Formal Methods for Computer System Design and Analysis

Diego Latella

(http://www.isti.cnr.it/People/D.Latella)

Consiglio Nazionale delle Ricerche
Ist. di Scienza e Tecnologie dell'Informazione "A. Faedo"
Formal Methods && Tools Lab

SEFM 2010

These slides are available at: http://www.sefm2010.isti.cnr.it/school/docs/introduction_and_motivations_latella.pdf

Outline

Outline

1. Background

Outline

1. Background
   - Engineering tradition;

Outline

1. Background
   - Engineering tradition;
   - (Notations for) Design Specifications;

Outline

1. Background
   - Engineering tradition;
   - (Notations for) Design Specifications;
   - (Notations for) Technical Specifications;

Outline

1. Background
   - Engineering tradition;
   - (Notations for) Design Specifications;
   - (Notations for) Requirement Specifications;

Outline

1. Background
   - Engineering tradition;
   - (Notations for) Design Specifications;
   - (Notations for) Requirement Specifications;
2. Formal Methods for Software Engineering;

Outline

1. Background
   - Engineering tradition;
   - (Notations for) Design Specifications;
   - (Notations for) Requirement Specifications;

2. Formal Methods for System Engineering;

Outline

1. Background
   - Engineering tradition;
   - (Notations for) Design Specifications;
   - (Notations for) Requirement Specifications;
2. Formal Methods for System Engineering;
3. Example: Process Algebraic approach to System Modelling;

Outline

1. Background
   - Engineering tradition;
   - (Notations for) Design Specifications;
   - (Notations for) Requirement Specifications;
2. Formal Methods for System Engineering;
3. Example: Process Algebraic approach to System Modelling;
4. Example: Temporal Logic approach to System Requirement Specification

Outline

1. Background
   - Engineering tradition;
   - (Notations for) Design Specifications;
   - (Notations for) Requirement Specifications;
2. Formal Methods for System Engineering;
3. Example: Process Algebraic approach to System Modelling;
4. Example: Temporal Logic approach to System Requirement Specification and Model-checking;

Outline

1. Background
   - Engineering tradition;
   - (Notations for) Design Specifications;
   - (Notations for) Requirement Specifications;
2. Formal Methods for System Engineering;
3. Example: Process Algebraic approach to System Modelling;
4. Example: Temporal Logic approach to System Requirement Specification and Model-checking;
5. Success Stories;

Outline

1. Background
   - Engineering tradition;
   - (Notations for) Design Specifications;
   - (Notations for) Requirement Specifications;
2. Formal Methods for System Engineering;
3. Example: Process Algebraic approach to System Modelling;
4. Example: Temporal Logic approach to System Requirement Specification and Model-checking;
5. Success Stories;
6. Extensions;

WARNING!!

WARNING!!

This presentation is

WARNING!!

This presentation is

- tutorial

WARNING!!

This presentation is

- tutorial
- informal

WARNING!!

This presentation is

- tutorial
- informal
- not always rigorous

WARNING!!

This presentation is

- tutorial
- informal
- not always rigorous, and
- quite incomplete !!

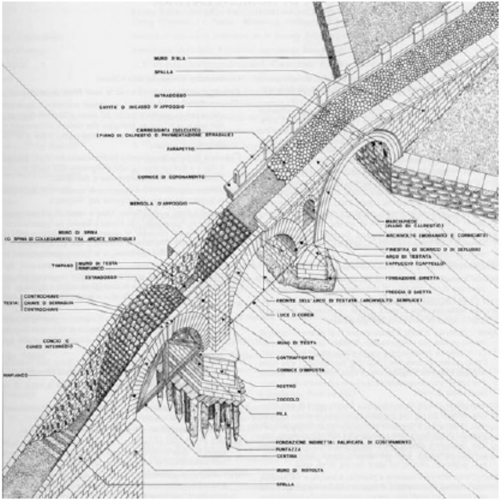
Background - Engineering

"All engineering disciplines make progress by employing mathematically based notations and methods."

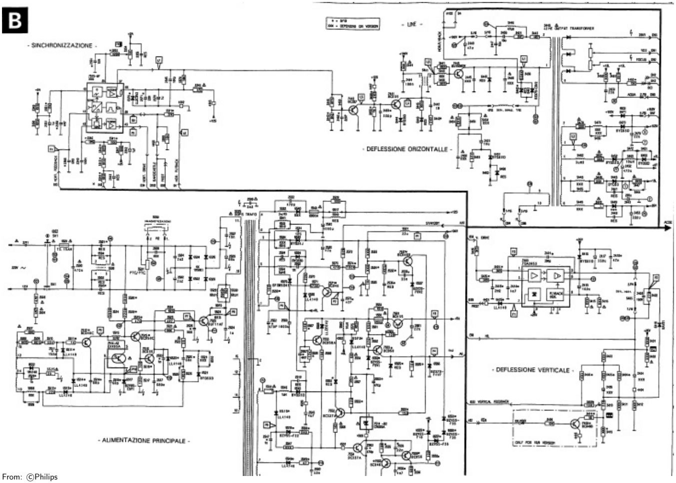
[C. Jones 2000]

Background - Engineering - Notations

Graphical

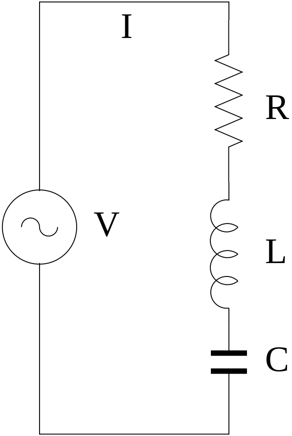


©M. Mazzoleni, L. Jurina "PONTI IN MURATURA: DIFETTI E PATOLOGIE", CIAS 2006, Bolzano



From: ©Philips

Graphical



- **Basic components**
- **Ways for composing them**

- **Basic components**
  e.g. Resistors, Inductances, Capacitors

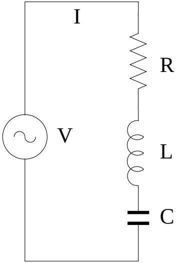
- **Ways for composing them**

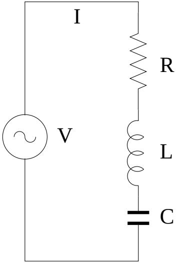
- **Basic components**
  e.g. Resistors, Inductances, Capacitors

- **Ways for composing them**
  e.g. SERIES, PARALLEL

Background - Engineering - Notations

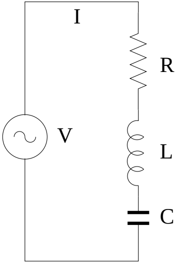
Graphical





circuit definition:

CIRCUIT RLC $(x_v, x_r, x_l, x_c) \triangleq$
CONNECT $(x_v$, SERIES (RES(x_r), IND(x_l), CAP(x_c)))



circuit definition:

CIRCUIT RLC $(x_v, x_r, x_l, x_c) \triangleq$
CONNECT (x_v, SERIES (RES(x_r), IND(x_l), CAP(x_c)))

circuit use (instantiation):

RLC(**V,R,L,C**)

Mathematically based Notations

Mathematically based Notations

- Rigorous (Formal) Syntax

- Rigorous (Formal) Semantics

Mathematically based Notations

- Rigorous (Formal) Syntax
  - (International) Standards for graphical notations
  - $\sqrt{75 + :}$ $\frac{z}{0}$
- Rigorous (Formal) Semantics

Mathematically based Notations

- Rigorous (Formal) Syntax
  - (International) Standards for graphical notations
  - $\sqrt{75 + :} \quad \frac{2}{0}$
- Rigorous (Formal) Semantics

Mathematically based Notations

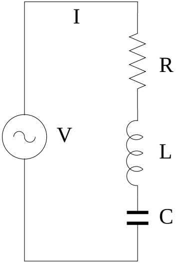
- Rigorous (Formal) Syntax
  - (International) Standards for graphical notations
  - ~~$\sqrt{75 + :} \quad \frac{2}{0}$~~
- Rigorous (Formal) Semantics
  - Set Theory, Relations and Functions
  - Continuous Mathematics
    - *Metric Spaces*
    - *Differential Calculus and Function Analysis*
    - *Linear Algebra*
    - *Differential Equations*
  - Mathematical Logic
  - *... and much more!*

Background - Engineering - Notations

Semantics: abstract definitions

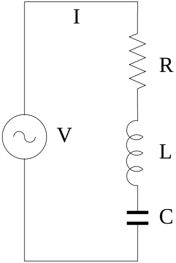
Background - Engineering - Notations

Semantics: abstract definitions



RLC (**V**, **R**, **L**, **C**)

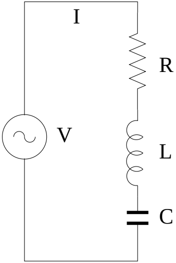
Semantics: abstract definitions



RLC (**V**, **R**, **L**, **C**)

Using the *Laws of Physics* (Kirchhoff Voltages Law) we can give an *abstract* and *rigorous description* of the relationship between the current and voltage at any time istant.

Semantics: abstract definitions



RLC (**V**, **R**, **L**, **C**)

Using the *Laws of Physics* (Kirchhoff Voltages Law) we can give an *abstract* and *rigorous description* of the relationship between the current and voltage at any time istant.

$$v(t) = R \cdot i(t) + L \cdot \frac{d\, i(t)}{d\, t} + \frac{1}{C} \cdot \int_{-\infty}^{t} i(x)\, d\, x$$

Semantics: methods for formal manipulation

$$v(t) = R \cdot i(t) + L \cdot \frac{d\, i(t)}{d\, t} + \frac{1}{C} \cdot \int_{-\infty}^{t} i(x)\, d\, x$$

Semantics: methods for formal manipulation

$$v(t) = R \cdot i(t) + L \cdot \frac{d\, i(t)}{d\, t} + \frac{1}{C} \cdot \int_{-\infty}^{t} i(x)\, d\, x$$

$\Leftrightarrow$ [Relationship between charge and current $i(t) = \frac{d\, q(t)}{d\, t}$]

$$v(t) = R \cdot i(t) + L \cdot \frac{d\, i(t)}{d\, t} + \frac{1}{C} \cdot \int_{-\infty}^{t} i(x)\, d\, x$$

$\Leftrightarrow$ [Relationship between charge and current $i(t) = \frac{d\, q(t)}{d\, t}$]

$$v(t) = R \cdot \frac{d\, q(t)}{d\, t} + L \cdot \frac{d^2\, q(t)}{d\, t} + \frac{1}{C} \cdot \int_{-\infty}^{t} \frac{d\, q(x)}{d\, x}\, d\, x$$

$$v(t) = R \cdot i(t) + L \cdot \frac{d\,i(t)}{d\,t} + \frac{1}{C} \cdot \int_{-\infty}^{t} i(x)\,d\,x$$

$\Leftrightarrow$ [Relationship between charge and current $i(t) = \frac{d\,q(t)}{d\,t}$]

$$v(t) = R \cdot \frac{d\,q(t)}{d\,t} + L \cdot \frac{d^2\,q(t)}{d\,t} + \frac{1}{C} \cdot \int_{-\infty}^{t} \frac{d\,q(x)}{d\,x}\,d\,x$$

$\Leftrightarrow$ [Differential/integral calculus]

$$v(t) = R \cdot i(t) + L \cdot \frac{d\,i(t)}{d\,t} + \frac{1}{C} \cdot \int_{-\infty}^{t} i(x)\,d\,x$$

$\Leftrightarrow$ [Relationship between charge and current $i(t) = \frac{d\,q(t)}{d\,t}$]

$$v(t) = R \cdot \frac{d\,q(t)}{d\,t} + L \cdot \frac{d^2\,q(t)}{d\,t} + \frac{1}{C} \cdot \int_{-\infty}^{t} \frac{d\,q(x)}{d\,x}\,d\,x$$

$\Leftrightarrow$ [Differential/integral calculus]

$$v(t) = R \cdot \frac{d\,q(t)}{d\,t} + L \cdot \frac{d^2\,q(t)}{d\,t} + \frac{1}{C} \cdot q(t)$$

Background - Engineering - Notations

Semantics: methods for formal manipulation

$$v(t) = R \cdot i(t) + L \cdot \frac{d\,i(t)}{d\,t} + \frac{1}{C} \cdot \int_{-\infty}^{t} i(x)\,d\,x$$

$\Leftrightarrow$ [Relationship between charge and current $i(t) = \frac{d\,q(t)}{d\,t}$]

$$v(t) = R \cdot \frac{d\,q(t)}{d\,t} + L \cdot \frac{d^2\,q(t)}{d\,t} + \frac{1}{C} \cdot \int_{-\infty}^{t} \frac{d\,q(x)}{d\,x}\,d\,x$$

$\Leftrightarrow$ [Differential/integral calculus]

$$v(t) = R \cdot \frac{d\,q(t)}{d\,t} + L \cdot \frac{d^2\,q(t)}{d\,t} + \frac{1}{C} \cdot q(t)$$

$\Leftrightarrow$ [Algebra, $L \neq 0$]

$$v(t) = R \cdot i(t) + L \cdot \frac{d\,i(t)}{d\,t} + \frac{1}{C} \cdot \int_{-\infty}^{t} i(x)\,d\,x$$

$\Leftrightarrow$ [Relationship between charge and current $i(t) = \frac{d\,q(t)}{d\,t}$]

$$v(t) = R \cdot \frac{d\,q(t)}{d\,t} + L \cdot \frac{d^2\,q(t)}{d\,t} + \frac{1}{C} \cdot \int_{-\infty}^{t} \frac{d\,q(x)}{d\,x}\,d\,x$$

$\Leftrightarrow$ [Differential/integral calculus]

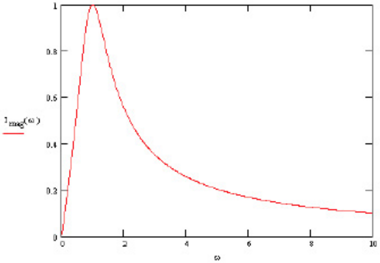
$$v(t) = R \cdot \frac{d\,q(t)}{d\,t} + L \cdot \frac{d^2\,q(t)}{d\,t} + \frac{1}{C} \cdot q(t)$$

$\Leftrightarrow$ [Algebra, $L \neq 0$]

$$\frac{d^2 q(t)}{dt} + \frac{R}{L} \cdot \frac{d\,q(t)}{d\,t} + \frac{1}{LC} \cdot q(t) = \frac{1}{L} \cdot v(t)$$

Background - Engineering - Notations

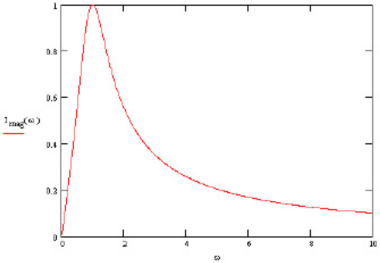
Semantics: identification and study of characteristic features



$I_{max}(\omega)$: current magnitude as a function of frequency ω

Background - Engineering - Notations

Semantics: identification and study of characteristic features



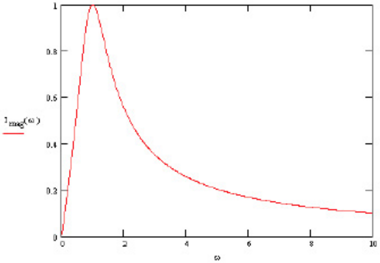
$I_{max}(\omega)$: current magnitude as a function of frequency ω

Resonance frequency:
the value ω_0 s.t. I_{max} has a peak in ω_0

Background - Engineering - Notations

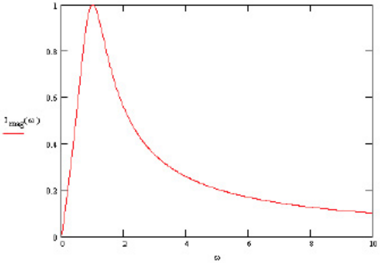
Computer support: mechanization of formal manipulation

Semantics: identification and study of characteristic features



$I_{max}(\omega)$: current magnitude as a function of frequency ω

Resonance frequency:
the value ω_0 s.t. I_{max} has a peak in ω_0



$I_{max}(\omega)$: current magnitude as a function of frequency ω

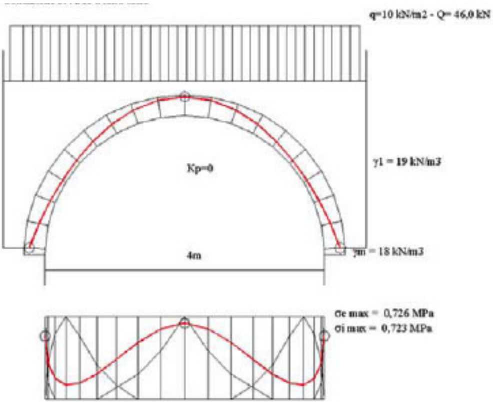
Resonance frequency:
the value ω_0 s.t. I_{max} has a peak in ω_0

Technical Specification

- Resonance frequency: 10.000 Hz
- ...

Sample technical specification: Audio Power Amplifier

Power output	25 W rms per channel
Load impedence	8 ohms
Total distorsion	$< 0.08\%$
Frequency response	$10 \div 50.000$ Hz ($+0.5$ dB, -2 dB)
Power requirements	
Power requirements	220 V (50 Hz)
Max power comsumption	160 W
Dimensions	430 mm W
	132 mm H
	247 mm D
Weight	4.5 Kg


q=10 kN/m2 - Q= 46,0 kN
Kp=0
γ1 = 19 kN/m3
4m
γm = 18 kN/m3
σe max = 0,726 MPa
σi max = 0,723 MPa

Sample technical specification: The bridge

Max load .... ??? t
Max wind speed .... ??? m/s
Oscillation Freq. .... ??? Hz
. . .

Background - Engineering - Technical Specifications

In general terms, *Technical Specifications*:

- are characteristic features of the system
- describe desirable requirements on the system
- help reasoning about the system
- should be met by the system

In general terms, *Technical Specifications*:

- are characteristic features of the system design
- describe desirable requirements on the system design
- help reasoning about the system design
- should be met by the system design

Civil, Naval, Nuclear, Electrical, Electronic (. . .) Engineers use notations for

- $\surd$ **technical specifications (requirements specifications)** as well as
- $\surd$ **design specifications/models**

which

- are strongly based on mathematics (and physics),
- are characterized by great and flexible descriptive power,
- allow for the formal manipulation of their objects
- are heavily supported by computer (software) tools (e.g. for relating models to technical specs)

Engineers are supposed to be aware of underlying theories but they are not required to completely master them.

Civil, Naval, Nuclear, Electrical, Electronic (. . .) Engineers use notations for

- √ **technical specifications (requirements specifications)** as well as
- √ **design specifications/models**

which

- are strongly based on mathematics (and physics),
- are characterized by great and flexible descriptive power,
- allow for the formal manipulation of their objects
- are heavily supported by computer (software) tools (e.g. for relating models to technical specs)

Engineers are supposed to be aware of underlying theories but they are not required to completely master them.

What about (Critical) Software Engineers?

Background - Software Engineering

- Software design and development is a matter of Art

Background - Software Engineering

- Software design and development is a matter of Art
- Software design and development is a matter of kraft-work

- Software design and development is a matter of Art
- Software design and development is a matter of kraft-work

but

Background - Software Engineering

- Software design and development is a matter of Art
- Software design and development is a matter of kraft-work

but

- Engineering requires (ingenuity, inspiration, ...) *and* the systematic application of sound techniques, with strong mathematical basis

Background - Software Engineering

- Software design and development is a matter of Art
- Software design and development is a matter of kraft-work

but

- Engineering requires (ingenuity, inspiration, ...) *and* the systematic application of sound techniques, with strong mathematical basis

⇒ Several techniques developed for *programming* (SP, OOP, EP ...)

