Master of Science in Computer Engineering

Software Systems Engineering

a.a. 2021-2022

Gigliola Vaglini

1

Formal Requirements Specifications

Lecture 5

2

2

Ways of writing a requirement specification

Notation	Description
Natural language	The requirements are written using numbered sentences in natural language. Each sentence should express one requirement.
Structured natural language	The requirements are written in natural language on a standard form or template. Each field provides information about an aspect of the requirement.
Design description languages	This approach uses a language like a programming language, but with more abstract features to specify the requirements by defining an operational model of the system. This approach is now rarely used although it can be useful for interface specifications.
Graphical notations	Graphical models, supplemented by text annotations, are used to define the functional requirements for the system; UML use case and sequence diagrams are commonly used.
Mathematical specifications	These notations are based on mathematical concepts such as finite-state machines or sets. Although these unambiguous specifications can reduce the ambiguity in a requirements document, most customers don't understand a formal specification. They cannot check that it represents what they want and are reluctant to accept it as a system contract

3

3

Static analyzers

- Static analysis tools
  - syntactic and semantic analyzers intended for checking software for conformance or for verifying desired properties.
- They are tools exploiting
  - mathematical approaches to solving software (and hardware) problems at the requirements and design levels
- In particular, they perform properties verification exploiting
  - Formal specification languages for the properties: logic languages, in general, based on different theories depending on the type of the system
  - Formal specification languages for the system
  - algorithms to verify the satisfaction of a property on a system specification.

4

4

Formal methods

- By the sentence Formal methods we indicate frameworks where properties (requirements) are verified on the system specification
- Various formal specification notations are available for systems, in particular Finite State Machine based methodologies that also allow executable software specification .
- The general approach called Formal methods are most likely to be applied where the software is safety critical. Software safety assurance standards demand formal methods.
- In essence
  - verification is carried on in an automatic way
  - verification is formal, i.e.,
    - Properties are formally defined
    - Proofs are rigourous

5

5

A particular system

- The concurrent systems case:
  - Dynamic verification is not satisfiable since the system behavior is not repeatable
  - Critical systems require a high level of dependability
  - Real time systems have requirements (properties) that are time dependent

6

6

The properties

- Properties of concurrent systems are difficult to be expressed in a graphical language.
- A formalism able to express time dependent properties is temporal logic that can use both a qualitative notion of time, specifying that either ''necessarily'' something will occur, or ''possibly'' something will occur, and a quantitative notion, specifying also that after t time units ''necessarily'' and ''possibly'' something will occur.
- The set of system properties defines, in some sense, a different, not operational, system model

7

7

Syntax of a logic language

- The syntax of a logic language is given through
  - Formulae: correct sequences of symbols belonging to a given alphabet
  - Inference rules: rules that derive formulae from formulae
  - Axioms: formulae known true
- Each formula represents a property (requirement) of the system.

8

8

Logic languages classes

- Propositional logic
  - Each formula expresses an absolute truth starting from known facts.
    - The result of the formula evaluation depends only on the values of the proposition symbols
- Predicate logic
  - Each formula expresses a relative truth with respect to particular sets of the world.
    - x exists such that A(x)
- Modal logic
  - Each formula expresses a relative truth with respect to a world and such truth can change from a world to another in a particular universe.

9

9

Extensional and intensional logics

- Classic logics are extensional: the truth value of each formula derives from that of the sub-formulae and the meaning of the operators.
- Modal logic is intensional: the truth value of a formula does not necessarily derive from that of the sub-formuae and the meaning of the operators.

10

10

Modal operators

- Fundamental modal operators are
  - "necessarily"
  - "possibly"
  - The operators are dual:
    - given the formula ϕ , possibly $\phi = \neg$ necessarily $\neg \phi$

11

11

Semantics of a logic language

- The formula semantics is a truth value determined through the interpretation.
  - In the predicate logics an interpretation is a pair

    $\mathcal{I}=(\mathcal{D}, \alpha)$

    where α associates each symbol (constant, variable, function) with an element (or n-ple of elements) of $\mathcal{D}$; each proposition or predicate is associated with a truth value.
  - In the modal logics an interpretation is a pair

    $\mathcal{I}=(\mathcal{W}, \mathcal{R})$,

    where $\mathcal{W}$ is called universe and is composed of a set of worlds ($\mathcal{W}_1, \ldots ,\mathcal{W}_n$) linked together through the relation $\mathcal{R}$. Each $\mathcal{W}_i= (\mathcal{D}, \alpha)$.

