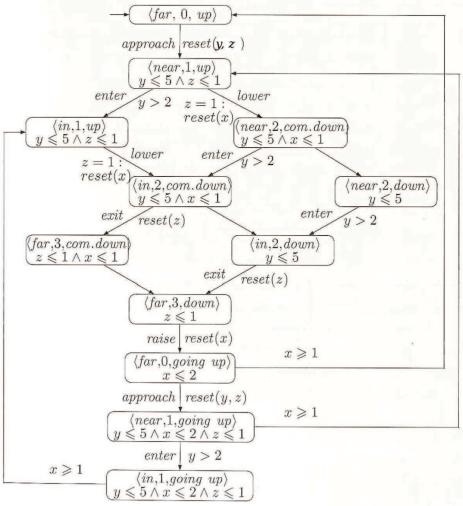
Example: Railroad Crossing



Gate

Example: Railroad Crossing

$(Train_{\{approach,exit\}}||Controller)||_{\{lower,raise\}}Gate$


$\langle far, 0, up\rangle$
$approach$
$reset(y, z)$
$\langle near,1,up\rangle$
$y \leqslant 5 \wedge z \leqslant 1$
$enter$
$y > 2$
$z = 1:$
$reset(x)$
$lower$
$\langle in,1,up\rangle$
$y \leqslant 5 \wedge z \leqslant 1$
$\langle near,2,com.down\rangle$
$y \leqslant 5 \wedge x \leqslant 1$
$z = 1:$
$reset(x)$
$lower$
$enter$
$y > 2$
$\langle in,2,com.down\rangle$
$y \leqslant 5 \wedge x \leqslant 1$
$\langle near,2,down\rangle$
$y \leqslant 5$
$exit$
$reset(z)$
$enter$
$y > 2$
$\langle far,3,com.down\rangle$
$z \leqslant 1 \wedge x \leqslant 1$
$\langle in,2,down\rangle$
$y \leqslant 5$
$exit$
$reset(z)$
$\langle far,3,down\rangle$
$z \leqslant 1$
$raise$
$reset(x)$
$\langle far,0,going\ up\rangle$
$x \leqslant 2$
$x \geqslant 1$
$approach$
$reset(y, z)$
$\langle near,1,going\ up\rangle$
$y \leqslant 5 \wedge x \leqslant 2 \wedge z \leqslant 1$
$x \geqslant 1$
$enter$
$y > 2$
$x \geqslant 1$
$\langle in,1,going\ up\rangle$
$y \leqslant 5 \wedge x \leqslant 2 \wedge z \leqslant 1$

Clock valuations

- A *clock valuation* v for set C of clocks is a function $v : C \to \mathbb{R}_{\geq 0}$
  - assigning to each clock $x \in C$ its current value $v(x)$
- Clock valuation $v{+}d$ for $d \in \mathbb{R}_{\geq 0}$ is defined by:
  - $(v{+}d)(x) = v(x) + d$ for all clocks $x \in C$
- Clock valuation reset x in v for clock x is defined by:

$$(\text{reset } x \text{ in } v)(y) = \begin{cases} v(y) & \text{if } y \neq x \\ 0 & \text{if } y = x. \end{cases}$$

  - reset x in (reset y in v) is abbreviated by reset x, y in v

Timed automaton semantics

For timed automaton $TA = \big(Loc, Act, C, \rightsquigarrow, Loc_0, inv, AP, L\big)$:
state graph $S(TA) = (Q, Q_0, E, L')$ over AP where:

- $Q = Loc \times val(C)$, state $s = \langle \ell, v \rangle$ for location ℓ and clock valuation v
- $Q_0 = \{ \langle \ell_0, v_0 \rangle \mid \ell_0 \in Loc_0 \ \wedge \ v_0(x) = 0 \text{ for all } x \in C \}$
- $L'(\langle \ell, v \rangle) = L(\ell)$
- E is the edge set defined on the next slide

Timed automaton semantics

The edge set E consist of the following two types of transitions:

- Discrete transition: $\langle \ell, v\rangle \xrightarrow{\alpha} \langle \ell', v'\rangle$ if all following conditions hold:
  - there is an edge labeled $(g : \alpha, D)$ from location ℓ to ℓ' such that:
  - g is satisfied by v, i.e., $v \models g$
  - $v' = v$ with all clocks in D reset to 0, i.e., $v' =$ reset D in v
  - v' fulfills the invariant of location ℓ', i.e., $v' \models inv(\ell')$
- Delay transition: $\langle \ell, v\rangle \xrightarrow{d} \langle \ell, v{+}d\rangle$ for positive real d
  - if for any $0 \leq d' \leq d$ the invariant of ℓ holds for $v{+}d'$, i.e. $v{+}d' \models inv(\ell)$

Time divergence

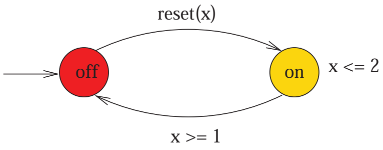
- Let for any $t < d$, for fixed $d \in \mathbb{R}_{>0}$, clock valuation $\eta{+}t \models inv(\ell)$
- A possible execution fragment starting from the location ℓ is:

$$\langle \ell, \eta\rangle \xrightarrow{d_1} \langle \ell, \eta{+}d_1\rangle \xrightarrow{d_2} \langle \ell, \eta{+}d_1{+}d_2\rangle \xrightarrow{d_3} \langle \ell, \eta{+}d_1{+}d_2{+}d_3\rangle \xrightarrow{d_4} \ldots$$