Background - Software Engineering

- Software design and development is a matter of Art
- Software design and development is a matter of kraft-work

but

- Engineering requires (ingenuity, inspiration, ...) *and* the systematic application of sound techniques, with strong mathematical basis

⇒ Several techniques developed for *programming* (SP, OOP, EP ...)

⇒ Some (in-/semi-)formal techniques developed for *system design*

Background - Software Engineering

- Software design and development is a matter of Art
- Software design and development is a matter of kraft-work

but

- Engineering requires (ingenuity, inspiration, ...) *and* the systematic application of sound techniques, with strong mathematical basis

⇒ Several techniques developed for *programming* (SP, OOP, EP ...)

⇒ Some (in-/semi-)formal techniques developed for *system design*

What is the mathematical basis of SE?

Background - Formal Methods

- “All engineering disciplines make progress by employing mathematically based notations and methods.

Background - Formal Methods

"All engineering disciplines make progress by employing mathematically based notations and methods.
Research on 'formal methods' follows this model and attempts to identify and develop mathematical approaches that can contribute to the task of creating computer systems"

[C. Jones 2000]

Background - Formal Methods

"All engineering disciplines make progress by employing mathematically based notations and methods.
Research on 'formal methods' follows this model and attempts to identify and develop mathematical approaches that can contribute to the task of creating computer systems"

[C. Jones 2000]

Attempt to provide the (software) engineer with "concepts and techniques as thinking tools, which are clean, adequate, and convenient, to support him (or her) in describing, reasoning about, and constructing complex software and hardware systems"

[W. Thomas 2000]

Formal Methods

$$\text{Applying} \left\{ \frac{\text{Logic in}}{\text{Theoretical}} \right\} \text{Computer Science}$$

$$\text{Applying} \left\{ \frac{\text{Logic in}}{\text{Theoretical}} \right\} \text{Computer Science}$$

For Supporting System Engineering

$$\text{Applying} \left\{ \frac{\text{Logic in}}{\text{Theoretical}} \right\} \text{Computer Science}$$

For Supporting System Engineering

Emphasis on

$$\text{Applying} \left\{ \frac{\text{Logic in}}{\text{Theoretical}} \right\} \text{Computer Science}$$

For Supporting System Engineering

Emphasis on

- Construction

Applying $\left\{ \frac{\text{Logic in}}{\text{Theoretical}} \right\}$ Computer Science

For Supporting System Engineering

Emphasis on

- Construction
- Pragmatics

$$\text{Applying} \left\{ \frac{\text{Logic in}}{\text{Theoretical}} \right\} \text{Computer Science}$$

For Supporting System Engineering

Emphasis on

- Construction
- Pragmatics
- Automatic, often push-botton, Software Tool Support

$$\text{Applying} \left\{ \frac{\text{Logic in}}{\text{Theoretical}} \right\} \text{Computer Science}$$

For Supporting System Engineering

Emphasis on

- Construction
- Pragmatics
- Automatic, often push-botton, Software Tool Support

rather than

Formal Methods

Applying $\left\{ \frac{\text{Logic in}}{\text{Theoretical}} \right\}$ Computer Science

For Supporting System Engineering

Emphasis on

- Construction
- Pragmatics
- Automatic, often push-botton, Software Tool Support

rather than

- classical issues like completeness.

Formal Methods - OUR FOCUS

- Here we focus on concurrent systems
  - System: composed of (very) many components
  - Component: performs (very) simple tasks (often sequential)
  - Interaction: complex; difficult to understand; non-deterministic; subtle (race conditions, synchronization issues, dead-/live-locks, etc.)

However notice that

sound mathematical theories for non-concurrent, sequential (functional, imperative) programs exist

- the bulk of Computation Theory (Gödel, Turing, Church, etc)
- *formal semantics, e.g.*
  - Operational Semantics
    (based on abstract machines)
  - Denotational semantics
    (based on lattices, complete partial orders, fixpoint theory)
- *formal analysis, e.g.*
  - Hoare Logic
  - Cousot Abstract Interpretation

A
Labelled Transition Systems
Process Algebraic
Approach to
System Modelling
(design specification)

State-Transition Structures

Description of behaviour of a system

- A set of *states* which the system can be at
- A set of *transitions* which describe *how* a system can move from *which* state to *which* one

State-Transition Structures

Description of behaviour of a system

- A set of *states* which the system can be at
- A set of *transitions* which describe *how* a system can move from *which* state to *which* one

STATE	TRANSITION
The specific set of voltages at the components of a circuit at a given point in time	Any change of such values

Description of behaviour of a system

- A set of *states* which the system can be at
- A set of *transitions* which describe *how* a system can move from *which* state to *which* one

STATE	TRANSITION
The specific set of voltages at the components of a circuit at a given point in time	Any change of such values
The specific set of values of variables and execution points (PC) of the SW components of a distributed system at a given point in time (i.e. the system **state vector**)	Execution of a command (e.g. variable assignment)

State-Transition Structures

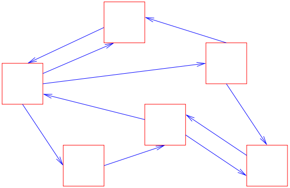
Description of behaviour of a system

- A set of *states* which the system can be at
- A set of *transitions* which describe *how* a system can move from *which* state to *which* one

STATE	TRANSITION
The specific set of voltages at the components of a circuit at a given point in time	Any change of such values
The specific set of values of variables and execution points (PC) of the SW components of a distributed system at a given point in time (i.e. the system **state vector**)	Execution of a command (e.g. variable assignment)
Being *free* or *in use* of a computing resource in a system	Granting (or refusing) a request of use

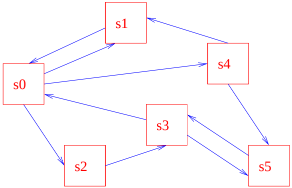
State -Transition Structures

- **Graphical notation**



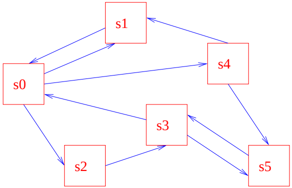
State -Transition Structures

- **Graphical notation**



State -Transition Structures

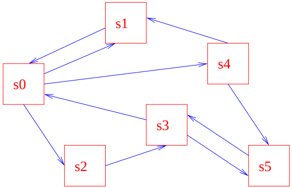
- **Graphical notation**



- **Mathematical definition**

State -Transition Structures

- **Graphical notation**



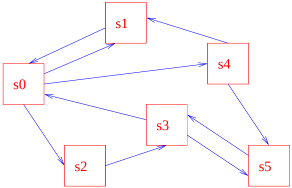
- **Mathematical definition**

A STS is a tuple $(S, \rightarrow)$ where:

- S is the set of states

State -Transition Structures

- **Graphical notation**



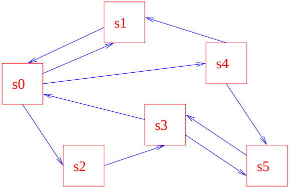
- **Mathematical definition**

  A STS is a tuple $(S, \rightarrow)$ where:

  - S is the set of states ($\{s_0, s_1, s_2, s_3, s_4, s_5\}$ in the example above)

State -Transition Structures

- **Graphical notation**

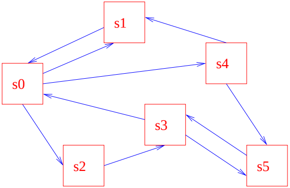


- **Mathematical definition**

A STS is a tuple $(S, \rightarrow)$ where:

- S is the set of states $(\{s_0, s_1, s_2, s_3, s_4, s_5\}$ in the example above)
- $\rightarrow \subseteq S \times S$ is the transition relation

- **Graphical notation**



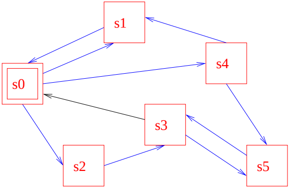
- **Mathematical definition**

A STS is a tuple $(S, \rightarrow)$ where:

- S is the set of states ($\{s_0, s_1, s_2, s_3, s_4, s_5\}$ in the example above)
- $\rightarrow \subseteq S \times S$ is the transition relation
  ($\rightarrow = \{(s_0, s_1), (s_1, s_0), (s_0, s_2), \ldots, (s_4, s_5)\}$ in the example above)

State -Transition Structures

- **Graphical notation**



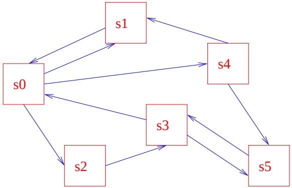
- **Mathematical definition**

A STS is a tuple $(S, \rightarrow, s_0)$ where:

  - S is the set of states $(\{s_0, s_1, s_2, s_3, s_4, s_5\}$ in the example above)
  - $\rightarrow \subseteq S \times S$ is the transition relation
    $(\rightarrow = \{(s_0, s_1), (s_1, s_0), (s_0, s_2), \ldots, (s_4, s_5)\}$ in the example above)
  - $s_0 \in S$ is the initial state

State -Transition Structures

- **Graphical notation**



- **Mathematical definition**

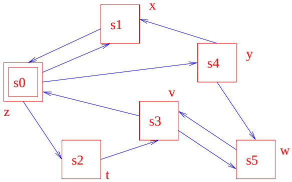
A STS is a tuple $(S, \rightarrow)$ where:

- S is the set of states ($\{s_0, s_1, s_2, s_3, s_4, s_5\}$ in the example above)
- $\rightarrow \subseteq S \times S$ is the transition relation ($\rightarrow = \{(s_0, s_1), (s_1, s_0), (s_0, s_2), \ldots, (s_4, s_5)\}$ in the example above)
- $s_0 \in S$ is the initial state

We write $s \rightarrow s'$ whenever $(s, s') \in \rightarrow$

State Labelled-Transition Structures

- **Graphical notation**

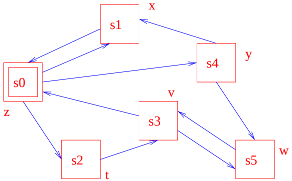


- **Mathematical definition**

A tuple $(S, A_s, L, \rightarrow, s_0)$ where:

- S is the set of states
- A_s is a set of state labels (in the example $\{x, y, v, w, t, z\}$)
- $L : S \longrightarrow A_s$ is a state-labelling function
    - *e.g.* $L(s)$ *is the* state vector *at* s,
    - *or* $L(s)$=*ok iff a given component is up and running in* s
    - *etc.*
- $\rightarrow \subseteq S \times S$ is the transition relation
- $s_0 \in S$ is the initial state

- **Graphical notation**



- **Mathematical definition**

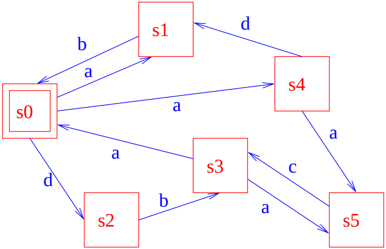
A tuple $(S, A_s, L, \rightarrow, s_0)$ where:

- S is the set of states
- A_s is a set of state labels (in the example $\{x, y, v, w, t, z\}$)
- $L : S \longrightarrow A_s$ is a state-labelling function
  - *e.g.* $L(s)$ *is the* state vector *at* s,
  - *or* $L(s)$=*ok iff a given component is up and running in* s
  - *etc.*
- $\rightarrow \subseteq S \times S$ is the transition relation
- $s_0 \in S$ is the initial state

- *Kripke Structure*: $L(s)$ is a set of *atomic propositions* holding in s

State-Transition Labelled Structures

- **Graphical notation**

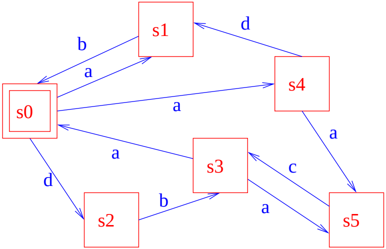


- **Mathematical definition**

  A tuple $(S, A_t, \rightarrow, s_0)$ where:
  - S is the set of states
  - A_t is a set of transition labels (actions) (in the example $\{a, b, c, d\}$)

    a may denote an interaction (e.g. synchronous communication) or a local operation (e.g. assignment)
  - $\rightarrow \subseteq S \times A_t \times S$ is the transition relation
  - $s_0 \in S$ is the initial state

- **Graphical notation**



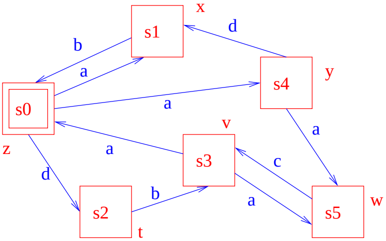
- **Mathematical definition**

  A tuple $(S, A_t, \rightarrow, s_0)$ where:
  - S is the set of states
  - A_t is a set of transition labels (actions) (in the example $\{a, b, c, d\}$)

    a may denote an interaction (e.g. synchronous communication) or a local operation (e.g. assignment)
  - $\rightarrow \subseteq S \times A_t \times S$ is the transition relation
  - $s_0 \in S$ is the initial state
- *Labelled Transition Systems (LTS)*

State and Transition Labelled Structures

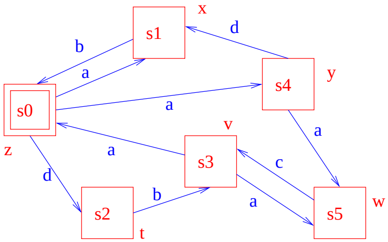
- **Graphical notation**



- **Mathematical definition**

  A tuple $(S, A_s, L, A_t, \rightarrow, s_0)$ where:
  - S is the set of states
  - A_s is a set of state labels
  - $L : S \longrightarrow A_s$ is a state-labelling function
  - $\rightarrow \subseteq S \times A_t \times S$
  - $s_0 \in S$ is the initial state

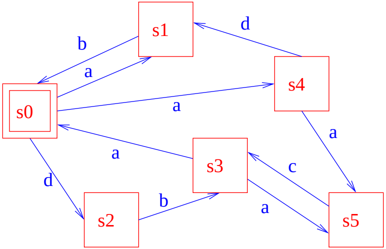
- **Graphical notation**



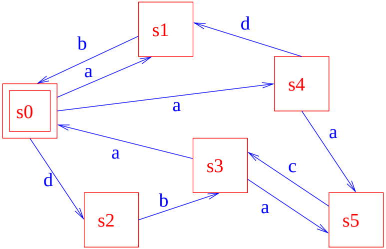
- **Mathematical definition**

  A tuple $(S, A_s, L, A_t, \rightarrow, s_0)$ where:
  - S is the set of states
  - A_s is a set of state labels
  - $L : S \longrightarrow A_s$ is a state-labelling function
  - $\rightarrow \subseteq S \times A_t \times S$
  - $s_0 \in S$ is the initial state
- *Doubly Labelled Transition Systems / Bi-Labelled Transition Systems ([De Nicola, Vaandeager]/ [Gnesi et al.])*

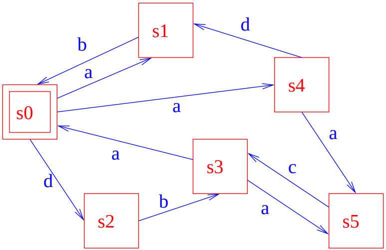
Example of Textual definition of a LTS



Example of Textual definition of a LTS



$$
\begin{aligned}
s0 &\triangleq a.s1 + d.s2 + a.s4 \\
s1 &\triangleq b.s0 \\
s2 &\triangleq b.s3 \\
s3 &\triangleq a.s0 + a.s5 \\
s4 &\triangleq d.s1 + a.s5 \\
s5 &\triangleq c.s3
\end{aligned}
$$

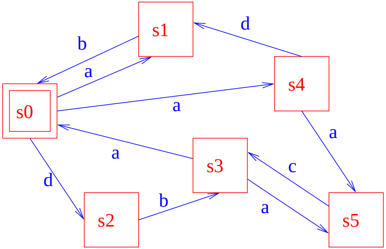


$$\begin{array}{lll} s0 & \triangleq & a.s1 + d.s2 + a.s4 \\ s1 & \triangleq & b.s0 \\ s2 & \triangleq & b.s3 \\ s3 & \triangleq & a.s0 + a.s5 \\ s4 & \triangleq & d.s1 + a.s5 \\ s5 & \triangleq & c.s3 \end{array}$$

Formal Syntax definition of *process* states

$$\begin{array}{llll} S & ::= & \mathbf{nil} & (\textit{no action}) \\ & \mid & \alpha.S & (\textit{action prefix}) \\ & \mid & S + S & (\textit{choice}) \\ & \mid & \alpha.X & (\textit{constant } X) \end{array}$$

Example of Textual definition of a LTS



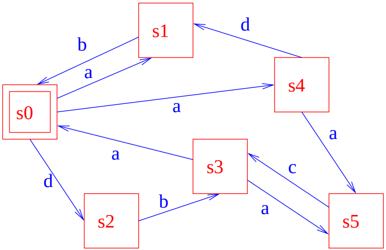
$$s0 \triangleq a.s1 + d.s2 + a.s4$$
$$s1 \triangleq b.s0$$
$$s2 \triangleq b.s3$$
$$s3 \triangleq a.s0 + a.s5$$
$$s4 \triangleq d.s1 + a.s5$$
$$s5 \triangleq c.s3$$

Formal Syntax definition of ***process*** **states**

$$
\begin{array}{lcll}
S & ::= & \mathbf{nil} & (\textit{no action}) \\
 & | & \alpha.S & (\textit{action prefix}) \\
 & | & S + S & (\textit{choice}) \\
 & | & \alpha.X & (\textit{constant } X)
\end{array}
$$

with actions $\alpha \in A_t$

Example of Textual definition of a LTS



$$
\begin{aligned}
s0 &\triangleq a.s1 + d.s2 + a.s4 \\
s1 &\triangleq b.s0 \\
s2 &\triangleq b.s3 \\
s3 &\triangleq a.s0 + a.s5 \\
s4 &\triangleq d.s1 + a.s5 \\
s5 &\triangleq c.s3
\end{aligned}
$$

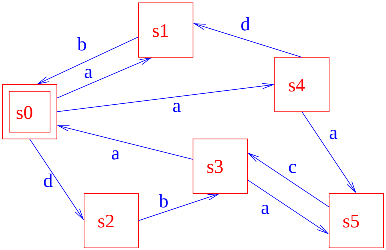
Formal Syntax definition of *process* states

$$
\begin{array}{lcll}
S & ::= & \mathbf{nil} & (\textit{no action}) \\
 & \mid & \alpha.S & (\textit{action prefix}) \\
 & \mid & S + S & (\textit{choice}) \\
 & \mid & \alpha.X & (\textit{constant } X)
\end{array}
$$

with actions $\alpha \in A_t$

and constants X defined via equations $X \triangleq S$

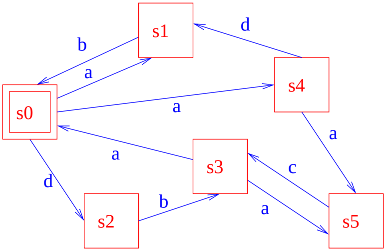
Example of Textual definition of a LTS



$$s0 \triangleq a.s1 + d.s2 + a.s4$$
$$s1 \triangleq b.s0$$
$$s2 \triangleq b.s3$$
$$s3 \triangleq a.s0 + a.s5$$
$$s4 \triangleq d.s1 + a.s5$$
$$s5 \triangleq c.s3$$

- **Basic components**
- **Ways for composing them**

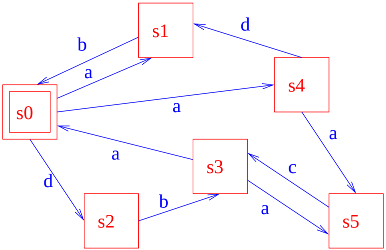
Example of Textual definition of a LTS



$$
\begin{aligned}
s0 &\triangleq a.s1 + d.s2 + a.s4 \\
s1 &\triangleq b.s0 \\
s2 &\triangleq b.s3 \\
s3 &\triangleq a.s0 + a.s5 \\
s4 &\triangleq d.s1 + a.s5 \\
s5 &\triangleq c.s3
\end{aligned}
$$

- **Basic components**
  e.g. Resistors, Inductances, Capacitors
- **Ways for composing them**

Example of Textual definition of a LTS

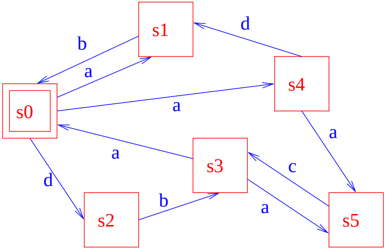


$$
\begin{aligned}
s0 &\triangleq a.s1 + d.s2 + a.s4 \\
s1 &\triangleq b.s0 \\
s2 &\triangleq b.s3 \\
s3 &\triangleq a.s0 + a.s5 \\
s4 &\triangleq d.s1 + a.s5 \\
s5 &\triangleq c.s3
\end{aligned}
$$

- **Basic components**
  e.g. **nil**, *Actions*

- **Ways for composing them**

Example of Textual definition of a LTS



$$
\begin{aligned}
s0 &\triangleq a.s1 + d.s2 + a.s4 \\
s1 &\triangleq b.s0 \\
s2 &\triangleq b.s3 \\
s3 &\triangleq a.s0 + a.s5 \\
s4 &\triangleq d.s1 + a.s5 \\
s5 &\triangleq c.s3
\end{aligned}
$$

- **Basic components**
  e.g. **nil**, *Actions*
- **Ways for composing them**
  e.g. action prefix operator $(\_.\_)$, *choice operator* $(\_ + \_)$

From algebraic terms to LTS via Formal Semantics.

$$s0 \triangleq a.s1 + d.s2 + a.s4$$
$$s1 \triangleq b.s0$$
$$s2 \triangleq b.s3$$
$$s3 \triangleq a.s0 + a.s5$$
$$s4 \triangleq d.s1 + a.s5$$
$$s5 \triangleq c.s3$$

From algebraic terms to LTS via Formal Semantics.