12

12

Examples

- $(a \wedge b) \vee (c \wedge d)$
- "$\forall x$, even(x)" , where
  - even(x)=tt if (x mod 2)=0, else ff
- "necessarily $a \vee b$"
  – This formula can be false independently from the truth value that the interpretation of one world in the universe associates with "$a \vee b$".

13

13

The relation $\mathcal{R}$

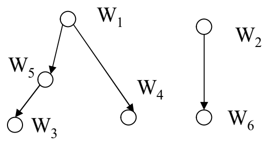
- *In fact*
- *The universe changes its characteristics depending on the relation type (a relation is symmetric, another one is transitive, and so on) used to connect worlds*
- The modal axioms define the type of R: a valid formula is true in all universes in which R is of a given type (thus not in all interpretations).

14

14

Example: $\mathcal{I}=(\mathcal{W}, \mathcal{R})$,

- W=({W_1,W_2,W_3,W_4,W_5,W_6}, R) R is a partial ordering relation



- Let us suppose
- W_2 = $(\mathcal{D}, \alpha_1)$, $\mathcal{D}$={a,b} α_1(b)=tt α_1(a)=ff
- W_6 = $(\mathcal{D}, \alpha_2)$, $\mathcal{D}$={a,b} α_2(b)=ff α_2(a)=ff

  ⇒ 'necessarily' $a \vee b$ is false in W_6 and then in W_2 too, and then in W

15

15

Example (2)

- Let us suppose
  - W_1 = $(\mathcal{D}, \alpha_1)$, $\mathcal{D}$={a,b} α_1(b)=tt α_1(a)=ff
  - W_4 = $(\mathcal{D}, \alpha_3)$, $\mathcal{D}$={a,b} α_2(b)=ff α_2(a)=tt
- In this case
  - "possibly $a \vee b$" is true in W_1 and in W_4, and then it is true in W.

16

16

Temporal logic

- Temporal logic is a particular modal logic where the worlds of an universe are temporal instants connected by a reflexive and transitive relation R: thus R establishes a partial ordering among worlds.
- The operators are [] (necessarily) and <> (possibly).
- A formula is true if it is true in all instants starting from the initial one (being R reflexive and transitive, all successive instants are reachable).
- Given a system (program), the set of computations of the system is an universe.

17

17

A TL logic: Hennessy-Milner Logic (HML)

$$\Phi ::= \mathbf{tt} \mid \mathbf{ff} \mid \Phi_1 \wedge \Phi_2 \mid \Phi_1 \vee \Phi_2 \mid [\mathbf{a}]\Phi \mid \langle \mathbf{a} \rangle \Phi$$

$$\mathbf{a} \in \mathbf{Act}$$

$[a]\Phi$ after each occurrence of action *a,* the resulting process must verify property Φ.

$\langle a \rangle\Phi$ at least an action *a* is required to occur and the resulting process must verify property Φ.

For example, in any state, <a>tt requires the ability of performing *a;* where [a]ff expresses the inability to perform such action.

18

18

Satisfaction (1)

- Given a formula ϕ , the processes satisfying ϕ can be singled out through the following rules defined on the structure of the formula. Given a process *p*, *p* satisfies ϕ (written p $\models \phi$) as follows
  - $\boldsymbol{p} \models \mathrm{tt}$, $\boldsymbol{p} \not\models \mathrm{ff}$
  - $\boldsymbol{p} \models \phi \wedge \xi$ iff $\boldsymbol{p} \models \phi$ and $\boldsymbol{p} \models \xi$
  - $\boldsymbol{p} \models \phi \vee \xi$ iff $\boldsymbol{p} \models \phi$ or $\boldsymbol{p} \models \xi$
  - $\boldsymbol{p} \models [a]\phi$ iff $\forall\, \boldsymbol{q} \in \{\boldsymbol{p'} : \boldsymbol{p} \xrightarrow{a} \boldsymbol{p'}\}.\ \boldsymbol{q} \models \phi$
  - $\boldsymbol{p} \models \langle a\rangle\phi$ iff $\exists\, \boldsymbol{q} \in \{\boldsymbol{p'} : \boldsymbol{p} \xrightarrow{a} \boldsymbol{p'}\}.\ \boldsymbol{q} \models \phi$

- Note that, from definion of $\models$
  - [a]tt is equivalent to tt
  - <a>ff is equivalent to ff

19

19

Some hints on the expressivity of HML

❖ [-]ff
  - Termination

❖ <->tt
  - Vitality

❖ <->tt $\wedge$ [-a]ff
  - Obligation

❑ The symbol "-" means any action, -a means any action except a
❑ The properties above are properties that have no time duration.
❑ Consider the property "action a can always be performed":
  ❑ Does the formula <a>tt express this property?

