1.3 Further Issues

System Modes vs. Environmental Modes

- environmental mode
  - equivalence class of histories
  - change depends on occurrence of events
  - initial env. mode depends on history before system turned on
- system mode
  - equivalence class of system states
  - change depends on *detection* of events
  - initial system mode is fixed
- *ideally*, system and env. modes should be equivalent

"Ideal" Behaviour is Impossible

- *accuracy* of measurement of analogue monitored quantities
- *tolerance* of analogue controlled quantities
- important analogue monitored quantity: *time*
  - detection of events needs time
  - reaction to events needs time

A Useful Heuristics for “Real” Behaviour

- specify “ideal” behaviour relation
- specify separately accuracy and tolerance relations and concatenate these relations
  - do not forget this!
- may not work for more complex timing
  - then need explicit “transition” modes

Example: Logic Probe

- device giving a short pulse of 100 ms when button pressed

^{Cl}probe =

Mode	Event Class	New Mode
^{Md}test	@T(^{m}Pulse = ^{C}Down)	^{Md}pulse
^{Md}pulse	@T(Since(@T(^{Md}pulse)) > 100 ms)	^{Md}test

Maximum Delay: 2 ms

Logic Probe With Delay, Expanded

- same behaviour, but without delay specification
- implicit transition modes made explicit for demonstration

^{Cl}probe =

Mode	Event Class	New Mode
$^{Md}\widehat{\text{test}}$	@T(^{m}Pulse = ^{C}Down)	^{Md}test–pulse
^{Md}test–pulse	@T(^{c}Requiv ≤ 320 Ω)	$^{Md}\widehat{\text{pulse}}$
	@T(Since(@T(^{Md}test–pulse)) ≥ 2 ms)	
$^{Md}\widehat{\text{pulse}}$	@T(Since(@T($^{Md}\widehat{\text{pulse}}$)) > 100 ms)	^{Md}pulse–test
^{Md}pulse–test	@T(^{c}Requiv ≥ 500 kΩ)	$^{Md}\widehat{\text{test}}$
	@T(Since(@T(^{Md}pulse–test)) ≥ 2 ms)	

- ^{c}Requiv: a controlled variable reflecting the mode (needed!)

Using Discrete Clocks

- many embedded software systems:
  cycle read→process→write→. . .
- read and write at discrete points of time
  - system requirements should permit such implementations
- concise requirements by specifying the required *resolution* of time
  - resolution = smallest significant increment of time

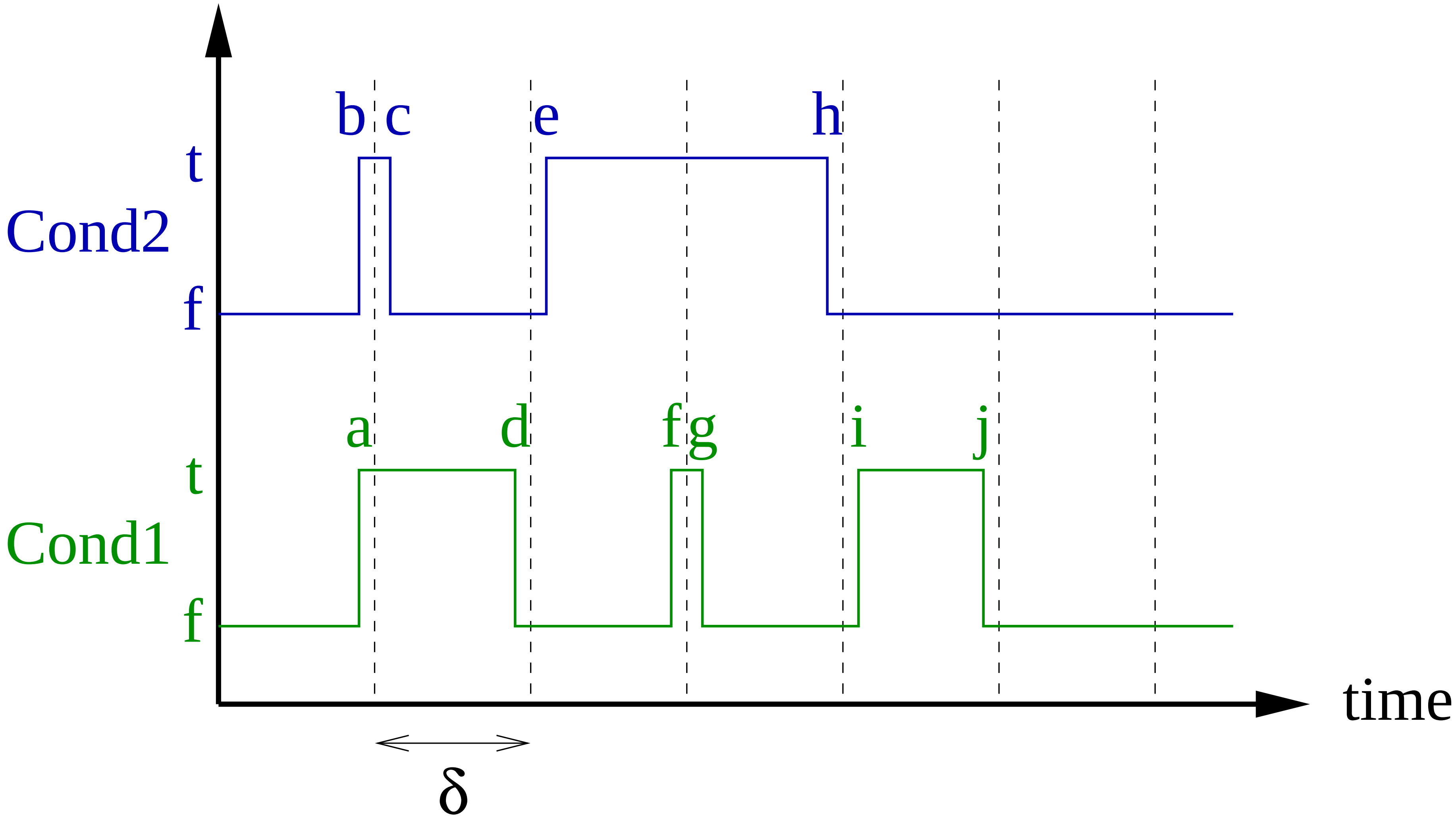
Implications for System when Specifying a Resolution of Time δ