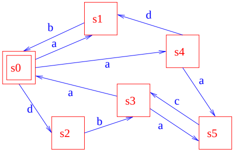
Formal Syntax definition (Grammar)

$$
\begin{aligned}
s0 &\triangleq a.s1 + d.s2 + a.s4\\
s1 &\triangleq b.s0\\
s2 &\triangleq b.s3\\
s3 &\triangleq a.s0 + a.s5\\
s4 &\triangleq d.s1 + a.s5\\
s5 &\triangleq c.s3
\end{aligned}
$$

From algebraic terms to LTS via Formal Semantics.

Formal Syntax definition (Grammar)

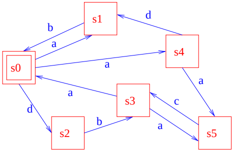
$s0 \triangleq a.s1 + d.s2 + a.s4$
$s1 \triangleq b.s0$
$s2 \triangleq b.s3$
$s3 \triangleq a.s0 + a.s5$
$s4 \triangleq d.s1 + a.s5$
$s5 \triangleq c.s3$



Formal Syntax definition
(Grammar)

$$
\begin{aligned}
s0 &\triangleq a.s1 + d.s2 + a.s4 \\
s1 &\triangleq b.s0 \\
s2 &\triangleq b.s3 \\
s3 &\triangleq a.s0 + a.s5 \\
s4 &\triangleq d.s1 + a.s5 \\
s5 &\triangleq c.s3
\end{aligned}
$$

Mathematical Objects
(LTS)

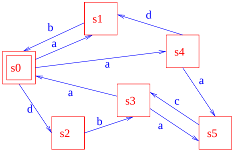


From algebraic terms to LTS via Formal Semantics.

Formal Syntax definition
(Grammar)

$$
\begin{aligned}
s0 &\triangleq a.s1 + d.s2 + a.s4 \\
s1 &\triangleq b.s0 \\
s2 &\triangleq b.s3 \\
s3 &\triangleq a.s0 + a.s5 \\
s4 &\triangleq d.s1 + a.s5 \\
s5 &\triangleq c.s3
\end{aligned}
$$

Mathematical Objects
(LTS)



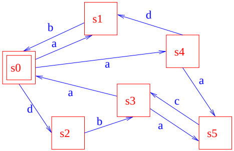
From algebraic terms to LTS via Formal Semantics.

Formal Syntax definition (Grammar)

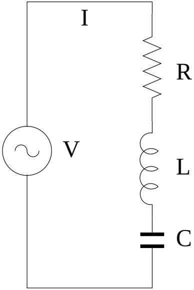
$$\begin{aligned} s0 &\triangleq a.s1 + d.s2 + a.s4 \\ s1 &\triangleq b.s0 \\ s2 &\triangleq b.s3 \\ s3 &\triangleq a.s0 + a.s5 \\ s4 &\triangleq d.s1 + a.s5 \\ s5 &\triangleq c.s3 \end{aligned}$$

Formal Semantics definition (Logic deduction system)

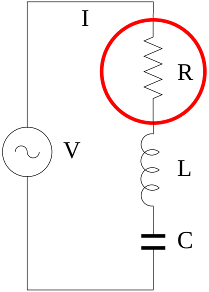
Mathematical Objects (LTS)



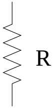
Back to the RLC circuit



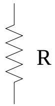
Focus on R

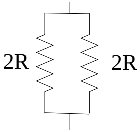


Focus on R

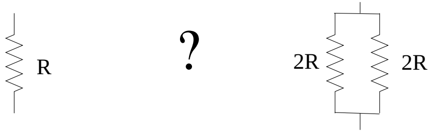


RES(**R**) and PARALLEL (RES(**2R**),RES(**2R**))

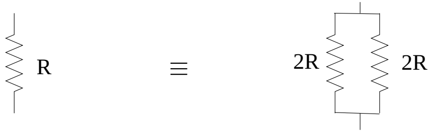




RES(**R**) and PARALLEL (RES(**2R**),RES(**2R**))



RES(**R**) ≡ PARALLEL (RES(**2R**),RES(**2R**))

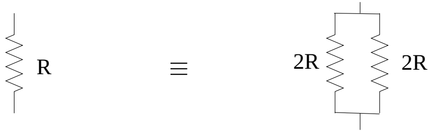


RES(**R**) $\equiv$ PARALLEL (RES(**2R**),RES(**2R**))

It can be proved:

$$Resistance(PARALLEL(RES(R_1), RES(R_2))) = \frac{1}{\frac{1}{R_1} + \frac{1}{R_2}}$$

RES(**R**) $\equiv$ PARALLEL (RES(**2R**),RES(**2R**))

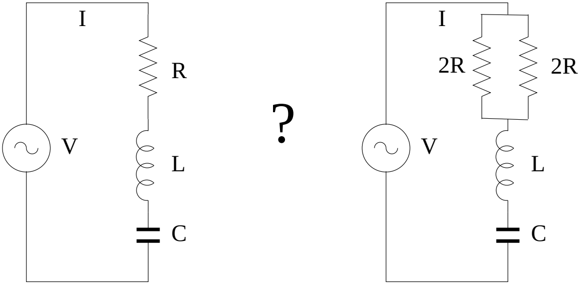


It can be proved:

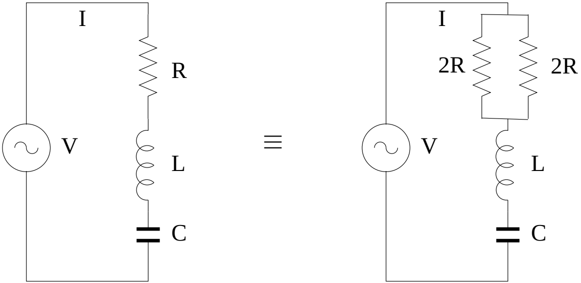
$$Resistance(PARALLEL(RES(R_1), RES(R_2))) = \frac{1}{\frac{1}{R_1}+\frac{1}{R_2}}$$

$$Resistance(PARALLEL_{j=1}^{k}(RES(R_j))) = \frac{1}{\sum_{j=1}^{k}\frac{1}{R_j}}$$

Replacing equivalent components ...

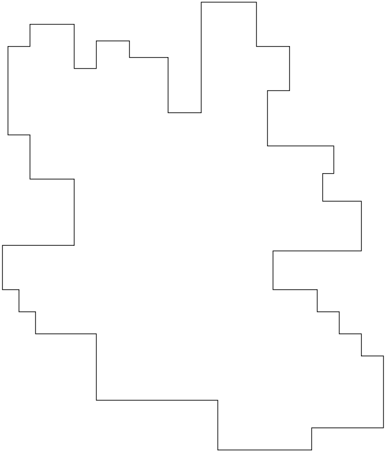


... brings to equivalent circuits

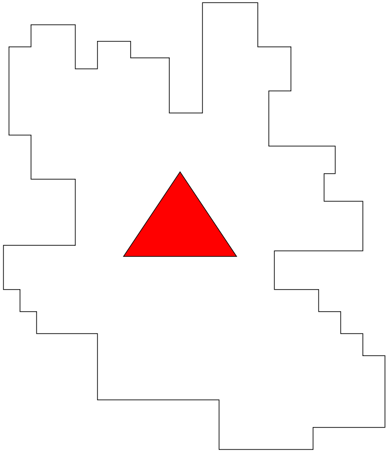


Congruence

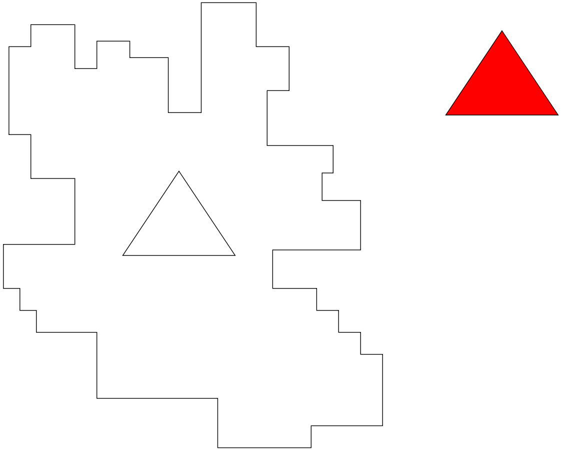
Congruence



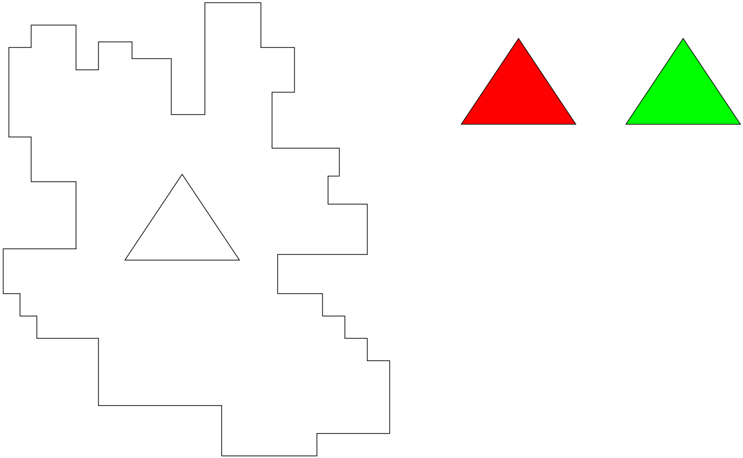
Congruence



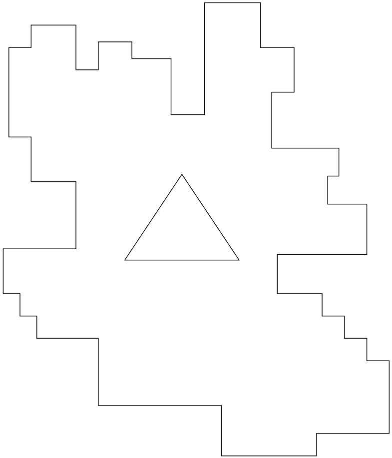
Congruence

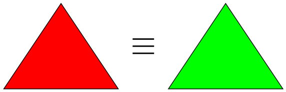


Congruence

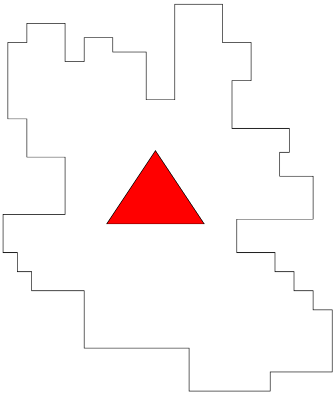


Congruence

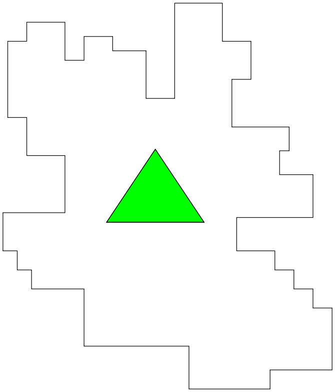




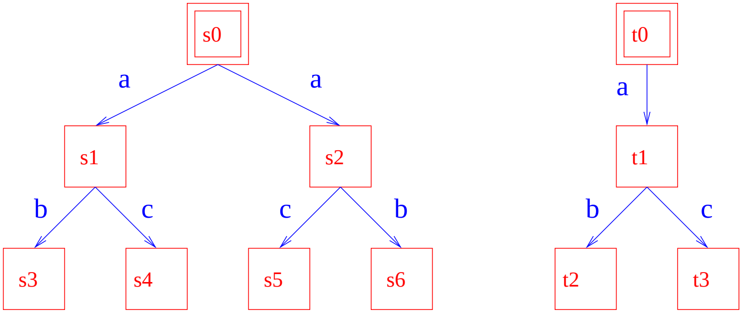
Congruence



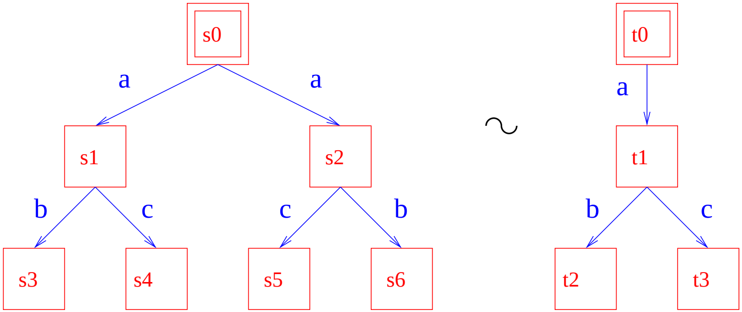
$\equiv$



LTS behaviour equivalence



LTS behaviour equivalence



(Strong) Bisimulation Equivalence

(Strong) Bisimulation Equivalence

Two states s and t are *Bisimulation Equivalent* ($s \sim t$) iff there exists *bisimulation relation* $\mathcal{B}$ s.t. $s \, \mathcal{B} \, t$

(Strong) Bisimulation Equivalence

Two states s and t are *Bisimulation Equivalent* ($s \sim t$) iff there exists *bisimulation relation* $\mathcal{B}$ s.t. $s \, \mathcal{B} \, t$

A binary relation $\mathcal{B}$ on the set of *states* is a *bisimulation relation* iff, for all s, t s.t. $s \, \mathcal{B} \, t$ and *transition labels* α:

(Strong) Bisimulation Equivalence

Two states s and t are *Bisimulation Equivalent* ($s \sim t$) iff there exists *bisimulation relation* $\mathcal{B}$ s.t. $s\,\mathcal{B}\,t$

A binary relation $\mathcal{B}$ on the set of *states* is a *bisimulation relation* iff, for all s, t s.t. $s\,\mathcal{B}\,t$ and *transition labels* α:

- whenever $s \xrightarrow{\alpha} s'$, there exists t' s.t. $t \xrightarrow{\alpha} t'$ and $s'\,\mathcal{B}\,t'$

(Strong) Bisimulation Equivalence

Two states s and t are *Bisimulation Equivalent* ($s \sim t$) iff there exists *bisimulation relation* $\mathcal{B}$ s.t. $s\,\mathcal{B}\,t$

A binary relation $\mathcal{B}$ on the set of *states* is a *bisimulation relation* iff, for all s, t s.t. $s\,\mathcal{B}\,t$ and *transition labels* α:

- whenever $s \xrightarrow{\alpha} s'$, there exists t' s.t. $t \xrightarrow{\alpha} t'$ and $s'\,\mathcal{B}\,t'$
- whenever $t \xrightarrow{\alpha} t'$, there exists s' s.t. $s \xrightarrow{\alpha} s'$ and $s'\,\mathcal{B}\,t'$

(Strong) Bisimulation Equivalence

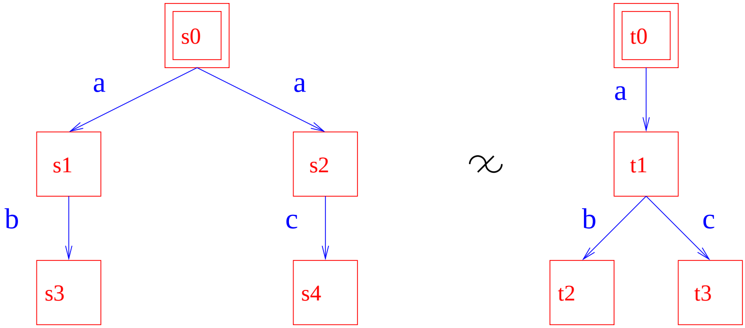
Two states s and t are *Bisimulation Equivalent* ($s \sim t$) iff there exists *bisimulation relation* $\mathcal{B}$ s.t. $s\,\mathcal{B}\,t$

A binary relation $\mathcal{B}$ on the set of *states* is a *bisimulation relation* iff, for all s, t s.t. $s\,\mathcal{B}\,t$ and *transition labels* α:

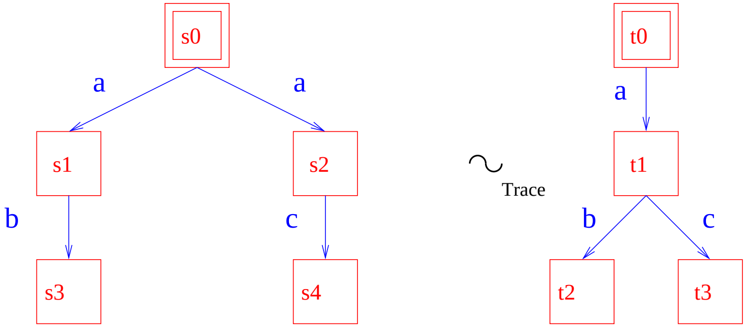
- whenever $s \xrightarrow{\alpha} s'$, there exists t' s.t. $t \xrightarrow{\alpha} t'$ and $s'\,\mathcal{B}\,t'$
- whenever $t \xrightarrow{\alpha} t'$, there exists s' s.t. $s \xrightarrow{\alpha} s'$ and $s'\,\mathcal{B}\,t'$

We usually refer to the *initial* states of two systems.