  - where $d_i > 0$ and the infinite sequence $d_1 + d_2 + \ldots$ *converges* towards d
  - such path fragments are called *time-convergent*

  ⇒ time advances only up to a certain value
- Time-convergent execution fragments are unrealistic and *ignored*

Example: light switch



The path

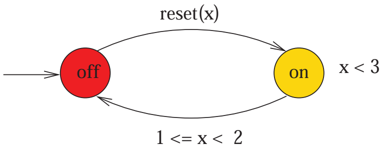
$$\pi = \langle \mathit{off}, 0\rangle \, \langle \mathit{off}, 1\rangle \, \langle \mathit{on}, 0\rangle \, \langle \mathit{on}, 1\rangle \, \langle \mathit{off}, 1\rangle \, \langle \mathit{off}, 2\rangle \, \langle \mathit{on}, 0\rangle \, \langle \mathit{on}, 1\rangle \, \langle \mathit{off}, 1\rangle \ldots$$

is *time-divergent*.
The path

$$\pi' = \langle \mathit{off}, 0\rangle \, \langle \mathit{off}, 1/2\rangle \, \langle \mathit{off}, 3/4\rangle \, \langle \mathit{off}, 7/8\rangle \, \langle \mathit{off}, 15/16\rangle \ldots$$

is *time-convergent*.

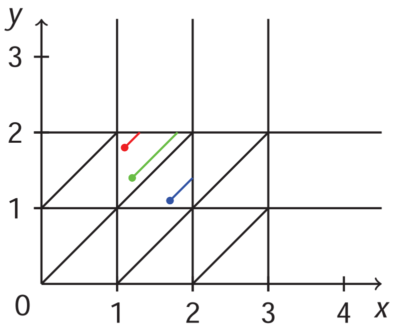
Timelock



- State $s \in S(TA)$ contains a *timelock* if there is a reachable state s where there is no time-divergent path from s
- Timelocks are considered as *modeling flaws* that should be avoided

Region Abstraction

- Consider a timed automaton with clocks x and y
- having maximal constants 3 and 2, respectively.

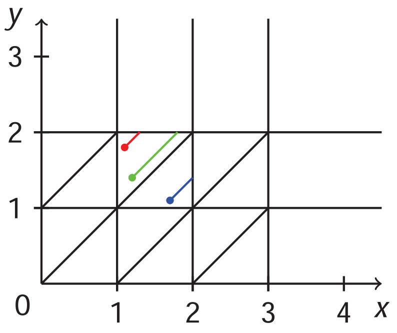


Equivalence relation $\simeq_R$

1. constraints
2. time elapsing
3. maximal constants

Region Abstraction

- Consider a timed automaton with clocks x and y
- having maximal constants 3 and 2, respectively.

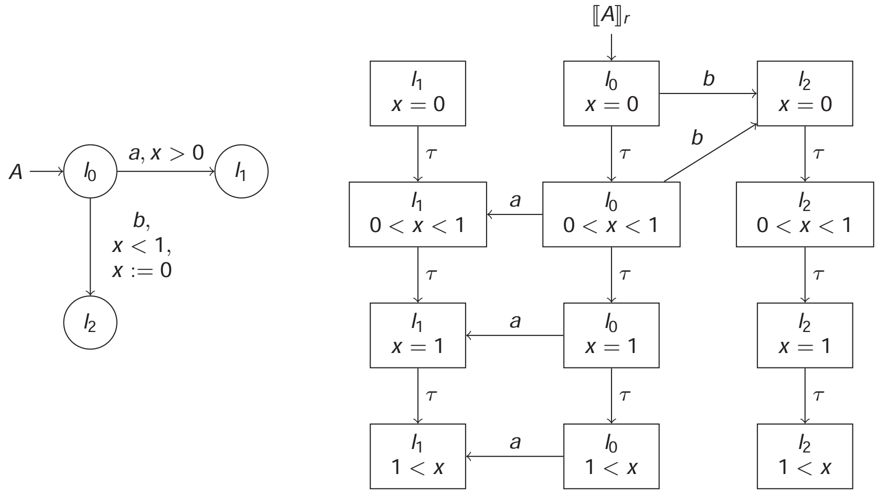


Equivalence relation $\simeq_R$

1. constraints
2. time elapsing
3. maximal constants

$\Longrightarrow$ finite index!

Finite Semantics: Region Automaton



Timed Analysis

Reachability is decidable

Theorem [Alur, 1994]:

$$\exists \text{ path } (l, \vec{t}) \longrightarrow (l', \vec{t}')$$

iff

$$\exists \text{ path } (l, [\vec{t}]_R) \longrightarrow (l', [\vec{t}']_R)$$

Symbolic data structures

- Clock Region = Finest integral unit
- Clock Zone = Convex union of clock regions
- Federation = (Non-convex) union of clock zones