20

20

Some hints on the expressivity of HML (cont.)

- <a>tt is true on both a terminating and a non terminating system provided that the action a occurs in the system initial state
- Some form of recursion in the application of the formula is needed to simulate the flowing of the time.

21

21

Modal μ-calculus

$\Phi ::= tt \mid ff \mid \Phi_1 \wedge \Phi_2 \mid \Phi_1 \vee \Phi_2 \mid [K]\Phi \mid \langle K\rangle\Phi \mid$
$\nu Z.\ \Phi \mid \mu Z.\ \Phi \mid Z$

$K \subseteq Act$

νZ. Φ and μZ. Φ are fixed point formulae (greatest and least respectively), where the operators νZ e μZ bind the occurrences of the variable Z in Φ. Closed formulae do not contain free variables. The constants **tt** and **ff** can be obtained also as νZ.Z and μZ.Z, respectively.

22

22

Examples

- $\phi_1 = \nu Z.(\langle - \rangle tt \;\wedge\; [-]Z)$
  - Deadlock-freeness
- $\phi_2 = \mu Z.([-]ff \;\vee\; \langle - \rangle Z)$
  - Can-deadlock
- $\phi_3 = \mu Z.(\langle - \rangle tt \;\wedge\; [-a]Z)$
  - "after a finite amount of time *a* will be executed"
- $\phi_4 = \nu Z.([a]ff \wedge\; [-]Z)$
  - "henceforth *a* will not be executed "

23

23

Properties that are expressible by TL: Liveness and Safety

- *"something good will eventually happen"*
  - *Liveness: each path of the LTS contains the good thing (an action for example) . For example,* ϕ_3 *is a liveness property.*
- *"something bad will never occur"*
  - *Safety: no path of the transition system contains the bad thing . For example,* ϕ_4 *is a safety property.*

24

24

MODEL CHECKING

25

25

Model checking

- Model checking is an algorithmic method to verify temporal logic properties of the system
- In particular, the properties are checked on a structure (in general a state structure) representing the system
- The verification is made employing an algorithm and not a theorem prover

26

26

Model checking vs Theorem proving

- MC is semantics based: the proof is made on the structure representing the system by moving from a node to the successive and in each node a sub-property must be satisfied. For non recursive properties the complexity of known algorithms is linear

- TP is syntax based: the aim is to build all programs enjoying a given property.
- In general there is an infinite number of programs enjoying a property
- The strategy employed to verify if the given program belongs to the set of correct programs is given by the verifier and is not automatable

27

27

System model in this framework

- The formal description of the system is given abstract from implementative details but it must be:
  - comprehensible
  - unambiguous
  - expressive

28

28

Concurrent Systems specification

- Concurrent systems are characterized by the existence of
  - Parallel events
    - True concurrency or interleaving?
  - Communicating events
    - Synchronous communication?
    - Alternatives?
  - Nondeterministic occurrence of events

29

29

Concurrency description

- Interleaving
  - The concurrent execution of action a and action b produces an effect equivalent to a sequential execution of a and b, i.e. ab or ba
- True concurrency
  - The concurrent execution of action a and action b can produce an effect equivalent neither to ab nor to ba

30

30

Not directly states

- Languages
- Language constructs for expressing
  - Parallelism
  - Communication
  - Nondeterminism
- A formal system specification is written in a rigorous language and is based on a sound theory

31

31

An alternative: Algebraic languages

- Algebraic languages
  - Algebra = data + operators on data
- Process algebra
  - Data=processes
  - Operators= parallel composition, nondeterministic choice, communication…

32

32

Calculus of Communicating Systems (CCS) (Milner '89)

- Concurrency as interleaving
- Sinchronous communication

33

33

CCS syntax

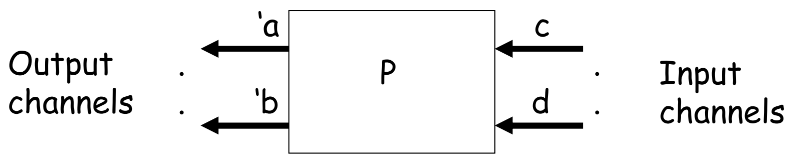
P ::= nil | a.P | P+P | P|P | P\L | P[f] | C

C := P	process definition
A = {a, b, .. }	input actions
'A = {'a, 'b .. }	output actions
Vis = A ∪ 'A	observable actions
τ	non observable action
Act = Vis U {τ}	action alphabet
L ⊆ Vis	f : Act ⟶ Act