LTS behaviour equivalence



LTS behaviour equivalence



Algebraic Laws for Bisimulation Equivalence

Algebraic Laws for Bisimulation Equivalence

For all terms S, S_1, S_2

$$
\begin{aligned}
S + \mathbf{nil} &\sim S \\
S + S &\sim S \\
S_1 + S_2 &\sim S_2 + S_1 \\
S + (S_1 + S_2) &\sim (S + S_1) + S_2
\end{aligned}
$$

Algebraic Laws for Bisimulation Equivalence

For all terms S, S_1, S_2

$$\begin{array}{rcl} S + \mathbf{nil} & \sim & S \\ S + S & \sim & S \\ S_1 + S_2 & \sim & S_2 + S_1 \\ S + (S_1 + S_2) & \sim & (S + S_1) + S_2 \end{array}$$

Bisimulation Equivalence is actually a *congruence*:
If $S_1 \sim S_2$ then, for all S and α

$$\begin{array}{rcl} \alpha.S_1 & \sim & \alpha.S_2 \\ S + S_1 & \sim & S + S_2 \end{array}$$

Algebraic Laws for Bisimulation Equivalence

For all terms S, S_1, S_2

$$
\begin{array}{rcl}
S + \mathbf{nil} & \sim & S \\
S + S & \sim & S \\
S_1 + S_2 & \sim & S_2 + S_1 \\
S + (S_1 + S_2) & \sim & (S + S_1) + S_2
\end{array}
$$

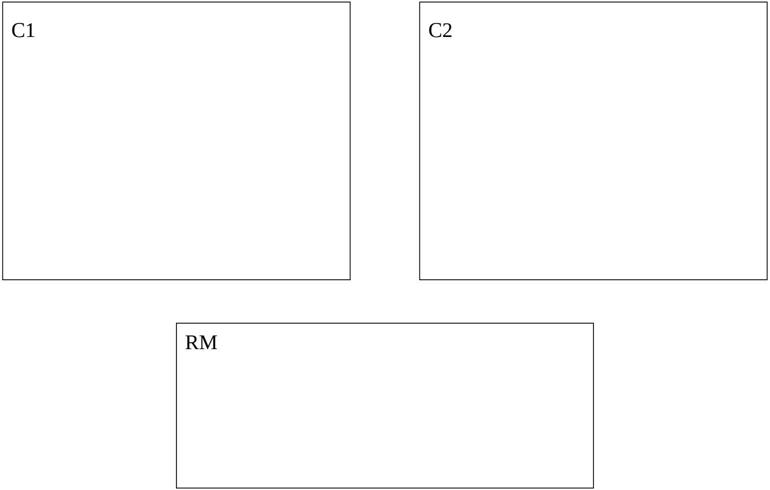
Bisimulation Equivalence is actually a *congruence*:
If $S_1 \sim S_2$ then, for all S and α

$$
\begin{array}{rcl}
\alpha.S_1 & \sim & \alpha.S_2 \\
S + S_1 & \sim & S + S_2
\end{array}
$$

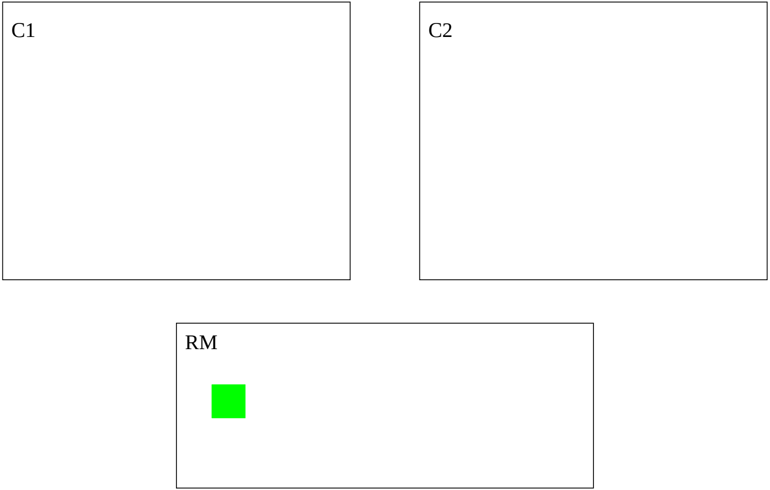
Expressions can be *reduced/simplified*!!

Parallel Composition

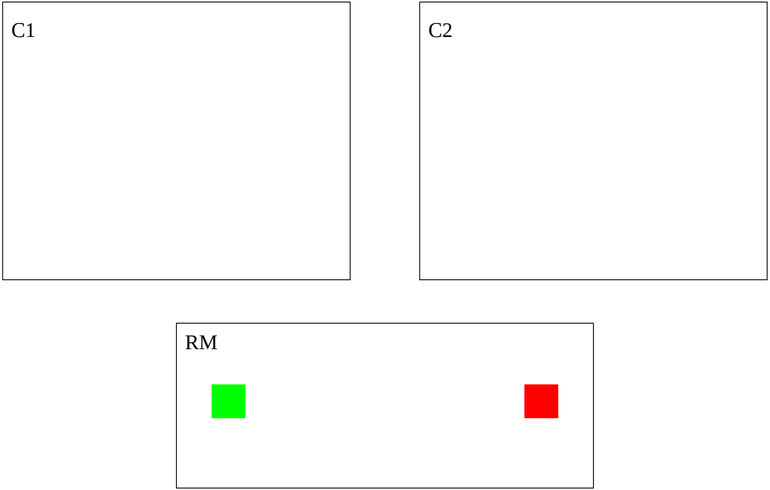
Example



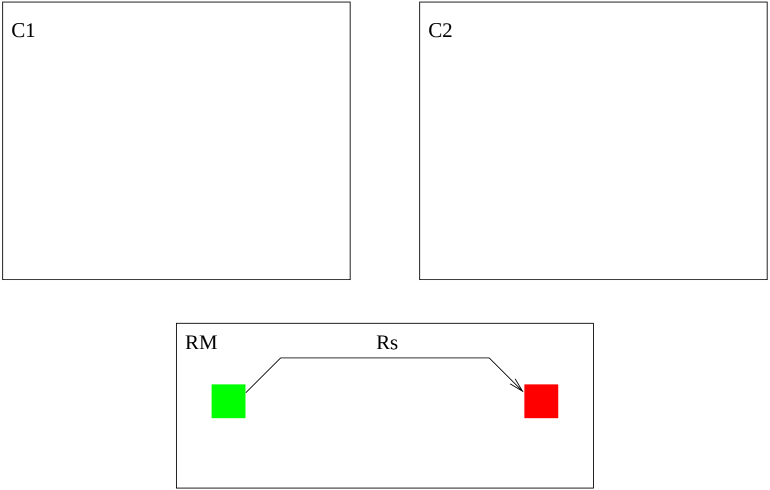
Example



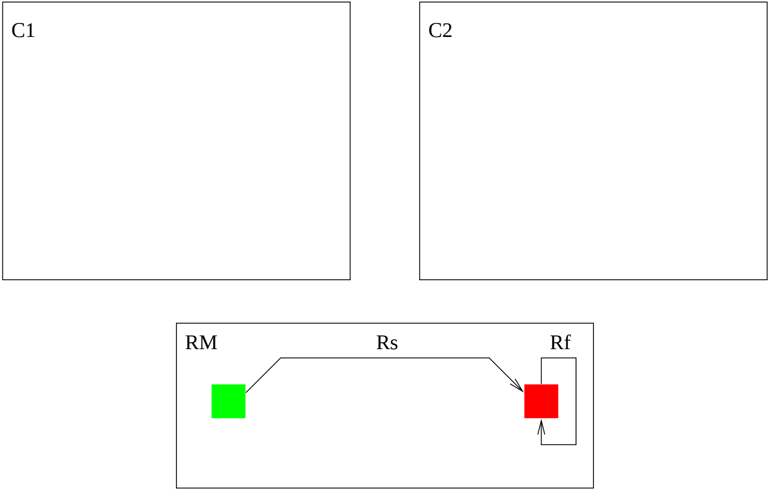
Example



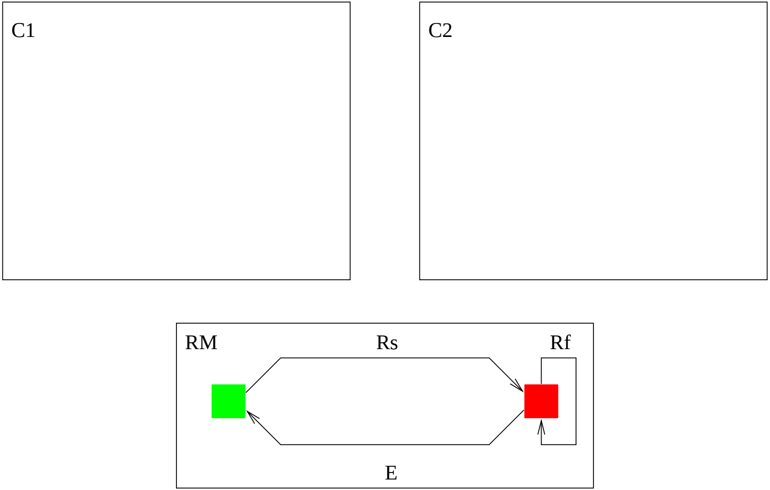
Example



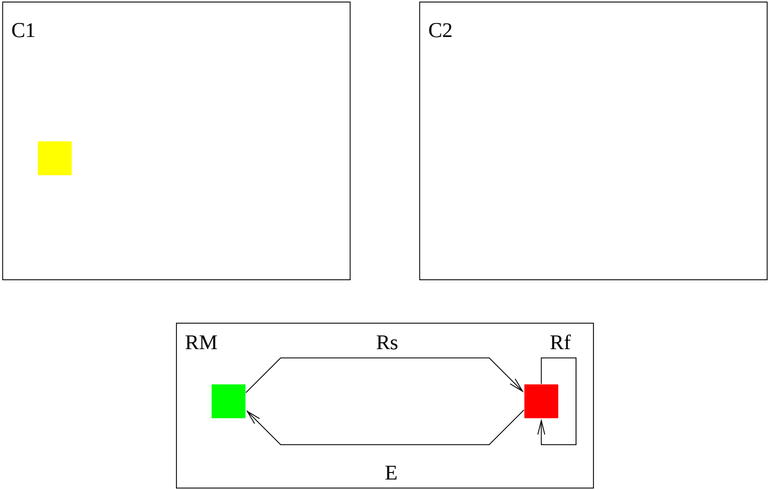
Example



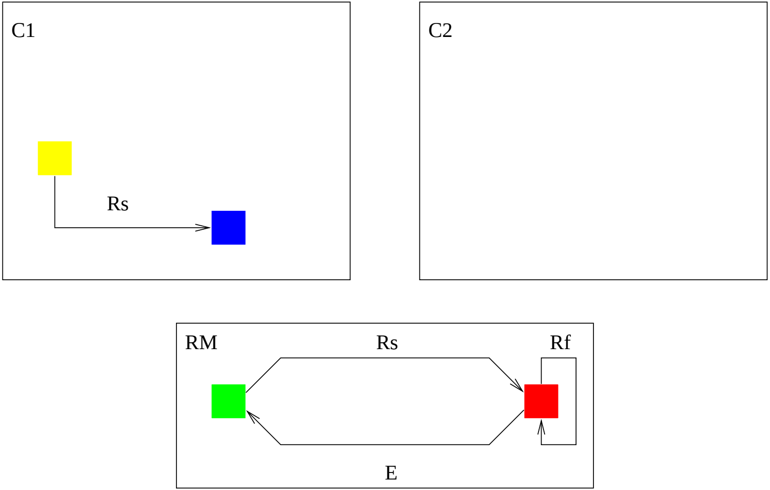
Example



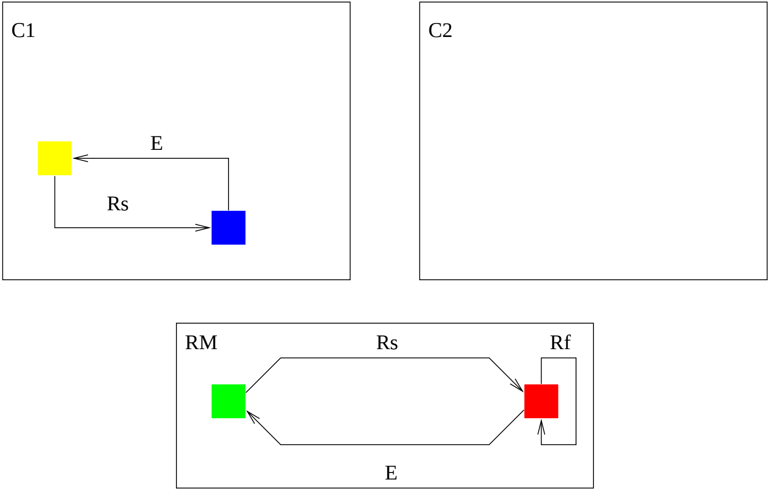
Example



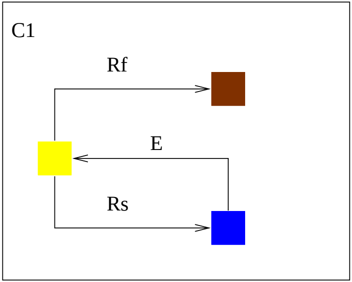
Example

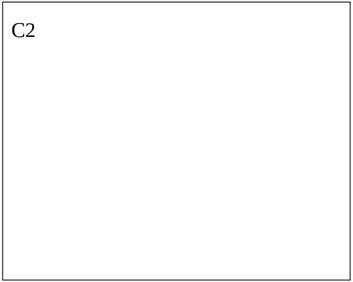


Example

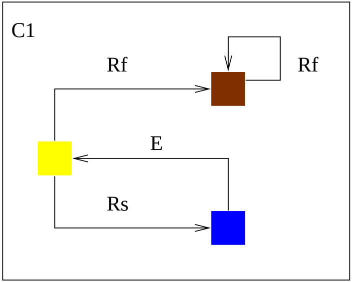


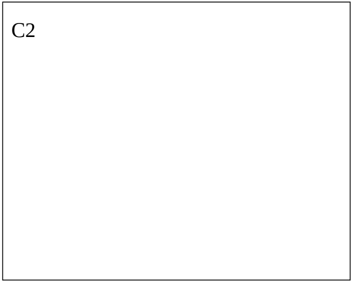
Example



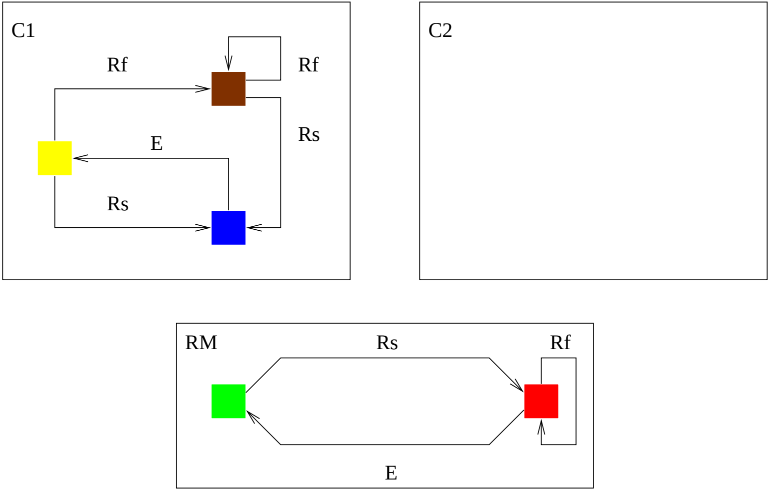


Example

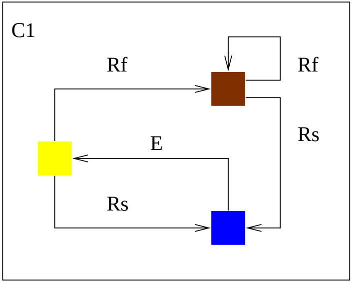


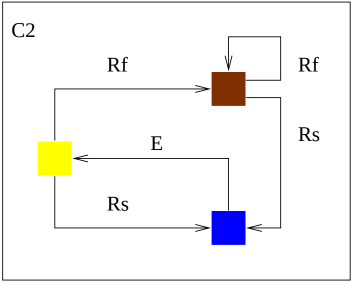


Example

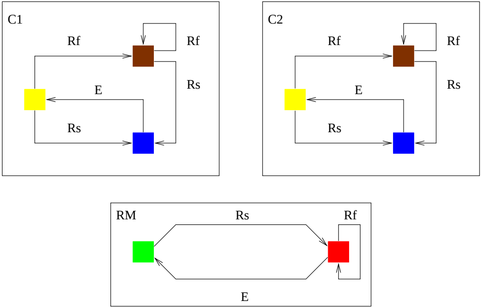


Example

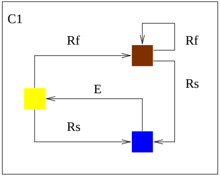


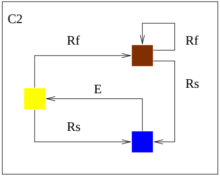


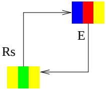
Example



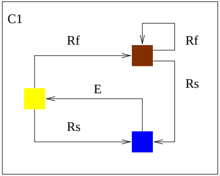
Example

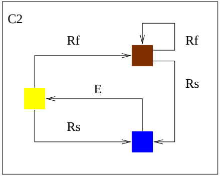


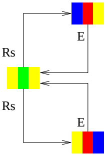




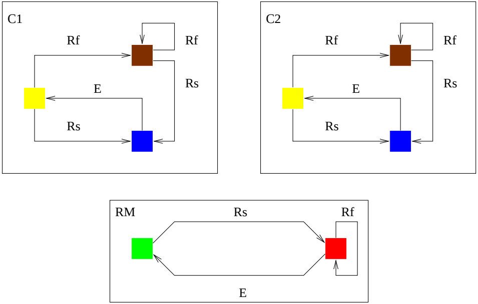
Example

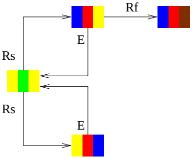




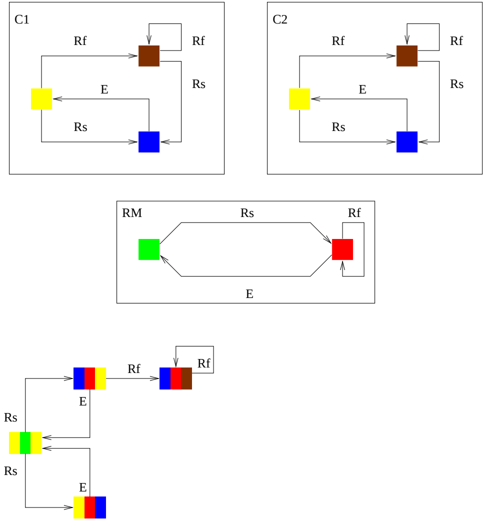


Example

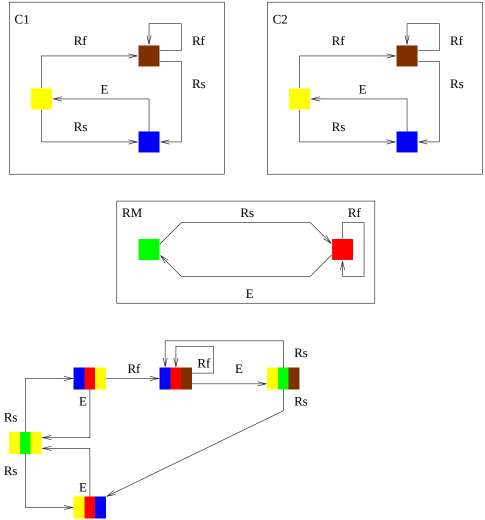




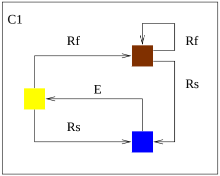
Example

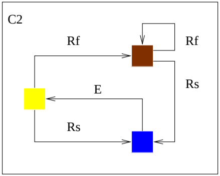


Example

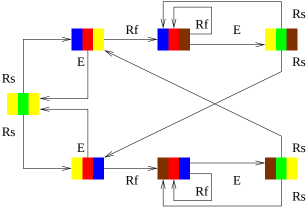


Example

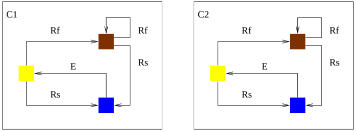

C1
Rf
Rf
Rs
E
Rs

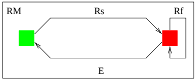

C2
Rf
Rf
Rs
E
Rs


RM
Rs
Rf
E

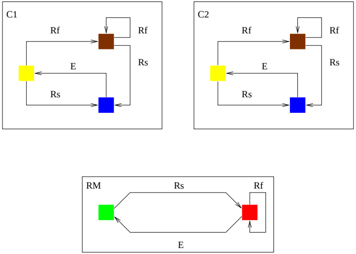

Rf
Rf
E
Rs
Rs
E
Rs
Rs
E
Rs
Rs
Rf
Rf
E
Rs

Example





Example

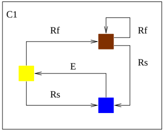


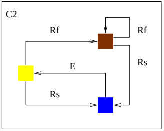
$\mathrm{Idle} \triangleq \mathrm{Rs}.\mathrm{Using} + \mathrm{Rf}.\mathrm{Retry}$