- system clock frequency $\geq \frac{1}{\delta}$
  - sufficient to sample monitored quantities at rate of $\frac{1}{\delta}$

Implications for Requirements when Specifying a Resolution of Time δ

- changes in environment that occur within δ may be considered simultanteous
- system can only be required to detect conditions that have held for at least δ
- max. measurement accuracy for instants: $+0$ / $-\delta$
- max. measurement accuracy for time intervals: $\pm\delta$
- min. delay tolerance for response to any event: δ

Example: Time Resolution



Useful Standard Functions For Time

- implicitly interpreted w.r.t. a particular behaviour on the interval of the system's operation $[t_i, t_f]$

Prev(e, t)	the set of events of event class e that occur prior to t
Last(e, t)	the time of the latest event of event class e before t
First(e, t)	the time of the earliest event of event class e before t

Drtn(p_i, t)	the duration that condition p_i has been continuously true up to time t
totalDrtn(p_i, t_1, t_2)	the total amount of time that condition p_i has been true between times t_1 and t_2
Since(e, t)	the time since the latest event of event class e before t

- if time argument t is current time t_f, it will be omitted by convention
- precise definitions in [Pet00, pp. 49]

Repetition: Events

An event e, is a pair, (t, c), where
$e.t \in \mathbb{R}$ is a time at which one or more conditions change value and
$e.c \in \{T, F, @T, @F\}^n$ indicates the status of all conditions at $e.t$, as follows:

$e.c[i]$	p_i
T	$`p_i(e.t) \wedge p_i'(e.t)$
F	$\neg `p_i(e.t) \wedge \neg p_i'(e.t)$
@T	$\neg `p_i(e.t) \wedge p_i'(e.t)$
@F	$`p_i(e.t) \wedge \neg p_i'(e.t)$

Some Useful Event Class Notation

Notation	$e.c[i]$
$*$	true
$\oslash$	false
–	F $\vee$ T
t	T $\vee$ @F
f	F $\vee$ @T
t′	T $\vee$ @T
f′	F $\vee$ @F

- $\mathsf{t}(p_i) = {}^\backprime p_i(e.t) \wedge \mathsf{true}$
- $\mathsf{t}'(p_i) = \mathsf{true} \wedge p_i{}'(e.t)$