34

34

Process interface

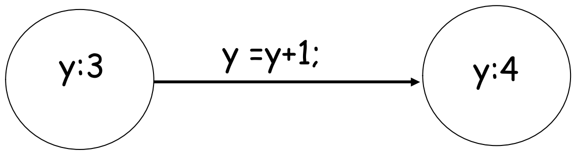


35

35

Operational semantics

- A new kind of semantics is given to processes based on automata theory :
  - (an action\event can cause a state transition)
  - thus, we have states and state transitions and
  - process semantics = state machine
- Each CCS process P is associated with a labeled transition system, LTS(P), that defines the behavior of P
- states = process
- initial state = P
- labels $\in$ Act



36

36

SOS semantics of CCS processes

- LTS(P) can be obtained by means of the structural operational semantics (SOS) of the CCS process P and
- LTS(P) is constructed by structural induction on the syntactic elements of P. This process is automatic.

37

37

Structural operational semantics of CCS

• A SOS consists of a set of inference rules to be used to define the transition relation of the LTS

$$\boxed{\frac{\text{premise}}{\text{conclusion}}}$$

38

38

SOS_{CCS}

$\boxed{\text{nil}}$

- no action is executed
- no inference rule is needed
- this constant expresses the process termination (success or failure/deadlock)

$\boxed{a.P}$

- the prefix of the term is an action
- no premise needs
- it expresses the sequentiality: the action a is executed and after the behavior of P is followed

act $\dfrac{}{a.P \xrightarrow{a} P}$

39

39

Nondeterministic choice

$\boxed{P + Q}$

- this process can behave as P or as Q
- the choice is non deterministic
- the operator + is associative

sum_1 $\dfrac{P \xrightarrow{a} P'}{P + Q \xrightarrow{a} P'}$ sum_2 $\dfrac{Q \xrightarrow{a} Q'}{P + Q \xrightarrow{a} Q'}$

40

40

Process name

$\boxed{C}$

- process name or constant definition
- it expresses recursion
- the meaning of *C* is that of the associated process

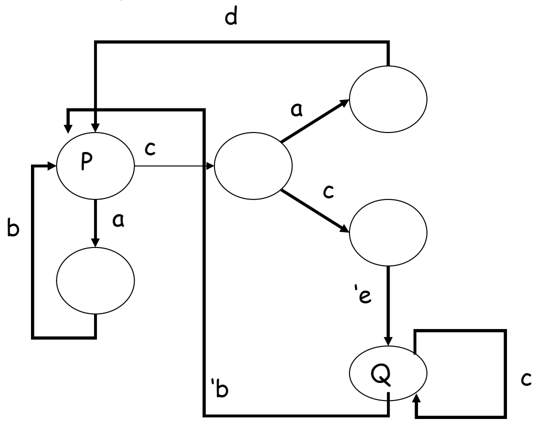
$$\text{con} \quad \frac{P \xrightarrow{a} P'}{C \xrightarrow{a} P'} \quad \text{if } C := P$$

41

41

Example

P := a.b.P + c. (a.d.P + c.'e.Q)
Q := c.Q + 'b.P



LTS(P)

42

42

Parallelism

$\boxed{P \mid Q}$

- the actions of P and Q are interleaved
- it is possible to perform a synchronous communication through an input action of P and an output action of Q (and vice versa)

- the operator is associative

43

43

(cont.)

par_1 $\dfrac{P \xrightarrow{a} P'}{P \mid Q \xrightarrow{a} P' \mid Q}$ par_2 $\dfrac{Q \xrightarrow{b} Q'}{P \mid Q \xrightarrow{b} P \mid Q'}$

interleaving

com $\dfrac{P \xrightarrow{a} P' \quad Q \xrightarrow{'a} Q'}{P \mid Q \xrightarrow{\tau} P' \mid Q'}$ $a \in Vis$

synchronization

44

44

(cont.)

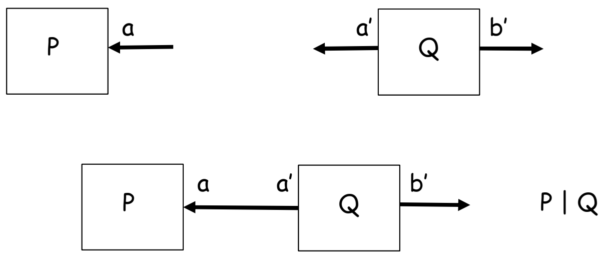
- a and ‘a are corresponding actions
- processes can communicate or behave independently
- when communicating P and Q perform the non observable action τ
- communication is always synchronous and between a pair of processes

45

45

(cont.)