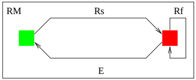
$\mathrm{Using} \triangleq \mathrm{E}.\mathrm{Idle}$

$\mathrm{Retry} \triangleq \mathrm{Rf}.\mathrm{Retry} + \mathrm{Rs}.\mathrm{Using}$

Example







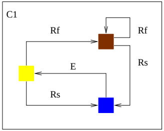
$\text{Idle} \triangleq \text{Rs}.\text{Using} + \text{Rf}.\text{Retry}$

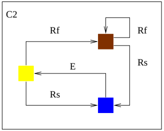
$\text{Using} \triangleq \text{E}.\text{Idle}$

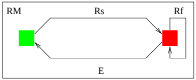
$\text{Retry} \triangleq \text{Rf}.\text{Retry} + \text{Rs}.\text{Using}$

$\text{C1} \triangleq \text{Idle}$

Example







$\text{Idle} \triangleq \text{Rs}.\text{Using} + \text{Rf}.\text{Retry}$

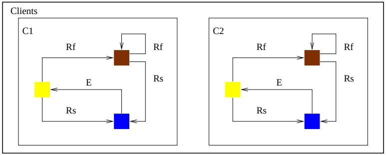
$\text{Using} \triangleq \text{E}.\text{Idle}$

$\text{Retry} \triangleq \text{Rf}.\text{Retry} + \text{Rs}.\text{Using}$

$\text{C1} \triangleq \text{Idle}$

$\text{C2} \triangleq \text{Idle}$

Example



$\text{Idle} \triangleq \text{Rs}.\text{Using} + \text{Rf}.\text{Retry}$

$\text{Using} \triangleq \text{E}.\text{Idle}$

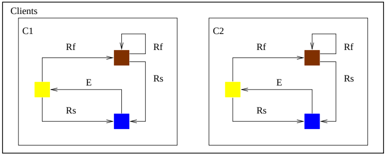
$\text{Retry} \triangleq \text{Rf}.\text{Retry} + \text{Rs}.\text{Using}$

$\text{C1} \triangleq \text{Idle}$

$\text{C2} \triangleq \text{Idle}$

$\text{Clients} \triangleq (\text{C1} \mid [\,] \mid \text{C2})$

Example



$\text{Idle} \triangleq \text{Rs}.\text{Using} + \text{Rf}.\text{Retry}$

$\text{Using} \triangleq \text{E}.\text{Idle}$

$\text{Retry} \triangleq \text{Rf}.\text{Retry} + \text{Rs}.\text{Using}$

$\text{C1} \triangleq \text{Idle}$

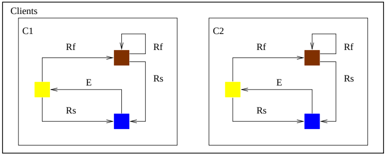
$\text{C2} \triangleq \text{Idle}$

$\text{Clients} \triangleq (\text{C1} \mid [\,] \mid \text{C2})$

$\text{Free} \triangleq \text{Rs}.\text{InUse}$

$\text{InUse} \triangleq \text{Rf}.\ \text{InUse} + \text{E}.\text{Free}$

Example



$\text{Idle} \triangleq \text{Rs}.\text{Using} + \text{Rf}.\text{Retry}$

$\text{Using} \triangleq \text{E}.\text{Idle}$

$\text{Retry} \triangleq \text{Rf}.\text{Retry} + \text{Rs}.\text{Using}$

$\text{C1} \triangleq \text{Idle}$

$\text{C2} \triangleq \text{Idle}$

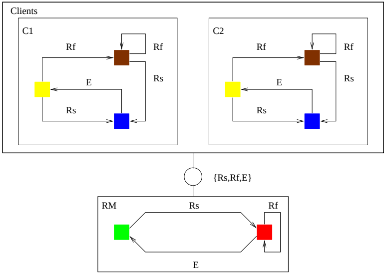
$\text{Clients} \triangleq (\text{C1} \mid [\,] \mid \text{C2})$

$\text{Free} \triangleq \text{Rs}.\text{InUse}$

$\text{InUse} \triangleq \text{Rf}.\ \text{InUse} + \text{E}.\text{Free}$

$\text{RM} \triangleq \text{Free}$

Example



$\text{Idle} \triangleq \text{Rs.Using} + \text{Rf.Retry}$

$\text{Using} \triangleq \text{E.Idle}$

$\text{Retry} \triangleq \text{Rf.Retry} + \text{Rs.Using}$

$\text{C1} \triangleq \text{Idle}$

$\text{C2} \triangleq \text{Idle}$

$\text{Clients} \triangleq (\text{C1} \mid [\,] \mid \text{C2})$

$\text{Free} \triangleq \text{Rs.InUse}$

$\text{InUse} \triangleq \text{Rf. InUse} + \text{E.Free}$

$\text{RM} \triangleq \text{Free}$

$\text{System} \triangleq \text{RM} \mid [\text{Rs,Rf,E}] \mid \text{Clients}$

A *process algebraic* approach to *system modelling*

A *process algebraic* approach to *system modelling*

1. Algebraic terms

A *process algebraic* approach to *system modelling*

1. Algebraic terms, defined via a Formal syntax

A *process algebraic* approach to *system modelling*

1. Algebraic terms, defined via a Formal syntax, often with **graphical tool support**;

A *process algebraic* approach to *system modelling*

1. Algebraic terms, defined via a Formal syntax, often with **graphical tool support**;
2. LTS

A *process algebraic* approach to *system modelling*

1. Algebraic terms, defined via a Formal syntax, often with **graphical tool support**;
2. LTS, the reference Mathematical Objects

A *process algebraic* approach to *system modelling*

1. Algebraic terms, defined via a Formal syntax, often with **graphical tool support**;
2. LTS, the reference Mathematical Objects, equipped with Behavioural Relations

A *process algebraic* approach to *system modelling*

1. Algebraic terms, defined via a Formal syntax, often with **graphical tool support**;
2. LTS, the reference Mathematical Objects, equipped with Behavioural Relations, i.e. Formal Preorders, Equivalences, and Congruences

A *process algebraic* approach to *system modelling*

1. Algebraic terms, defined via a Formal syntax, often with **graphical tool support**;
2. LTS, the reference Mathematical Objects, equipped with Behavioural Relations, i.e. Formal Preorders, Equivalences, and Congruences
3. A mapping of terms to LTS

A *process algebraic* approach to *system modelling*

1. Algebraic terms, defined via a Formal syntax, often with **graphical tool support**;
2. LTS, the reference Mathematical Objects, equipped with Behavioural Relations, i.e. Formal Preorders, Equivalences, and Congruences
3. A mapping of terms to LTS, the Formal Semantics definition

A *process algebraic* approach to *system modelling*

1. Algebraic terms, defined via a Formal syntax, often with **graphical tool support**;
2. LTS, the reference Mathematical Objects, equipped with Behavioural Relations, i.e. Formal Preorders, Equivalences, and Congruences
3. A mapping of terms to LTS, the Formal Semantics definition
4. Algebraic terms manipulation rules

A *process algebraic* approach to *system modelling*

1. Algebraic terms, defined via a Formal syntax, often with **graphical tool support**;
2. LTS, the reference Mathematical Objects, equipped with Behavioural Relations, i.e. Formal Preorders, Equivalences, and Congruences
3. A mapping of terms to LTS, the Formal Semantics definition
4. Algebraic terms manipulation rules, i.e. Axiomatizations of Equivalences

An example (Hoare CSP-like)

An example (Hoare CSP-like)

Formal Syntax definition:

$S ::= \textbf{nil} \mid \alpha.S \mid S + S \mid X \mid S|[\alpha_1, \ldots, \alpha_n]|S$

with $\alpha, \alpha_1 \ldots . \alpha_n \in A_t$ and constants defined via equations $X \triangleq S$.

An example (Hoare CSP-like)

Formal Syntax definition:

$S ::= \textbf{nil} \mid \alpha.S \mid S + S \mid X \mid S|[\alpha_1, \ldots, \alpha_n]|S$

with $\alpha, \alpha_1 \ldots . \alpha_n \in A_t$ and constants defined via equations $X \triangleq S$.

$\mathcal{P} \stackrel{\text{def}}{=}$ the set of terms generated by the above grammar.

An example (Hoare CSP-like)

Formal Semantics definition:

$$\alpha.S \xrightarrow{\alpha} S \qquad \frac{S_1 \xrightarrow{\alpha} S}{S_1 + S_2 \xrightarrow{\alpha} S} \qquad \frac{S_2 \xrightarrow{\alpha} S}{S_1 + S_2 \xrightarrow{\alpha} S} \qquad \frac{S \xrightarrow{\alpha} S', X \triangleq S}{X \xrightarrow{\alpha} S'}$$

$$\frac{S_1 \xrightarrow{\alpha} S_1', \alpha \notin L}{S_1|L|S_2 \xrightarrow{\alpha} S_1'|L|S_2} \qquad \frac{S_2 \xrightarrow{\alpha} S_2', \alpha \notin L}{S_1|L|S_2 \xrightarrow{\alpha} S_1|L|S_2'}$$

$$\frac{S_1 \xrightarrow{\alpha} S_1', S_2 \xrightarrow{\alpha} S_2', \alpha \in L}{S_1|L|S_2 \xrightarrow{\alpha} S_1'|L|S_2'}$$

An example (Hoare CSP-like)

Formal Semantics definition:

$$\alpha.S \xrightarrow{\alpha} S \quad \frac{S_1 \xrightarrow{\alpha} S}{S_1 + S_2 \xrightarrow{\alpha} S} \quad \frac{S_2 \xrightarrow{\alpha} S}{S_1 + S_2 \xrightarrow{\alpha} S} \quad \frac{S \xrightarrow{\alpha} S', X \triangleq S}{X \xrightarrow{\alpha} S'}$$

$$\frac{S_1 \xrightarrow{\alpha} S_1', \alpha \notin L}{S_1|L|S_2 \xrightarrow{\alpha} S_1'|L|S_2} \quad \frac{S_2 \xrightarrow{\alpha} S_2', \alpha \notin L}{S_1|L|S_2 \xrightarrow{\alpha} S_1|L|S_2'}$$

$$\frac{S_1 \xrightarrow{\alpha} S_1', S_2 \xrightarrow{\alpha} S_2', \alpha \in L}{S_1|L|S_2 \xrightarrow{\alpha} S_1'|L|S_2'}$$

An example (Hoare CSP-like)

Formal Semantics definition:

$$\alpha.S \xrightarrow{\alpha} S \qquad \frac{S_1 \xrightarrow{\alpha} S}{S_1 + S_2 \xrightarrow{\alpha} S} \qquad \frac{S_2 \xrightarrow{\alpha} S}{S_1 + S_2 \xrightarrow{\alpha} S} \qquad \frac{S \xrightarrow{\alpha} S', X \triangleq S}{X \xrightarrow{\alpha} S'}$$

$$\frac{S_1 \xrightarrow{\alpha} S_1', \alpha \notin L}{S_1|L|S_2 \xrightarrow{\alpha} S_1'|L|S_2} \qquad \frac{S_2 \xrightarrow{\alpha} S_2', \alpha \notin L}{S_1|L|S_2 \xrightarrow{\alpha} S_1|L|S_2'}$$

$$\frac{S_1 \xrightarrow{\alpha} S_1', S_2 \xrightarrow{\alpha} S_2', \alpha \in L}{S_1|L|S_2 \xrightarrow{\alpha} S_1'|L|S_2'}$$

$LTS_G \stackrel{\text{def}}{=} (\mathcal{P}, A_t, \longrightarrow)$ with:

1. $\mathcal{P}$: the set of states defined by the above grammar
2. A_t: the set of transition labels
3. $\longrightarrow \subseteq \mathcal{P} \times A_t \times \mathcal{P}$, the *least* relation satisfying the above rules.

An example (Hoare CSP-like)

Formal Semantics definition:

$$\alpha.S \xrightarrow{\alpha} S \qquad \frac{S_1 \xrightarrow{\alpha} S}{S_1 + S_2 \xrightarrow{\alpha} S} \qquad \frac{S_2 \xrightarrow{\alpha} S}{S_1 + S_2 \xrightarrow{\alpha} S} \qquad \frac{S \xrightarrow{\alpha} S', X \triangleq S}{X \xrightarrow{\alpha} S'}$$

$$\frac{S_1 \xrightarrow{\alpha} S_1', \alpha \notin L}{S_1|L|S_2 \xrightarrow{\alpha} S_1'|L|S_2} \qquad \frac{S_2 \xrightarrow{\alpha} S_2', \alpha \notin L}{S_1|L|S_2 \xrightarrow{\alpha} S_1|L|S_2'}$$

$$\frac{S_1 \xrightarrow{\alpha} S_1', S_2 \xrightarrow{\alpha} S_2', \alpha \in L}{S_1|L|S_2 \xrightarrow{\alpha} S_1'|L|S_2'}$$

$LTS_G \stackrel{\text{def}}{=} (\mathcal{P}, A_t, \longrightarrow)$ with:

1. $\mathcal{P}$: the set of states defined by the above grammar
2. A_t: the set of transition labels
3. $\longrightarrow \subseteq \mathcal{P} \times A_t \times \mathcal{P}$, the *least* relation satisfying the above rules.

For $S \in \mathcal{P}$, let $\mathcal{R}_S$ be the set of states in $\mathcal{P}$ which are reachable from S via $\longrightarrow$, $LTS_S \stackrel{\text{def}}{=} (\mathcal{R}_S, A_t, \longrightarrow \cap (\mathcal{R}_S \times A_t \times \mathcal{R}_S), S)$

Advantages

Advantages

- Compositionality

Advantages

- Compositionality
- Induction principles

Advantages

- Compositionality
- Induction principles
  - Natural induction principle

Advantages

- Compositionality
- Induction principles
  - Natural induction principle, but also

Advantages

- Compositionality
- Induction principles
  - Natural induction principle, but also
  - Structural induction

Advantages

- Compositionality
- Induction principles
    - Natural induction principle, but also
    - Structural induction
    - Computational induction

Advantages

- Compositionality
- Induction principles
  - Natural induction principle, but also
  - Structural induction
  - Computational induction

Advantages

- Compositionality
- Induction principles
  - Natural induction principle, but also
  - Structural induction
  - Computational induction
  - Derivation induction

Advantages

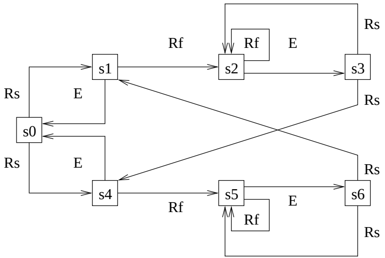
- Compositionality
- Induction principles
  - Natural induction principle, but also
  - Structural induction
  - Computational induction
  - Derivation induction
- Axiomatic reasoning

A
Temporal Logics
Approach to
Requirement Specification

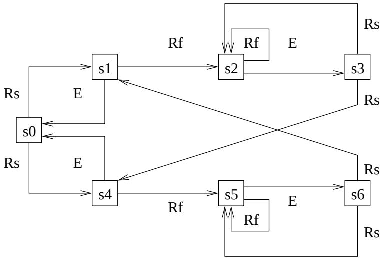
Computation Tree Structures

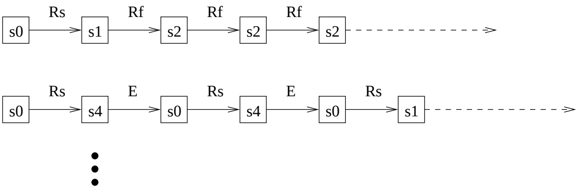
- Description of behaviour of a system by means of the set of its computations:
  - Computation: (possibly) infinite sequence of states which are reached, and transitions which take place during a single system run from the initial state;
  - Set of computations: represented as an (infinite) tree;
    - A *Computation Tree* associated to each system
    - A computation of the system: a *path* in the CT

Computation Tree Structures

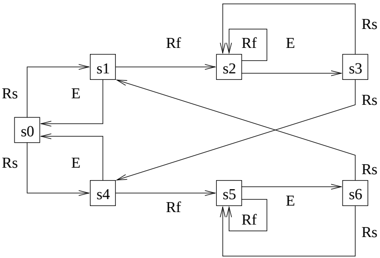


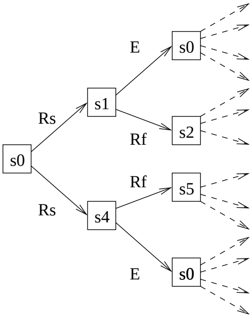
Computation Tree Structures

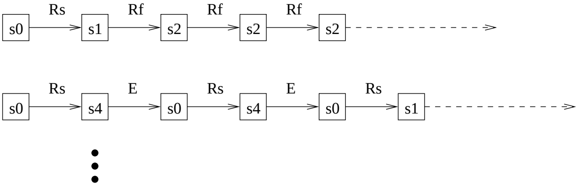




Computation Tree Structures

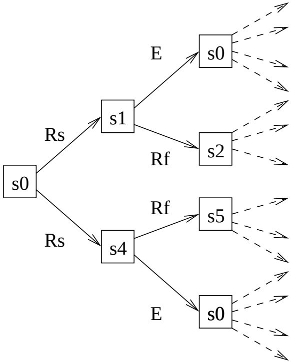




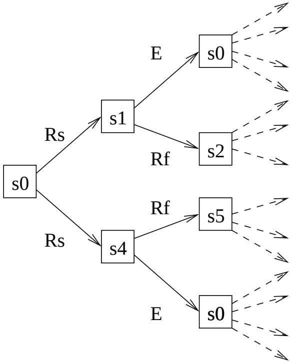


Computation Tree Structures

- **Graphical notation** (...)



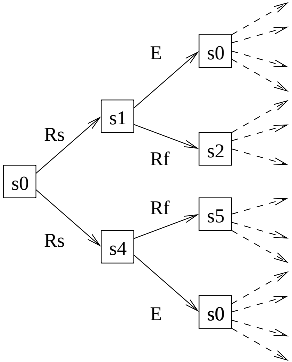
- **Graphical notation** (...)