Example: Telephone Connection

- table describes the connection mode between any two users u and v
- from a large requirements specification (Bredereke)

current mode	**conditions**			**next mode**
	^{m}connectReq(u,v)	inmode(^{Md}connection-ResourceAvail(u,v))	^{m}connectRsp(v)	
^{Md}Idle(u,v)	@T	t′	–	^{Md}Setup(u,v)
	@T	f′	–	^{Md}OTeardown(u,v)
^{Md}Setup(u,v)	–	T	@T	^{Md}Established(u,v)
	@F	$*$	–	^{Md}Idle(u,v)
	–	@F	$*$	^{Md}OTeardown(u,v)
^{Md}Established(u,v)	–	$*$	@F	^{Md}OTeardown(u,v)
	@F	$*$	–	^{Md}TTeardown(u,v)
	–	@F	–	^{Md}BTeardown(u,v)
^{Md}OTeardown(u,v)	@F	$*$	–	^{Md}Idle(u,v)
^{Md}TTeardown(u,v)	–	$*$	@F	^{Md}Idle(u,v)
^{Md}BTeardown(u,v)	–	$*$	@F	^{Md}OTeardown(u,v)
	@F	$*$	–	^{Md}TTeardown(u,v)

Tabular vs. Scalar Notation for Event Classes

tabular	scalar
p_i	
T	WHILE(p_i)
F	WHILE$(\neg p_i)$
@T	@T(p_i)
@F	@F(p_i)
$*$	*(not useful)*
–	CONT(p_i)
$\oslash$	*(not useful)*

tabular	scalar
p_i	
t	WHEN(p_i)
f	WHEN$(\neg p_i)$
t'	*(no notation defined)*
f'	*(no notation defined)*

Example: Tabular Expressions

^{Cl}floor:

current mode	**conditions**			**next mode**
	^{p}at1stFloor	^{p}at2ndFloor	^{p}at3rdFloor	
^{Md}in1stFloor	–	@T	–	^{Md}in2ndFloor
^{Md}in2ndFloor	@T	–	–	^{Md}in1stFloor
	–	–	@T	^{Md}in3rdFloor
^{Md}in3rdFloor	–	@T	–	^{Md}in2ndFloor

Example: Scalar Expressions

^{Cl}floor:

Mode	Event Class	New mode
^{Md}in1stFloor	@T(^{p}at2ndFloor)	^{Md}in2ndFloor
^{Md}in2ndFloor	@T(^{p}at1stFloor)	^{Md}in1stFloor
	@T(^{p}at3rdFloor)	^{Md}in3rdFloor
^{Md}in3rdFloor	@T(^{p}at2ndFloor)	^{Md}in2ndFloor

Requirements Feasibility

$\longrightarrow$ blackboard. . .

Fail-Soft Behaviour in the Four-Variable Approach

- repetition: acceptability of a software SOF:
  $((\mathrm{IN} \cdot \mathrm{SOF} \cdot \mathrm{OUT}) \cap \mathrm{NAT}) \subseteq \mathrm{REQ}$
- if devices are broken, software is not constrained at all
- specify weaker versions of IN, OUT, and SOF that hold if some devices are broken
- software must satisfy the conjunction of of all requirements specified this way

Merit Functions

- although all behaviours in REQ are acceptable, some are preferable over others
- examples:

  processing speed: quicker responses preferred

  soft real-time constraints: failure to respond within specified time not catastrophic, but undesirable

  safety margins: controlled values may approach certain thresholds, but the larger the safety margin the better

  stability: large oscillations in controlled values are undesirable

Definition 3 (Merit function)

A merit function is a function of a behaviour that indicates which behaviours are preferred over which others – the higher the merit function value the more preferred the behaviour.

- related to "objective function" in control systems and optimization

Limitations of the Approach

necessary:

1. env. quantities can be expressed as functions of time that are either
   - piecewise-continuous (for real-valued quantities), or
   - finitely variable (for discrete-valued quantities)
2. the acceptable behaviour can be characterized by a relation on the env. quantities

Environmental Quantities Not Expressible

- if cannot be expressed effectively
  - example: compiler
  - source code = array of characters???
- if not usefully viewed as functions of time
  - example: compiler
  - only two instants of time relevant (start, termination)

- approach unsuitable for "information processing" systems in particular

Requirements Relation Not Expressible

- non-behavioural properties
  - maintainability
  - code size
- internal properties
  - number of times an instruction is invoked (if not externally observable)
- requirements not preserved under sub-setting of behaviours

Requirements Not Preserved Under Sub-Setting of Behaviours

- average response time over all behaviours
  - different from average over a single behaviour (which can be expressed)
  - usually, such statistical properties can be approximated reasonably well and specified with reference to a lengthy execution

- possibilistic properties
  - important for security
  - “if behaviour A is possible, then behaviour B must also be possible”
  - this is not the same as
    $A \in \mathrm{REQ} \Rightarrow B \in \mathrm{REQ}$
  - what is acceptable in an implementation
    is different from what is possible
  - usually, REQ is non-deterministic, but the implementation is not
  - intruders must not be able to infer information
    from the possibility of A and the impossibility of B