The parallel operator composes processes through channels with corresponding names



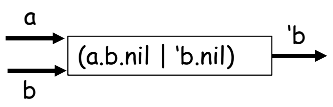
46

46

Example

$$\begin{tikzcd}
 & \text{a.b.nil} \mid \text{'b.nil} \arrow[dl, "a"'] \arrow[dr, "\text{'b}"] & \\
\text{b.nil} \mid \text{'b.nil} \arrow[d, "b"] \arrow[r, "\text{'b}"] \arrow[dd, bend right, "\tau"'] & \text{b.nil} \mid \text{nil} \arrow[d, "b"] & \text{a.b.nil} \mid \text{nil} \arrow[l, "a"'] \\
\text{nil} \mid \text{'b.nil} \arrow[r, "\text{'b}"] & \text{nil} \mid \text{nil} & \\
 & &
\end{tikzcd}$$

LTS (a.b.nil | b'.nil)



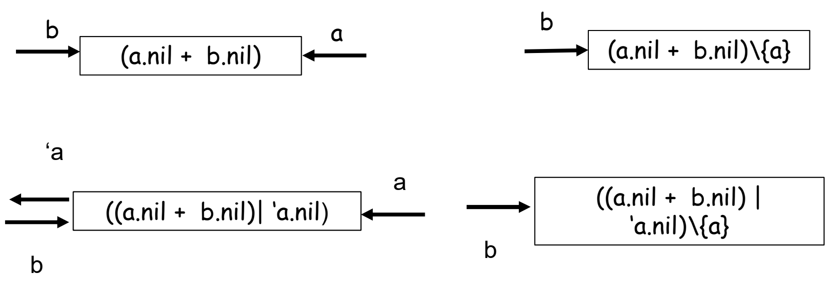
Restriction

$\boxed{\text{P\textbackslash L}}$

- L ⊆ Vis
- P can perform visible actions in L
- P\L cannot perform visible actions in L
- if P is a parallel process, its communication channels become local

$$\boxed{\text{res} \qquad \frac{P \xrightarrow{a} P'}{P\backslash L \xrightarrow{a} P'\backslash L} \qquad \text{se } a \notin (L \cup \text{'}L)}$$

Examples



49

49

Relabeling

$\boxed{P\,[f]}$

- f: Act $\longrightarrow$ Act by which all the actions of Act are relabeled
- it can be used to manage modularity

$$\boxed{\text{rel} \quad \frac{P \xrightarrow{a} P'}{P[f] \xrightarrow{f(a)} P'[f]}}$$

$$f(a) = \begin{cases} b_i \text{ if } a = a_i \text{ for some } i \\ a \text{ otherwise} \end{cases}$$

Property:

$f(\tau) = \tau$

50

50

CCS expressiveness

CCS is Turing-equivalent

- infinite state processes exist

properties are decidable if the transition system is finite

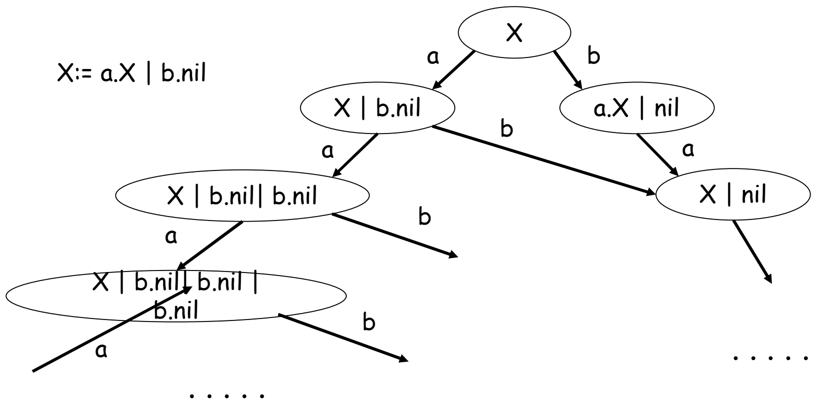
- Note: LTS(p) is finite, not p
- Syntactic restrictions can assure that LTS is finite

51

51

Infinite LTS

X:= a.X | b.nil



No finite state automata behaves as LTS(X)

52

52

Satisfaction (2)

- Given the transition system representing the process p, LTS(p) satisfies property ϕ, iff ϕ is verified in its initial state.
- The initial state of LTS(p) corresponds to p.
- LTS(p) represents the universe in which all properties of must hold and from its initial state all other states are reachable.

53

53

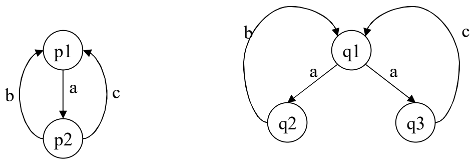
Examples

Are the formulae ϕ , Ψ and ξ true for the transition systems LTS(p1) and LTS(q1)?