- **Mathematical definition**

Computation Tree Structures

- **Graphical notation** (...)



- **Mathematical definition**

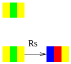
  e.g. in the framework of formal (ω-)languages or Computation Trees

Properties of paths

REMEMBER: paths represent system computations (traces, logs ...)!

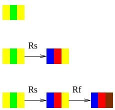
Properties of paths

REMEMBER: paths represent system computations (traces, logs ...)!



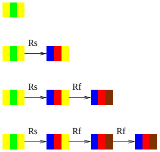
Properties of paths

REMEMBER: paths represent system computations (traces, logs ...)!



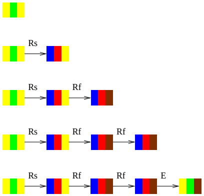
Properties of paths

REMEMBER: paths represent system computations (traces, logs ...)!



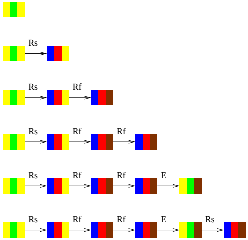
Properties of paths

REMEMBER: paths represent system computations (traces, logs ...)!



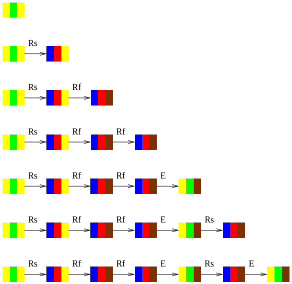
Properties of paths

REMEMBER: paths represent system computations (traces, logs ...)!



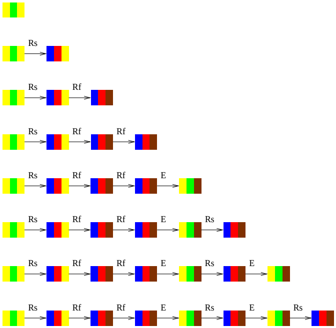
Properties of paths

REMEMBER: paths represent system computations (traces, logs ...)!



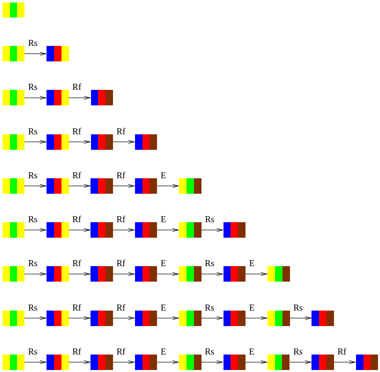
Properties of paths

REMEMBER: paths represent system computations (traces, logs ...)!



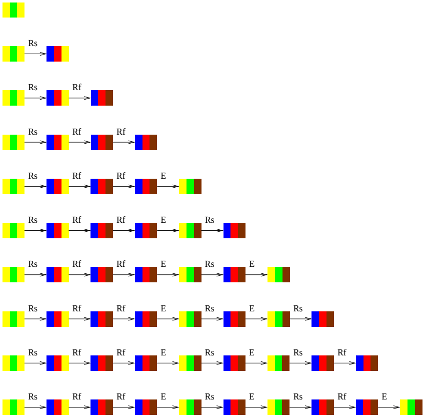
Properties of paths

REMEMBER: paths represent system computations (traces, logs ...)!



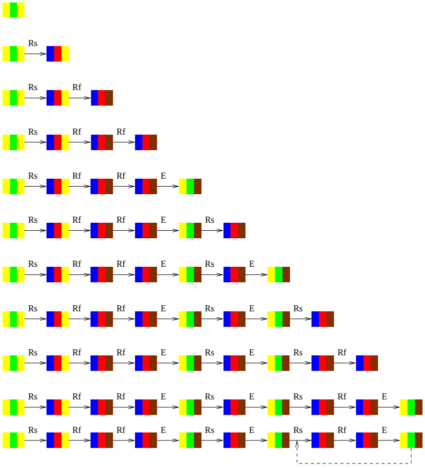
Properties of paths

REMEMBER: paths represent system computations (traces, logs ...)!



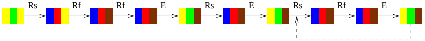
Properties of paths

REMEMBER: paths represent system computations (traces, logs ...)!



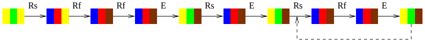
Properties of paths

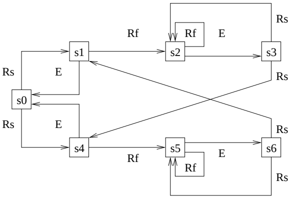
REMEMBER: paths represent system computations (traces, logs ...)!



Properties of paths

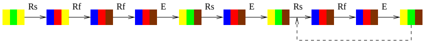
REMEMBER: paths represent system computations (traces, logs ...)!

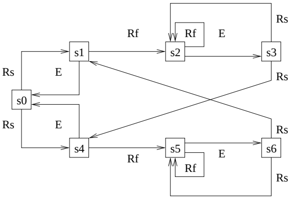




Properties of paths

REMEMBER: paths represent system computations (traces, logs ...)!





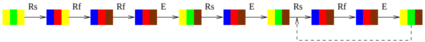
textually, the word

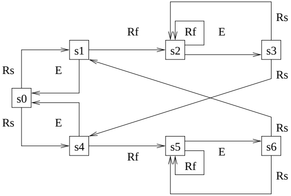
s0.Rs.s1.Rf.s2.Rf.s2.E.s3.Rs.s2.E.s3.Rs.s2.Rf.s2.E.s3.Rs.s2.Rf.s2.E.s3.Rs.s2

... forever Rf.s2.E.s3.Rs.s2

Properties of paths

REMEMBER: paths represent system computations (traces, logs ...)!





textually, the word

s0.Rs.s1.Rf.s2.Rf.s2.E.s3.Rs.s2.E.s3.Rs.s2.Rf.s2.E.s3.Rs.s2.Rf.s2.E.s3.Rs.s2

... forever Rf.s2.E.s3.Rs.s2

that is, in ω-languages notation

s0.Rs.s1.(Rf.s2)^2.E.s3.Rs.s2.E.s3(Rs.s2.Rf.s2.E.s3)$^\omega$

Properties of paths are expressed in logic

REMEMBER: paths represent system computations (traces, logs ...)!

Properties of paths are expressed in logic

REMEMBER: paths represent system computations (traces, logs ...)!

$s0.Rs.s1.(Rf.s2)^2.E.s3.Rs.s2.E.s3(Rs.s2.Rf.s2.E.s3)^\omega$

Properties of paths are expressed in logic

REMEMBER: paths represent system computations (traces, logs ...)!

$s0.Rs.s1.(Rf.s2)^2.E.s3.Rs.s2.E.s3(Rs.s2.Rf.s2.E.s3)^\omega$

- *eventually*, state s2 is reached *in the path*

Properties of paths are expressed in logic

REMEMBER: paths represent system computations (traces, logs ...)!

s0.Rs.s1.$(\text{Rf.s2})^2$.E.s3.Rs.s2.E.s3$(\text{Rs.s2.Rf.s2.E.s3})^\omega$

- *eventually*, state s2 is reached *in the path*

$$\boxed{\Diamond \underline{in}(s2)}$$

Properties of paths are expressed in logic

REMEMBER: paths represent system computations (traces, logs ...)!

s0.Rs.s1.(Rf.s2)^2.E.s3.Rs.s2.E.s3(Rs.s2.Rf.s2.E.s3)$^\omega$

- *eventually*, state s2 is reached *in the path*

$$\Diamond \underline{in}(s2)$$

Properties of paths are expressed in logic

REMEMBER: paths represent system computations (traces, logs ...)!

s0.Rs.s1.(Rf.s2)^2.E.s3.Rs.s2.E.s3(Rs.s2.Rf.s2.E.s3)$^\omega$

- *eventually*, state s2 is reached *in the path*

$$\Diamond \underline{in}(s2)$$

A set of atomic predicates $(tt, \underline{in}(s), x > 0, \ldots a, b, \ldots)$ *over/labeling states*

Properties of paths are expressed in logic

REMEMBER: paths represent system computations (traces, logs ...)!

s0.Rs.s1.(Rf.s2)^2.E.s3.Rs.s2.E.s3(Rs.s2.Rf.s2.E.s3)$^\omega$

- *eventually*, state s2 is reached *in the path*

$$\Diamond \underline{in}(s2)$$

A set of atomic predicates ($tt, \underline{in}(s), x > 0, \ldots a, b, \ldots$) over/labeling states

- state s3 is *never* reached *in the path*

Properties of paths are expressed in logic

REMEMBER: paths represent system computations (traces, logs ...)!

s0.Rs.s1.(Rf.s2)^2.E.s3.Rs.s2.E.s3(Rs.s2.Rf.s2.E.s3)$^\omega$

- *eventually*, state s2 is reached *in the path*

$$\Diamond \underline{in}(s2)$$

A set of atomic predicates $(tt, \underline{in}(s), x > 0, \ldots a, b, \ldots)$ *over/labeling states*

- state s3 is *never* reached *in the path*

$$\neg \Diamond \underline{in}(s3)$$

Properties of paths are expressed in logic

REMEMBER: paths represent system computations (traces, logs ...)!

s0.Rs.s1.$(\text{Rf.s2})^2$.E.s3.Rs.s2.E.s3$(\text{Rs.s2.Rf.s2.E.s3})^\omega$

- *eventually*, state s2 is reached *in the path*

$$\Diamond \underline{in}(s2)$$

A set of atomic predicates $(tt, \underline{in}(s), x > 0, \ldots a, b, \ldots)$ *over/labeling states*

- state s3 is *never* reached *in the path*

$$\neg\Diamond \underline{in}(s3)$$

Properties of paths are expressed in logic

REMEMBER: paths represent system computations (traces, logs ...)!

s0.Rs.s1.(Rf.s2)^2.E.s3.Rs.s2.E.s3(Rs.s2.Rf.s2.E.s3)$^\omega$

- *eventually*, state s2 is reached *in the path*

  $\Diamond \underline{in}(s2)$

A set of atomic predicates $(tt, \underline{in}(s), x > 0, \ldots a, b, \ldots)$ *over/labeling states*

- state s3 is *never* reached *in the path*

  $\neg\Diamond \underline{in}(s3)$

- the system is *always* in a state different from s4 and s5 *in the path*

Properties of paths are expressed in logic

REMEMBER: paths represent system computations (traces, logs ...)!

$s0.Rs.s1.(Rf.s2)^2.E.s3.Rs.s2.E.s3(Rs.s2.Rf.s2.E.s3)^\omega$

- *eventually*, state s2 is reached *in the path*

$$\Diamond \underline{in}(s2)$$

A set of atomic predicates $(tt, \underline{in}(s), x > 0, \ldots a, b, \ldots)$ *over/labeling states*

- state s3 is *never* reached *in the path*

$$\neg\Diamond \underline{in}(s3)$$

- the system is *always* in a state different from s4 and s5 *in the path*

$$\Box\neg(\underline{in}(s4) \vee \underline{in}(s5))$$

Properties of paths are expressed in logic

REMEMBER: paths represent system computations (traces, logs ...)!

s0.Rs.s1.(Rf.s2)^2.E.s3.Rs.s2.E.s3(Rs.s2.Rf.s2.E.s3)$^\omega$

- *eventually*, state s2 is reached *in the path*

  $\Diamond \underline{in}(s2)$

A set of atomic predicates ($tt, \underline{in}(s), x > 0, \ldots a, b, \ldots$) *over/labeling states*

- state s3 is *never* reached *in the path*

  $\neg \Diamond \underline{in}(s3)$

- the system is *always* in a state different from s4 and s5 *in the path*

  $\Box \neg (\underline{in}(s4) \vee \underline{in}(s5))$

Properties of paths are expressed in logic

REMEMBER: paths represent system computations (traces, logs ...)!

$s0.Rs.s1.(Rf.s2)^2.E.s3.Rs.s2.E.s3(Rs.s2.Rf.s2.E.s3)^\omega$

- *eventually*, state s2 is reached *in the path*

$$\Diamond \underline{in}(s2)$$

A set of atomic predicates $(tt, \underline{in}(s), x > 0, \ldots a, b, \ldots)$ *over/labeling states*

- state s3 is *never* reached *in the path*

$$\neg\Diamond \underline{in}(s3)$$

- the system is *always* in a state different from s4 and s5 *in the path*

$$\Box\neg(\underline{in}(s4) \vee \underline{in}(s5))$$

$$\Box\Phi \equiv \neg\Diamond\neg\Phi$$

Properties of paths are expressed in logic

REMEMBER: paths represent system computations (traces, logs ...)!

$s0.Rs.s1.(Rf.s2)^2.E.s3.Rs.s2.E.s3(Rs.s2.Rf.s2.E.s3)^\omega$

Properties of paths are expressed in logic

REMEMBER: paths represent system computations (traces, logs ...)!

- s0.Rs.s1.(Rf.s2)^2.E.s3.Rs.s2.E.s3(Rs.s2.Rf.s2.E.s3)$^\omega$
- s0, s1, and s2, and only them, are visited *until* s3 is reached, *in the path*

Properties of paths are expressed in logic

REMEMBER: paths represent system computations (traces, logs ...)!

- $s0.Rs.s1.(Rf.s2)^2.E.s3.Rs.s2.E.s3(Rs.s2.Rf.s2.E.s3)^\omega$
- s0, s1, and s2, and only them, are visited *until* s3 is reached, *in the path*
    - s3 is eventually reached *in the path*

Properties of paths are expressed in logic

REMEMBER: paths represent system computations (traces, logs ...)!

- $s0.Rs.s1.(Rf.s2)^2.E.s3.Rs.s2.E.s3(Rs.s2.Rf.s2.E.s3)^\omega$
- s0, s1, and s2, and only them, are visited *until* s3 is reached, *in the path*
  - s3 is eventually reached *in the path*
  - *in the path*, the only states which the system can be in, before reaching s3, are s0, s1, and s2.

Properties of paths are expressed in logic

REMEMBER: paths represent system computations (traces, logs ...)!

- s0.Rs.s1.(Rf.s2)^2.E.s3.Rs.s2.E.s3(Rs.s2.Rf.s2.E.s3)$^\omega$
- s0, s1, and s2, and only them, are visited *until* s3 is reached, *in the path*
  - s3 is eventually reached *in the path*
  - *in the path*, the only states which the system can be in, before reaching s3, are s0, s1, and s2.

$$(\underline{in}(s0) \vee \underline{in}(s1) \vee \underline{in}(s2)) \;\; \mathcal{U} \; \underline{in}(s3)$$

Properties of paths are expressed in logic

REMEMBER: paths represent system computations (traces, logs ...)!

- s0.Rs.s1.(Rf.s2)^2.E.s3.Rs.s2.E.s3(Rs.s2.Rf.s2.E.s3)$^\omega$
- s0, s1, and s2, and only them, are visited *until* s3 is reached, *in the path*
    - s3 is eventually reached *in the path*
    - *in the path*, the only states which the system can be in, before reaching s3, are s0, s1, and s2.

$$(\underline{in}(s0) \vee \underline{in}(s1) \vee \underline{in}(s2)) \;\; \mathcal{U} \; \underline{in}(s3)$$

Properties of paths are expressed in logic

REMEMBER: paths represent system computations (traces, logs ...)!

- $s0.Rs.s1.(Rf.s2)^2.E.s3.Rs.s2.E.s3(Rs.s2.Rf.s2.E.s3)^\omega$
- s0, s1, and s2, and only them, are visited *until* s3 is reached, *in the path*
  - s3 is eventually reached *in the path*
  - *in the path*, the only states which the system can be in, before reaching s3, are s0, s1, and s2.

$$(\underline{in}(s0) \vee \underline{in}(s1) \vee \underline{in}(s2)) \;\; \mathcal{U} \; \underline{in}(s3)$$

$$\Diamond \Phi \equiv \mathsf{tt} \;\; \mathcal{U} \; \Phi$$

Properties of paths are expressed in logic

REMEMBER: paths represent system computations (traces, logs ...)!

- s0.Rs.s1.(Rf.s2)^2.E.s3.Rs.s2.E.s3(Rs.s2.Rf.s2.E.s3)$^\omega$
- s0, s1, and s2, and only them, are visited *until* s3 is reached, *in the path*
  - s3 is eventually reached *in the path*
  - *in the path*, the only states which the system can be in, before reaching s3, are s0, s1, and s2.

$$(\underline{in}(s0) \vee \underline{in}(s1) \vee \underline{in}(s2)) \;\; \mathcal{U} \;\; \underline{in}(s3)$$

$$\Diamond \Phi \equiv \mathrm{tt} \;\; \mathcal{U} \;\; \Phi$$

- the *next state* reached is s1

Properties of paths are expressed in logic

REMEMBER: paths represent system computations (traces, logs ...)!

- s0.Rs.s1.(Rf.s2)^2.E.s3.Rs.s2.E.s3(Rs.s2.Rf.s2.E.s3)$^\omega$
- s0, s1, and s2, and only them, are visited *until* s3 is reached, *in the path*
  - s3 is eventually reached *in the path*
  - *in the path*, the only states which the system can be in, before reaching s3, are s0, s1, and s2.

$$(\underline{in}(s0) \vee \underline{in}(s1) \vee \underline{in}(s2)) \;\; \mathcal{U} \; \underline{in}(s3)$$

$$\Diamond\Phi \equiv \mathrm{tt} \;\; \mathcal{U} \; \Phi$$

- the *next state* reached is s1

$$\mathbf{X} \; \underline{in}(s1)$$

Properties of paths are expressed in logic

REMEMBER: paths represent system computations (traces, logs ...)!

- s0.Rs.s1.(Rf.s2)^2.E.s3.Rs.s2.E.s3(Rs.s2.Rf.s2.E.s3)$^\omega$
- s0, s1, and s2, and only them, are visited *until* s3 is reached, *in the path*
  - s3 is eventually reached *in the path*
  - *in the path*, the only states which the system can be in, before reaching s3, are s0, s1, and s2.

$$(\underline{in}(s0) \vee \underline{in}(s1) \vee \underline{in}(s2)) \;\; \mathcal{U} \; \underline{in}(s3)$$

$$\Diamond \Phi \equiv \mathrm{tt} \;\; \mathcal{U} \; \Phi$$

- the *next state* reached is s1

$$\mathbf{X} \; \underline{in}(s1)$$

Properties of paths: *Path Formulae*

REMEMBER: paths represent system computations (traces, logs ...)!

Formal Syntax definition of *Path Formulae*

$$\begin{array}{lcll} \varphi & ::= & \Phi \;\; \mathcal{U} \; \Phi & (\textit{strong until}) \\ & | & \mathbf{X} \; \Phi & (\textit{next state}) \end{array}$$

Properties of paths: *Path Formulae*

REMEMBER: paths represent system computations (traces, logs ...)!

Formal Syntax definition of *Path Formulae*

$$
\begin{array}{lcll}
\varphi & ::= & \Phi \ \mathcal{U} \ \Phi & (\textit{strong until}) \\
 & \mid & \mathbf{X}\ \Phi & (\textit{next state}) \\
 & \mid & \Diamond \Phi & (\textit{eventually: } \Diamond \Phi \equiv \textit{tt} \ \mathcal{U} \ \Phi) \\
 & \mid & \Box \Phi & (\textit{always: } \Box \Phi \equiv \neg \Diamond \neg \Phi)
\end{array}
$$

Properties of paths: *Path Formulae*

REMEMBER: paths represent system computations (traces, logs ...)!