$\phi=[a](\langle b\rangle tt \wedge \langle c\rangle tt)$

$\Psi= [a](\langle b\rangle tt \vee \langle c\rangle tt)$

$\xi=\langle a\rangle[b]ff$



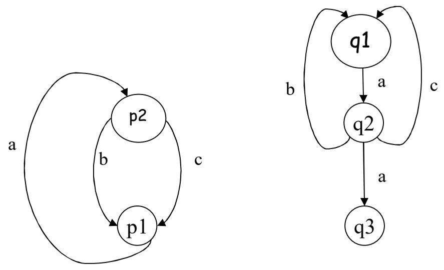
54

54

Examples (2)

- And do LTS(p_2) and LTS(q_1) satisfy the formulae ϕ and ξ ?

ϕ=[a](<b>tt $\wedge$ <c>tt)
ξ=<a>[b]ff



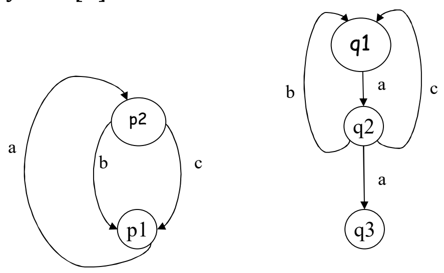
55

55

Examples (2)

- And do LTS(p_2) and LTS(q_1) satisfy the formulae ϕ and ξ ?

ϕ=[a](<b>tt $\wedge$ <c>tt)
ξ=<a>[b]ff



- p2 $\models \phi$ e q1 $\models \phi$, mentre p2 $\not\models \xi$ e q1 $\not\models \xi$.

56

56

MC application example

- Security: Needham-Schroeder encryption protocol
  - This is one of the two communication protocols intended for use over an insecure network, both proposed by Roger Needham and Michael Schroeder. In particular, we refer to
  - The *Needham–Schroeder Public-Key Protocol (1978)*, based on public-key cryptography. This protocol is intended to provide mutual authentication between two parties communicating on a network, but in its proposed form is insecure.

57

57

An attack on the protocol

- The protocol is vulnerable to a man-in-the-middle attack. If an impostor can persuade A to initiate a session with him, he can relay the messages to B and convince B that he is communicating with A.
- The attack was first described in a 1995 paper by Gavin Lowe.
- The error remained undiscovered for 17 years

58

58

How the error was discovered

- In the same paper Lowe fixed the man-in-the-middle attack and described a fixed version of the scheme, referred to as the **Needham–Schroeder–Lowe protocol**.
- To discover the error Lowe modeled the protocol by CSP and built the corresponding LTS
- A model checker, called FDR, revealed the error by checking a TL formula expressing the correctness of the protocol.

59

59

Example

- Consider a simple communication protocol where
    - the *sender* process receives a message from outside and transmits it to the *medium* process
    - *medium* either in turn transmits the message to the *receiver* process or loses it, in this case it asks for a retransmission
    - *receiver* transmits the message outside and acknoledges the sender for the end of transmission; after the ack, sender can accept a new message from outside.

60

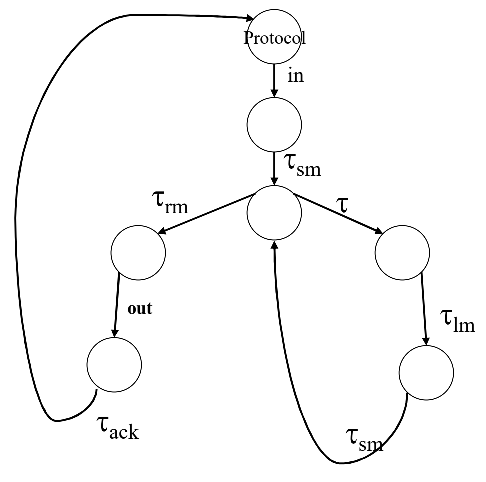
60

Characteristics

- Parallelism:
  - Three concurrent processes Sender, Medium and Receiver
- Communication:
  - The messages are exchanged between processes
- Nondeterminism:
  - The medium may either retransmit the message or loose it

CCS description

Protocol:= (Sender | Medium |Receiver)\ {sm,lm,rm,ack}
Sender := in.’sm.Send1
Send1 :=lm.’sm.Send1 + ack.Protocol
Medium :=sm.(‘rm.Medium + τ.’lm.Medium)
Receiver :=rm.’out.’ack.Receiver



LTS(Protocol)

63

63

Quantitative time representation in system specification

Timed CCS

64

64

Temporal properties

- In real time systems time is fundamental; in addition, time should be precisely measured
- A temporal property requiring that an action has to follow another one can be specified in mu-calculus, but a property requiring that an action has to follow a in given amount of time cannot be specified.

65

65

Quantitative time

- Both property specification language and system specification language must be extended, but

NB: every extension may induce a complexity blow-up.

66

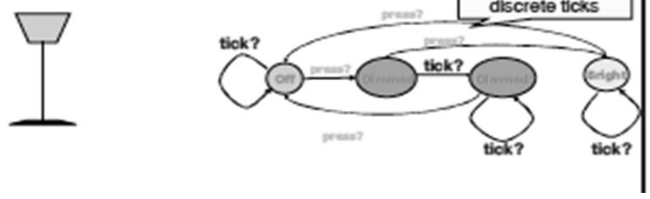
66

Discrete time

- Time is an event of the system
  - Only addition of a tick event
- Adequate for synchronous systems evolving in known steps
- Inadequate for many asynchronous distributed systems

Discrete time

Intelligent light switch
- Press button twice quickly to switch to bright
- Press it once to switch to dimmed
- If light is on or dimmed, pressing button switches light off



It is possible either to let flow the time (at least a tick) or to press immediately the button

Timed CCS syntax

$$\boxed{P ::= nil \mid \alpha.P \mid P{+}P \mid P|P \mid P\backslash L \mid P[f] \mid C \mid \delta.P}$$

Where δ is any nonnegative number

69

69

SOS of TCCS

$$\text{act1}\ \frac{}{a.P \xrightarrow{\alpha} P} \qquad \text{con1}\ \frac{P \xrightarrow{\alpha} P'}{C \xrightarrow{\alpha} P'}\quad C{:=}P$$

$$\text{sum1}\ \frac{P \xrightarrow{\alpha} P'}{P{+}Q \xrightarrow{\alpha} P'} \qquad \text{sum2}\ \frac{Q \xrightarrow{\alpha} Q'}{P{+}Q \xrightarrow{\alpha} Q'}$$

$$\text{par1}\ \frac{P \xrightarrow{\alpha} P'}{P|Q \xrightarrow{\alpha} P'|Q} \qquad \text{par2}\ \frac{Q \xrightarrow{\alpha} Q'}{P|Q \xrightarrow{\alpha} P|Q'}$$

$$\text{com1}\ \frac{P \xrightarrow{\alpha} P' \quad Q \xrightarrow{`\alpha} Q'}{P|Q \xrightarrow{\tau} P'|Q'}$$

$$\text{res1}\ \frac{P \xrightarrow{\alpha} P'}{P\backslash L \xrightarrow{\alpha} P'\backslash L}\quad \alpha \notin (L \cup `L) \qquad \text{rel1}\ \frac{P \xrightarrow{\alpha} P'}{P[f] \xrightarrow{f(\alpha)} P'[f]}$$

70

70

SOS of TCCS

$$\text{ONE}\,\frac{}{d.P \xrightarrow{1} (d-1).P}\,d \geq 1 \qquad \text{ACT}\,\frac{}{\alpha.P \xrightarrow{1} \alpha.P}\,\alpha \neq \tau \qquad \text{REL}\,\frac{P \xrightarrow{1} P'}{P[f] \xrightarrow{1} P'[f]}$$

$$\text{SUM}\,\frac{P \xrightarrow{1} P' \quad Q \xrightarrow{1} Q'}{P + Q \xrightarrow{1} P' + Q'} \qquad \text{CON}\,\frac{P \xrightarrow{1} P'}{K \xrightarrow{1} P'}\,K \overset{\text{def}}{=} P \qquad \text{RES}\,\frac{P \xrightarrow{1} P'}{P \setminus L \xrightarrow{1} P' \setminus L}$$

$$\text{COM}\,\frac{P \xrightarrow{1} P' \quad Q \xrightarrow{1} Q'}{P \mid Q \xrightarrow{1} P' \mid Q'}\ \text{if } P \mid Q \not\xrightarrow{\tau}$$

71

71

Notes

- Note that also delays longer than single-unit time can be directly specified
- Visible actions can be delayed
- Non observable actions cannot be delayed

72

72

Discussion

- R:= τ.nil + δ(2).b.nil
  - R may transform into nil, but not into τ.nil + δ(1).b.nil
- Q:= a.nil + δ(2).b.nil may transform into
  - a.nil + δ(1).b.nil
- S:= (δ(2).b.nil | a.nil) | 'a.nil may transform only into
  - (δ(2).b.nil | nil) | nil