Formal Syntax definition of *Path Formulae*

$$
\begin{array}{lcll}
\varphi & ::= & \Phi \ \ \mathcal{U} \ \Phi & (\textit{strong until}) \\
 & \mid & \mathbf{X}\ \Phi & (\textit{next state}) \\
 & \mid & \Diamond\Phi & (\textit{eventually: } \Diamond\Phi \equiv tt \ \ \mathcal{U}\ \Phi) \\
 & \mid & \Box\Phi & (\textit{always: } \Box\Phi \equiv \neg\Diamond\neg\Phi)
\end{array}
$$

Φ is a *State Formula*

Formal Syntax definition of *Path Formulae*

$$
\begin{array}{lclll}
\varphi & ::= & \Phi \ \ \mathcal{U} \ \Phi & & (\textit{strong until}) \\
& | & \mathbf{X}\ \Phi & & (\textit{next state}) \\
& | & \Diamond\Phi & & (\textit{eventually: } \Diamond\Phi \equiv tt \ \ \mathcal{U} \ \Phi) \\
& | & \Box\Phi & & (\textit{always: } \Box\Phi \equiv \neg\Diamond\neg\Phi)
\end{array}
$$

Φ is a *State Formula*, including Atomic propositions
$(\mathrm{tt}, \underline{in}(s), x > 0, \ldots a, b, \ldots)$

Properties of paths: *Path Formulae*

REMEMBER: paths represent system computations (traces, logs ...)!

Formal Syntax definition of *Path Formulae*

$$
\begin{array}{llll}
\varphi & ::= & \Phi \;\; \mathcal{U} \; \Phi & (\textit{strong until}) \\
 & \mid & \mathbf{X}\, \Phi & (\textit{next state}) \\
 & \mid & \Diamond\Phi & (\textit{eventually: } \Diamond\Phi \equiv tt \;\; \mathcal{U} \; \Phi) \\
 & \mid & \Box\Phi & (\textit{always: } \Box\Phi \equiv \neg\Diamond\neg\Phi)
\end{array}
$$

Φ is a *State Formula*, including Atomic propositions $(\mathrm{tt}, \underline{in}(s), x > 0, \ldots a, b, \ldots)$

- **Basic components**
- **Ways for composing them**

Properties of paths: *Path Formulae*

REMEMBER: paths represent system computations (traces, logs ...)!

Formal Syntax definition of *Path Formulae*

$$
\begin{array}{lcll}
\varphi & ::= & \Phi \;\; \mathcal{U} \; \Phi & (\textit{strong until}) \\
 & | & \mathbf{X} \; \Phi & (\textit{next state}) \\
 & | & \Diamond \Phi & (\textit{eventually: } \Diamond \Phi \equiv tt \;\; \mathcal{U} \; \Phi) \\
 & | & \Box \Phi & (\textit{always: } \Box \Phi \equiv \neg \Diamond \neg \Phi)
\end{array}
$$

Φ is a *State Formula*, including Atomic propositions $(\mathrm{tt}, \underline{in}(s), x > 0, \ldots a, b, \ldots)$

- **Basic components**
  Atomic (state) Formulae
- **Ways for composing them**

Properties of paths: *Path Formulae*

REMEMBER: paths represent system computations (traces, logs ...)!

Formal Syntax definition of *Path Formulae*

$$\begin{array}{lll} \varphi & ::= & \Phi \;\; \mathcal{U} \;\Phi & (\textit{strong until}) \\ & \mid & \mathbf{X}\; \Phi & (\textit{next state}) \\ & \mid & \Diamond\Phi & (\textit{eventually: } \Diamond\Phi \equiv tt \;\; \mathcal{U} \;\Phi) \\ & \mid & \Box\Phi & (\textit{always: } \Box\Phi \equiv \neg\Diamond\neg\Phi) \end{array}$$

Φ is a *State Formula*, including Atomic propositions $(\mathrm{tt}, \underline{in}(s), x > 0, \ldots a, b, \ldots)$

- **Basic components**
  Atomic (state) Formulae
- **Ways for composing them**
  $\wedge,\ \vee,\ \neg,\ \ \mathcal{U}\ ,\ \mathbf{X}\ ,\ \Diamond,\ \Box$

From logic formulae to computations via Formal Semantics

$$(\underline{in}(s0) \vee \underline{in}(s1) \vee \underline{in}(s2)) \;\; \mathcal{U} \;\; \underline{in}(s3)$$

From logic formulae to computations via Formal Semantics

Formal Syntax definition
(Grammar)

$(\underline{in}(s0) \vee \underline{in}(s1) \vee \underline{in}(s2)) \;\; \mathcal{U} \;\; \underline{in}(s3)$

From logic formulae to computations via Formal Semantics

Formal Syntax definition (Grammar)

$(\underline{in}(s0) \vee \underline{in}(s1) \vee \underline{in}(s2)) \;\; \mathcal{U} \;\; \underline{in}(s3)$

s0.Rs.s1.(Rf.s2)^2.E.s3.Rs.s1.E.s0(Rs.s1.Rf.s2.E.s3)$^\omega$

From logic formulae to computations via Formal Semantics

Formal Syntax definition
(Grammar)

$(\underline{in}(s0) \vee \underline{in}(s1) \vee \underline{in}(s2)) \; \mathcal{U} \; \underline{in}(s3)$

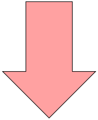
Mathematical Objects
(Paths & CTS)

$s0.Rs.s1.(Rf.s2)^2.E.s3.Rs.s1.E.s0(Rs.s1.Rf.s2.E.s3)^\omega$

From logic formulae to computations via Formal Semantics

Formal Syntax definition
(Grammar)

$$(\underline{in}(s0) \vee \underline{in}(s1) \vee \underline{in}(s2)) \;\; \mathcal{U} \;\; \underline{in}(s3)$$



Mathematical Objects
(Paths & CTS)

$s0.Rs.s1.(Rf.s2)^2.E.s3.Rs.s1.E.s0(Rs.s1.Rf.s2.E.s3)^\omega$

From logic formulae to computations via Formal Semantics

Formal Syntax definition
(Grammar)

$(\underline{in}(s0) \vee \underline{in}(s1) \vee \underline{in}(s2)) \;\, \mathcal{U} \;\, \underline{in}(s3)$

Formal Semantics definition
(Satisfaction Relation)

Mathematical Objects
(Paths & CTS)

$s0.Rs.s1.(Rf.s2)^2.E.s3.Rs.s1.E.s0(Rs.s1.Rf.s2.E.s3)^\omega$

Properties of states are expressed in logic

REMEMBER: paths start from states and are collected in a CT

- *In all paths* starting from s0, s0 will eventually be reached (again)

Properties of states are expressed in logic

REMEMBER: paths start from states and are collected in a CT

- *In all paths* starting from s0, s0 will eventually be reached (again)

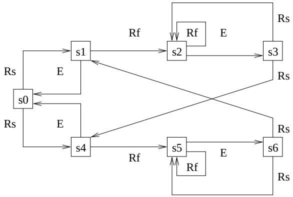
$$\boxed{\forall \mathbf{X}\ \forall \Diamond \underline{in}(so)}$$

Properties of states are expressed in logic

REMEMBER: paths start from states and are collected in a CT

- *In all paths* starting from s0, s0 will eventually be reached (again)

$$\forall \mathbf{X}\ \forall \Diamond \underline{in}(so)$$

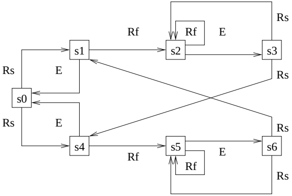


Properties of states are expressed in logic

REMEMBER: paths start from states and are collected in a CT

- *There is a path* starting from s0, s0 will eventually be reached (again)

$$\exists \mathbf{X}\ \exists\Diamond \underline{in}(so)$$

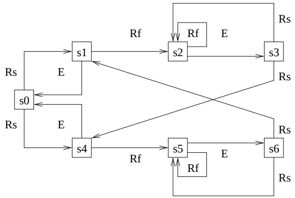


Properties of states are expressed in logic

REMEMBER: paths start from states and are collected in a CT

- *In all paths* starting from s0, s0 will eventually be reached (again)

$$\forall \mathbf{X}\ \forall\Diamond \underline{in}(so)$$

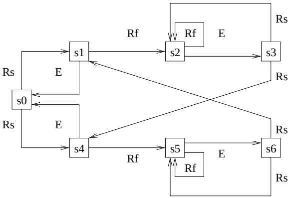


Properties of states are expressed in logic

REMEMBER: paths start from states and are collected in a CT

- *In all paths* starting from s0, s0 will eventually be reached (again)

$$\forall \mathbf{X}\ \forall \Diamond \underline{in}(so)$$



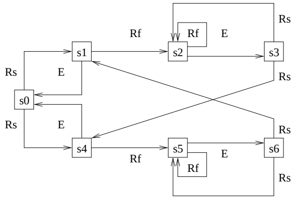
- *In all paths* starting from s0, one of s0, s3, or s6 will eventually be reached

Properties of states are expressed in logic

REMEMBER: paths start from states and are collected in a CT

- *In all paths* starting from s0, s0 will eventually be reached (again)

$$\forall \mathbf{X}\ \forall\Diamond \underline{in}(so)$$



- *In all paths* starting from s0, one of s0, s3, or s6 will eventually be reached

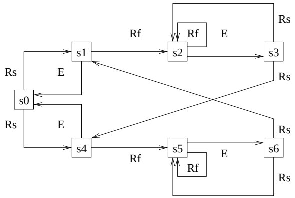
$$\forall \mathbf{X}\ \forall\Diamond(\underline{in}(s0) \vee \underline{in}(s3) \vee \underline{in}(s6))$$

Properties of states are expressed in logic

REMEMBER: paths start from states and are collected in a CT

- *In all paths* starting from s0, s0 will eventually be reached (again)

$$\forall \mathbf{X}\ \forall\Diamond \underline{in}(so)$$



- *In all paths* starting from s0, one of s0, s3, or s6 will eventually be reached

$$\forall \mathbf{X}\ \forall\Diamond(\underline{in}(s0) \vee \underline{in}(s3) \vee \underline{in}(s6))$$

Properties of states are expressed in logic

REMEMBER: paths start from states and are collected in a CT

- s0 will eventually be reached from all states in all computations

Properties of states are expressed in logic

REMEMBER: paths start from states and are collected in a CT

- s0 will eventually be reached from all states in all computations

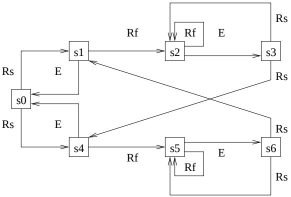
$$\boxed{\forall\Box\forall\Diamond \underline{in}(s0)}$$

Properties of states are expressed in logic

REMEMBER: paths start from states and are collected in a CT

- s0 will eventually be reached from all states in all computations

$$\forall\Box\forall\Diamond \underline{in}(s0)$$

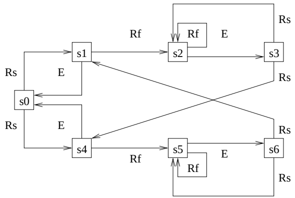


Properties of states are expressed in logic

REMEMBER: paths start from states and are collected in a CT

- s0 will eventually be reached from all states in all computations

$$\forall\Box\forall\Diamond \underline{in}(s0)$$

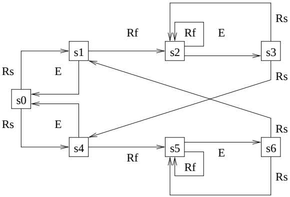


Properties of states are expressed in logic

REMEMBER: paths start from states and are collected in a CT

- s0 will eventually be reached from all states in all computations

$$\forall\Box\forall\Diamond \underline{in}(s0)$$

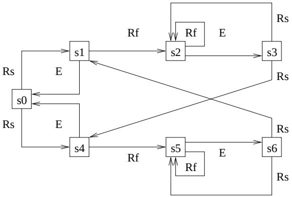


Properties of states are expressed in logic

REMEMBER: paths start from states and are collected in a CT

- s0 will eventually be reached from all states in all computations

$$\forall\Box\forall\Diamond \underline{in}(s0)$$

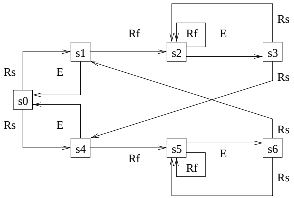


Properties of states are expressed in logic

REMEMBER: paths start from states and are collected in a CT

- s0 will eventually be reached from all states in all computations

$$\forall\Box\forall\Diamond \underline{in}(s0)$$



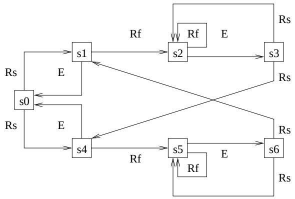
- Whenever s2 is reached s4 will be reached

Properties of states are expressed in logic

REMEMBER: paths start from states and are collected in a CT

- s0 will eventually be reached from all states in all computations

$$\forall\Box\forall\Diamond \underline{in}(s0)$$



- Whenever s2 is reached s4 will be reached

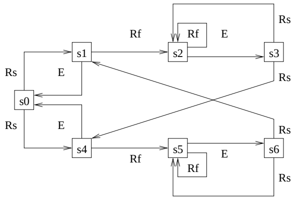
$$\forall\Box(\underline{in}(s2) \Rightarrow \forall\Diamond \underline{in}(s4))$$

Properties of states are expressed in logic

REMEMBER: paths start from states and are collected in a CT

- s0 will eventually be reached from all states in all computations

$$\forall\Box\forall\Diamond \underline{in}(s0)$$



- Whenever s2 is reached s4 will be reached

$$\forall\Box(\underline{in}(s2) \Rightarrow \forall\Diamond \underline{in}(s4))$$

Properties of states are expressed in logic

REMEMBER: paths start from states and are collected in a CT

- There is a path in which eventually s2 is reached and then s4 is never reached

Properties of states are expressed in logic

REMEMBER: paths start from states and are collected in a CT

- There is a path in which eventually s2 is reached and then s4 is never reached

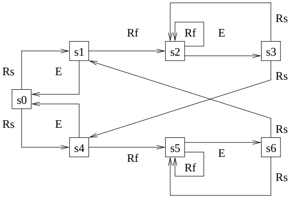
$$\boxed{\exists\Diamond(\underline{in}(s2) \wedge \forall\Box\neg\underline{in}(s4))}$$

Properties of states are expressed in logic

REMEMBER: paths start from states and are collected in a CT

- There is a path in which eventually s2 is reached and then s4 is never reached

$$\exists\Diamond(\underline{in}(s2) \wedge \forall\Box\neg\underline{in}(s4))$$

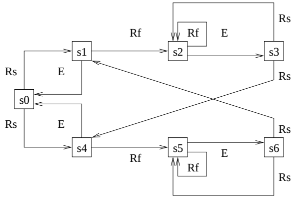


Properties of states are expressed in logic

REMEMBER: paths start from states and are collected in a CT

- There is a path in which eventually s2 is reached and then s4 is never reached

$$\exists\Diamond(\underline{in}(s2) \wedge \forall\Box\neg\underline{in}(s4))$$



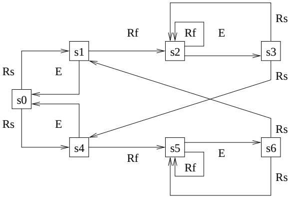
- It cannot be that s1 is reached without having first reached s0 or s6

Properties of states are expressed in logic

REMEMBER: paths start from states and are collected in a CT

- There is a path in which eventually s2 is reached and then s4 is never reached

$$\exists\Diamond(\underline{in}(s2) \land \forall\Box\neg\underline{in}(s4))$$



- It cannot be that s1 is reached without having first reached s0 or s6

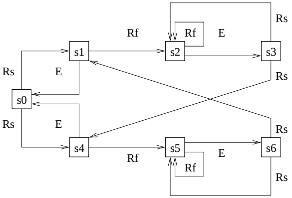
$$\neg\exists(\neg(\underline{in}(s0) \lor \underline{in}(s6)) \;\; \mathcal{U} \; \underline{in}(s1))$$

Properties of states are expressed in logic

REMEMBER: paths start from states and are collected in a CT

- There is a path in which eventually s2 is reached and then s4 is never reached

$$\exists\Diamond(\underline{in}(s2) \wedge \forall\Box\neg\underline{in}(s4))$$



- It cannot be that s1 is reached without having first reached s0 or s6

$$\neg\exists(\neg(\underline{in}(s0) \vee \underline{in}(s6)) \;\; \mathcal{U} \;\; \underline{in}(s1))$$

A *Temporal Logics* approach to *Requirements specification*

A *Temporal Logics* approach to *Requirements specification*

1. Logic formulae

A *Temporal Logics* approach to *Requirements specification*

1. Logic formulae, defined via a Formal syntax

A *Temporal Logics* approach to *Requirements specification*

1. Logic formulae, defined via a Formal syntax, often with (graphical) **tool support**;

A *Temporal Logics* approach to *Requirements specification*

1. Logic formulae, defined via a Formal syntax, often with (graphical) **tool support**;
2. CT

A *Temporal Logics* approach to *Requirements specification*

1. Logic formulae, defined via a Formal syntax, often with (graphical) **tool support**;
2. CT, the reference Mathematical Objects

A *Temporal Logics* approach to *Requirements specification*

1. Logic formulae, defined via a Formal syntax, often with (graphical) **tool support**;
2. CT, the reference Mathematical Objects, equipped with proper operations and tightly related to LTS;

A *Temporal Logics* approach to *Requirements specification*

1. Logic formulae, defined via a Formal syntax, often with (graphical) **tool support**;
2. CT, the reference Mathematical Objects, equipped with proper operations and tightly related to LTS;
3. A relation between formulae and CT (states of / paths over LTS)

A *Temporal Logics* approach to *Requirements specification*

1. Logic formulae, defined via a Formal syntax, often with (graphical) **tool support**;
2. CT, the reference Mathematical Objects, equipped with proper operations and tightly related to LTS;
3. A relation between formulae and CT (states of / paths over LTS), the Satisfaction Relation

A *Temporal Logics* approach to *Requirements specification*

1. Logic formulae, defined via a Formal syntax, often with (graphical) **tool support**;
2. CT, the reference Mathematical Objects, equipped with proper operations and tightly related to LTS;
3. A relation between formulae and CT (states of / paths over LTS), the Satisfaction Relation, i.e. the Formal Semantics definition;

A *Temporal Logics* approach to *Requirements specification*

1. Logic formulae, defined via a Formal syntax, often with (graphical) **tool support**;
2. CT, the reference Mathematical Objects, equipped with proper operations and tightly related to LTS;
3. A relation between formulae and CT (states of / paths over LTS), the Satisfaction Relation, i.e. the Formal Semantics definition;
4. Logic formulae manipulation rules

   ;

A *Temporal Logics* approach to *Requirements specification*

1. Logic formulae, defined via a Formal syntax, often with (graphical) **tool support**;
2. CT, the reference Mathematical Objects, equipped with proper operations and tightly related to LTS;
3. A relation between formulae and CT (states of / paths over LTS), the Satisfaction Relation, i.e. the Formal Semantics definition;
4. Logic formulae manipulation rules, i.e. Axiomatizations and deduction systems.;

A *Temporal Logics* approach to *Requirements specification*

1. Logic formulae, defined via a Formal syntax, often with (graphical) **tool support**;
2. CT, the reference Mathematical Objects, equipped with proper operations and tightly related to LTS;
3. A relation between formulae and CT (states of / paths over LTS), the Satisfaction Relation, i.e. the Formal Semantics definition;
4. Logic formulae manipulation rules, i.e. Axiomatizations and deduction systems.;
5. Automatic verification: i.e. model-checking