73

73

TIMED PROPERTIES

Timed mu-calculus

74

74

Timed mu-calculus

- Mu-calculus syntax is extended by

$$\langle d\rangle\phi,\ \ [d]\phi.$$

- Whose meanings are
  - It is possible to delay *d* time units and then satisfy Φ
  - Whenever it is possible to delay *d* time units then Φ must be satisfied
  - respectively

75

75

Notes

- Independently from the used operator (<> or []), after d time-units Φ must be satisfied
- Remind that it is possible to have processes that cannot be delayed (those performing a non observable action).
  - In this case [d] Φ is always satisfied, while <d> Φ is never satisfied

76

76

CAAL framework

-
- http://caal.cs.aau.dk

77

77

Continuos time representation in system specification

Timed automata

78

78

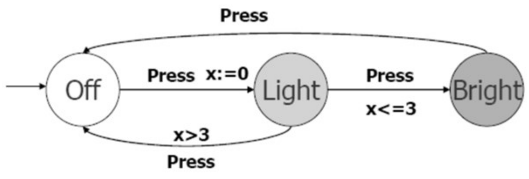
Continuos time

- A thick value does not exist in the system
- Time is expressed by continuous variables, starting from 0 and modifying their value as increasing real numbers.
- The values of these variables can be seen as part of each state of the system
- In the following solution the specification of the system is given through a state machine not a language

79

79

A solution



Solution: Add a real-valued clock **x**

Adding continuous variables to state machines

80

80

- After a *press*, the time flows (x is initially set to 0)
- The interesting interval between two successive *press events* is set to be between 0 and 3
- When more than 3 time units are passed, a new press causes the light switched off
- When at most 3 sec are passed, after a new press the light becomes bright.

81

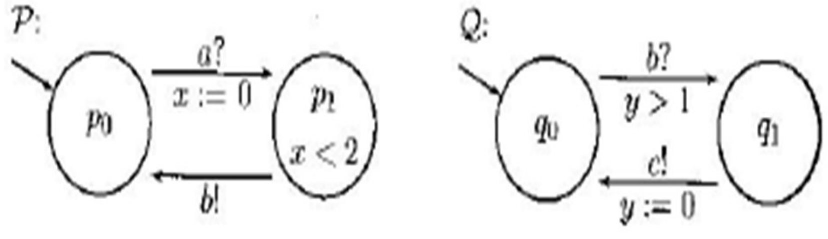
81

- Concurrency and communication are expressed by means of the parallel composition and restriction (CCS operators) of timed automata

82

82

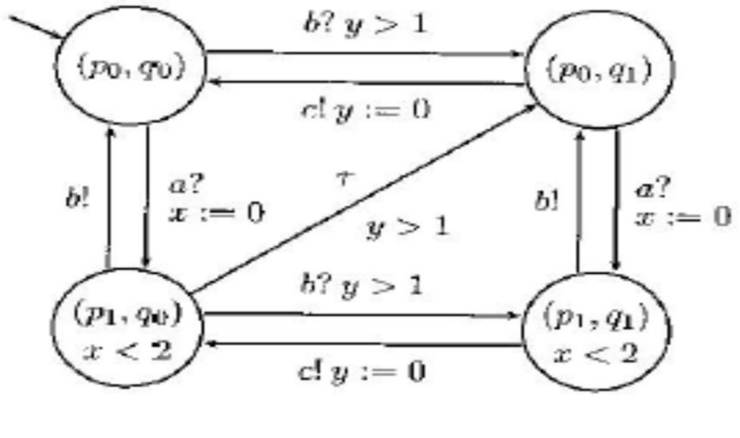
Example: Timed Buffer



83

83

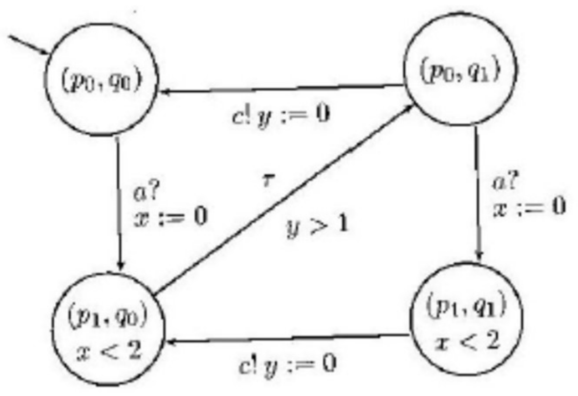
Cont.: P||Q



84

84

Cont.: the network (P||Q)\{b}



85

85

Uppaal framework

- https://uppaal.org/

86

86