An example (Clarke et Al. CTL-like)

An example (Clarke et Al. CTL-like)

- Formal Syntax definition:

$$\mathcal{A} ::= \mathrm{tt} \mid a \mid \ldots$$
$$\Phi ::= \mathcal{A} \mid \neg\Phi \mid \Phi \wedge \Phi \mid \Phi \vee \Phi \mid \forall\varphi \mid \exists\varphi$$
$$\varphi ::= \mathbf{X}\ \Phi \mid \Phi\ \ \mathcal{U}\ \Phi \mid \Diamond\Phi \mid \Box\Phi$$

An example (Clarke et Al. CTL-like)

- Formal Syntax definition:

$$\mathcal{A} ::= \text{tt} \mid a \mid \ldots$$
$$\Phi ::= \mathcal{A} \mid \neg\Phi \mid \Phi \wedge \Phi \mid \Phi \vee \Phi \mid \forall\varphi \mid \exists\varphi$$
$$\varphi ::= \mathbf{X}\ \Phi \mid \Phi\ \ \mathcal{U}\ \Phi \mid \Diamond\Phi \mid \Box\Phi$$

- Formal Semantics definition:

An example (Clarke et Al. CTL-like)

- Formal Syntax definition:

$$\mathcal{A} ::= \mathrm{tt} \mid a \mid \ldots$$
$$\Phi ::= \mathcal{A} \mid \neg\Phi \mid \Phi \wedge \Phi \mid \Phi \vee \Phi \mid \forall\varphi \mid \exists\varphi$$
$$\varphi ::= \mathbf{X}\ \Phi \mid \Phi\ \ \mathcal{U}\ \Phi \mid \Diamond\Phi \mid \Box\Phi$$

- Formal Semantics definition:

$$s \models \mathrm{tt}$$

An example (Clarke et Al. CTL-like)

- Formal Syntax definition:

$$\mathcal{A} ::= \mathrm{tt} \;\Big|\; a \;\Big|\; \ldots$$
$$\Phi ::= \mathcal{A} \;\Big|\; \neg\Phi \;\Big|\; \Phi \wedge \Phi \;\Big|\; \Phi \vee \Phi \;\Big|\; \forall\varphi \;\Big|\; \exists\varphi$$
$$\varphi ::= \mathbf{X}\ \Phi \;\Big|\; \Phi\ \ \mathcal{U}\ \Phi \;\Big|\; \Diamond\Phi \;\Big|\; \Box\Phi$$

- Formal Semantics definition:

$s \models \mathrm{tt}$ $\quad s \models a$ iff $a \in L(s)$

An example (Clarke et Al. CTL-like)

- Formal Syntax definition:

$$\mathcal{A} ::= \text{tt} \;\Big|\; a \;\Big|\; \ldots$$
$$\Phi ::= \mathcal{A} \;\Big|\; \neg\Phi \;\Big|\; \Phi \wedge \Phi \;\Big|\; \Phi \vee \Phi \;\Big|\; \forall\varphi \;\Big|\; \exists\varphi$$
$$\varphi ::= \mathbf{X}\,\Phi \;\Big|\; \Phi\;\;\mathcal{U}\;\Phi \;\Big|\; \Diamond\Phi \;\Big|\; \Box\Phi$$

- Formal Semantics definition:

$s \models \text{tt}$ $\quad s \models a$ iff $a \in L(s)$ $\quad s \models \neg\Phi$ iff not $s \models \Phi$

An example (Clarke et Al. CTL-like)

- Formal Syntax definition:

$$\mathcal{A} ::= \mathrm{tt} \mid a \mid \ldots$$
$$\Phi ::= \mathcal{A} \mid \neg\Phi \mid \Phi \wedge \Phi \mid \Phi \vee \Phi \mid \forall\varphi \mid \exists\varphi$$
$$\varphi ::= \mathbf{X}\ \Phi \mid \Phi\ \ \mathcal{U}\ \Phi \mid \Diamond\Phi \mid \Box\Phi$$

- Formal Semantics definition:

$s \models \mathrm{tt}$ $\quad s \models a$ iff $a \in L(s)$ $\quad s \models \neg\Phi$ iff not $s \models \Phi$

$s \models \Phi_1 \wedge \Phi_2$ iff $s \models \Phi_1$ and $s \models \Phi_2$

An example (Clarke et Al. CTL-like)

- Formal Syntax definition:

$$\mathcal{A} ::= \mathsf{tt} \mid a \mid \ldots$$
$$\Phi ::= \mathcal{A} \mid \neg\Phi \mid \Phi \wedge \Phi \mid \Phi \vee \Phi \mid \forall\varphi \mid \exists\varphi$$
$$\varphi ::= \mathbf{X}\ \Phi \mid \Phi\ \ \mathcal{U}\ \Phi \mid \Diamond\Phi \mid \Box\Phi$$

- Formal Semantics definition:

$$s \models \mathsf{tt} \qquad s \models a \text{ iff } a \in L(s) \qquad s \models \neg\Phi \text{ iff not } s \models \Phi$$
$$s \models \Phi_1 \wedge \Phi_2 \text{ iff } s \models \Phi_1 \text{ and } s \models \Phi_2$$
$$s \models \Phi_1 \vee \Phi_2 \text{ iff } s \models \Phi_1 \text{ or } s \models \Phi_2$$

An example (Clarke et Al. CTL-like)

- Formal Syntax definition:

$$\mathcal{A} ::= \mathrm{tt} \;\big|\; a \;\big|\; \ldots$$
$$\Phi ::= \mathcal{A} \;\big|\; \neg\Phi \;\big|\; \Phi \wedge \Phi \;\big|\; \Phi \vee \Phi \;\big|\; \forall\varphi \;\big|\; \exists\varphi$$
$$\varphi ::= \mathbf{X}\; \Phi \;\big|\; \Phi \;\; \mathcal{U}\; \Phi \;\big|\; \Diamond\Phi \;\big|\; \Box\Phi$$

- Formal Semantics definition:

$s \models \mathrm{tt} \quad s \models a$ iff $a \in L(s) \quad s \models \neg\Phi$ iff not $s \models \Phi$
$s \models \Phi_1 \wedge \Phi_2$ iff $s \models \Phi_1$ and $s \models \Phi_2$
$s \models \Phi_1 \vee \Phi_2$ iff $s \models \Phi_1$ or $s \models \Phi_2$
$s \models \forall\varphi$ iff for all paths γ with $\gamma[0] = s$: $\gamma \models \varphi$

An example (Clarke et Al. CTL-like)

- Formal Syntax definition:

$$\mathcal{A} ::= \text{tt} \mid a \mid \ldots$$
$$\Phi ::= \mathcal{A} \mid \neg\Phi \mid \Phi \wedge \Phi \mid \Phi \vee \Phi \mid \forall\varphi \mid \exists\varphi$$
$$\varphi ::= \mathbf{X}\,\Phi \mid \Phi\ \mathcal{U}\ \Phi \mid \Diamond\Phi \mid \Box\Phi$$

- Formal Semantics definition:

$s \models \text{tt}$ $\quad s \models a$ iff $a \in L(s)$ $\quad s \models \neg\Phi$ iff not $s \models \Phi$
$s \models \Phi_1 \wedge \Phi_2$ iff $s \models \Phi_1$ and $s \models \Phi_2$
$s \models \Phi_1 \vee \Phi_2$ iff $s \models \Phi_1$ or $s \models \Phi_2$
$s \models \forall\varphi$ iff for all paths γ with $\gamma[0] = s$: $\gamma \models \varphi$
$s \models \exists\varphi$ iff there exists a path γ with $\gamma[0] = s$: $\gamma \models \varphi$

An example (Clarke et Al. CTL-like)

- Formal Syntax definition:

$$\mathcal{A} ::= \mathrm{tt} \;\Big|\; a \;\Big|\; \ldots$$
$$\Phi ::= \mathcal{A} \;\Big|\; \neg\Phi \;\Big|\; \Phi \wedge \Phi \;\Big|\; \Phi \vee \Phi \;\Big|\; \forall\varphi \;\Big|\; \exists\varphi$$
$$\varphi ::= \mathbf{X}\,\Phi \;\Big|\; \Phi\ \mathcal{U}\ \Phi \;\Big|\; \Diamond\Phi \;\Big|\; \Box\Phi$$

- Formal Semantics definition:

$s \models \mathrm{tt}$ $\quad s \models a$ iff $a \in L(s)$ $\quad s \models \neg\Phi$ iff not $s \models \Phi$

$s \models \Phi_1 \wedge \Phi_2$ iff $s \models \Phi_1$ and $s \models \Phi_2$

$s \models \Phi_1 \vee \Phi_2$ iff $s \models \Phi_1$ or $s \models \Phi_2$

$s \models \forall\varphi$ iff for all paths γ with $\gamma[0] = s$: $\gamma \models \varphi$

$s \models \exists\varphi$ iff there exists a path γ with $\gamma[0] = s$: $\gamma \models \varphi$

$\gamma \models \mathbf{X}\,\Phi$ iff $\gamma[1] \models \Phi$

An example (Clarke et Al. CTL-like)

- Formal Syntax definition:

$$\mathcal{A} ::= \mathrm{tt} \mid a \mid \ldots$$
$$\Phi ::= \mathcal{A} \mid \neg\Phi \mid \Phi \wedge \Phi \mid \Phi \vee \Phi \mid \forall\varphi \mid \exists\varphi$$
$$\varphi ::= \mathbf{X}\ \Phi \mid \Phi\ \ \mathcal{U}\ \Phi \mid \Diamond\Phi \mid \Box\Phi$$

- Formal Semantics definition:

$s \models \mathrm{tt}$ $\quad s \models a$ iff $a \in L(s)$ $\quad s \models \neg\Phi$ iff not $s \models \Phi$

$s \models \Phi_1 \wedge \Phi_2$ iff $s \models \Phi_1$ and $s \models \Phi_2$

$s \models \Phi_1 \vee \Phi_2$ iff $s \models \Phi_1$ or $s \models \Phi_2$

$s \models \forall\varphi$ iff for all paths γ with $\gamma[0] = s$: $\gamma \models \varphi$

$s \models \exists\varphi$ iff there exists a path γ with $\gamma[0] = s$: $\gamma \models \varphi$

$\gamma \models \mathbf{X}\ \Phi$ iff $\gamma[1] \models \Phi$

$\gamma \models \Phi_1\ \ \mathcal{U}\ \Phi_2$ iff there exists $j \geq 0$ s.t. $\gamma[j] \models \Phi_2$ and $\gamma[i] \models \Phi_1$, for all $0 \leq i < j$

An example (Clarke et Al. CTL-like)

- Formal Syntax definition:

$$\mathcal{A} ::= \mathrm{tt} \mid a \mid \ldots$$
$$\Phi ::= \mathcal{A} \mid \neg\Phi \mid \Phi \wedge \Phi \mid \Phi \vee \Phi \mid \forall\varphi \mid \exists\varphi$$
$$\varphi ::= \mathbf{X}\ \Phi \mid \Phi\ \ \mathcal{U}\ \Phi \mid \Diamond\Phi \mid \Box\Phi$$

- Formal Semantics definition:

$$s \models \mathrm{tt} \quad s \models a \text{ iff } a \in L(s) \quad s \models \neg\Phi \text{ iff not } s \models \Phi$$
$$s \models \Phi_1 \wedge \Phi_2 \text{ iff } s \models \Phi_1 \text{ and } s \models \Phi_2$$
$$s \models \Phi_1 \vee \Phi_2 \text{ iff } s \models \Phi_1 \text{ or } s \models \Phi_2$$
$$s \models \forall\varphi \text{ iff for all paths } \gamma \text{ with } \gamma[0] = s\text{: } \gamma \models \varphi$$
$$s \models \exists\varphi \text{ iff there exists a path } \gamma \text{ with } \gamma[0] = s\text{: } \gamma \models \varphi$$
$$\gamma \models \mathbf{X}\ \Phi \text{ iff } \gamma[1] \models \Phi$$
$$\gamma \models \Phi_1\ \ \mathcal{U}\ \Phi_2 \text{ iff there exists } j \geq 0 \text{ s.t. } \gamma[j] \models \Phi_2 \text{ and } \gamma[i] \models \Phi_1, \text{ for all } 0 \leq i < j$$

Formulae manipulation:

An example (Clarke et Al. CTL-like)

- Formal Syntax definition:

$$\mathcal{A} ::= \text{tt} \mid a \mid \ldots$$
$$\Phi ::= \mathcal{A} \mid \neg\Phi \mid \Phi \wedge \Phi \mid \Phi \vee \Phi \mid \forall\varphi \mid \exists\varphi$$
$$\varphi ::= \mathbf{X}\ \Phi \mid \Phi\ \ \mathcal{U}\ \Phi \mid \Diamond\Phi \mid \Box\Phi$$

- Formal Semantics definition:

$$s \models \text{tt} \quad s \models a \text{ iff } a \in L(s) \quad s \models \neg\Phi \text{ iff not } s \models \Phi$$
$$s \models \Phi_1 \wedge \Phi_2 \text{ iff } s \models \Phi_1 \text{ and } s \models \Phi_2$$
$$s \models \Phi_1 \vee \Phi_2 \text{ iff } s \models \Phi_1 \text{ or } s \models \Phi_2$$
$$s \models \forall\varphi \text{ iff for all paths } \gamma \text{ with } \gamma[0] = s\text{: } \gamma \models \varphi$$
$$s \models \exists\varphi \text{ iff there exists a path } \gamma \text{ with } \gamma[0] = s\text{: } \gamma \models \varphi$$
$$\gamma \models \mathbf{X}\ \Phi \text{ iff } \gamma[1] \models \Phi$$
$$\gamma \models \Phi_1\ \ \mathcal{U}\ \Phi_2 \text{ iff there exists } j \geq 0 \text{ s.t. } \gamma[j] \models \Phi_2 \text{ and } \gamma[i] \models \Phi_1, \text{ for all } 0 \leq i < j$$

Formulae manipulation: $\forall\Box\Phi \Rightarrow \exists\Box\Phi$

Formal Methods Model Checkers

Computer support: mechanization of formal manipulation

Automatic Model Checking

Automatic Model Checking

Requirements Specification
(technical specification)

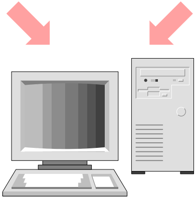
Automatic Model Checking

Requirements Specification
(technical specification)

Design Specification
(system model)

Requirements Specification
(technical specification)

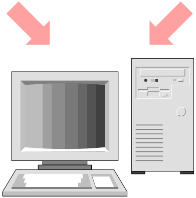
Design Specification
(system model)



Automatic Model Checking

Requirements Specification
(technical specification)

Design Specification
(system model)

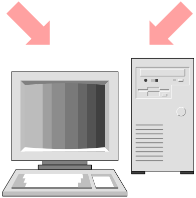


OK!

Automatic Model Checking

Requirements Specification
(technical specification)

Design Specification
(system model)

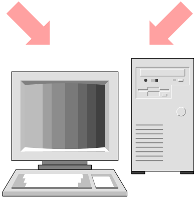


ERROR!!

Automatic Model Checking

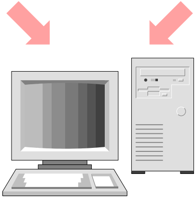
Requirements Specification
(technical specification)

Design Specification
(system model)



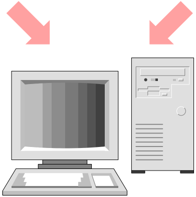
ERROR!!
+
counter example!!

Automatic Model Checking

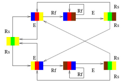


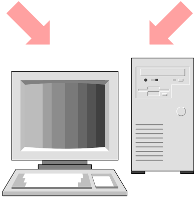
Automatic Model Checking

System $\triangleq$
RM | [Rs,Rf,E] | Clients



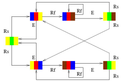
Automatic Model Checking

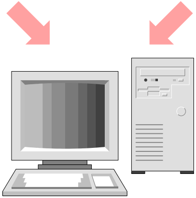




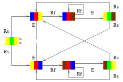
Automatic Model Checking

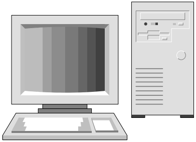
$\forall\Box\forall\Diamond \underline{in}(s0)$





$\forall\Box\forall\Diamond\, \underline{in}(s0)$

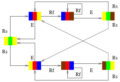


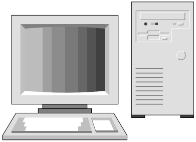


ERROR!!

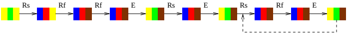
Automatic Model Checking

$\forall\Box\forall\Diamond \underline{in}(s0)$





ERROR!!



MC Algorithm Specification

Given: $\mathcal{K} = (S, A_s, L, \rightarrow)$ and Φ, returns: $Sat(\Phi) \stackrel{\text{def}}{=} \{s \in S \mid s \models \Phi\}$.

MC Algorithm Specification

Given: $\mathcal{K} = (S, A_s, L, \rightarrow)$ and Φ, returns: $Sat(\Phi) \stackrel{\text{def}}{=} \{s \in S \mid s \models \Phi\}$.

$$Sat(\text{tt}) \stackrel{\text{def}}{=} S$$

MC Algorithm Specification

Given: $\mathcal{K} = (S, A_s, L, \rightarrow)$ and Φ, returns: $Sat(\Phi) \stackrel{\text{def}}{=} \{s \in S \mid s \models \Phi\}$.

$$Sat(\text{tt}) \stackrel{\text{def}}{=} S$$

$$Sat(a) \stackrel{\text{def}}{=} \{s \in S \mid a \in L(s)\}$$

MC Algorithm Specification

Given: $\mathcal{K} = (S, A_s, L, \rightarrow)$ and Φ, returns: $Sat(\Phi) \stackrel{\text{def}}{=} \{s \in S \mid s \models \Phi\}$.

$$
\begin{array}{lcl}
Sat(\mathrm{tt}) & \stackrel{\text{def}}{=} & S \\
Sat(a) & \stackrel{\text{def}}{=} & \{s \in S \mid a \in L(s)\} \\
Sat(\neg\Phi) & \stackrel{\text{def}}{=} & S \setminus Sat(\Phi) \\
Sat(\Phi_1 \wedge \Phi_2) & \stackrel{\text{def}}{=} & Sat(\Phi_1) \cap Sat(\Phi_2) \\
Sat(\Phi_1 \vee \Phi_2) & \stackrel{\text{def}}{=} & Sat(\Phi_1) \cup Sat(\Phi_2)
\end{array}
$$

MC Algorithm Specification

Given: $\mathcal{K} = (S, A_s, L, \rightarrow)$ and Φ, returns: $Sat(\Phi) \stackrel{\text{def}}{=} \{s \in S \mid s \models \Phi\}$.

$$
\begin{array}{lcl}
Sat(\mathrm{tt}) & \stackrel{\text{def}}{=} & S \\
Sat(a) & \stackrel{\text{def}}{=} & \{s \in S \mid a \in L(s)\} \\
Sat(\neg\Phi) & \stackrel{\text{def}}{=} & S \setminus Sat(\Phi) \\
Sat(\Phi_1 \wedge \Phi_2) & \stackrel{\text{def}}{=} & Sat(\Phi_1) \cap Sat(\Phi_2) \\
Sat(\Phi_1 \vee \Phi_2) & \stackrel{\text{def}}{=} & Sat(\Phi_1) \cup Sat(\Phi_2) \\
Sat(\forall \mathbf{X}\, \Phi) & \stackrel{\text{def}}{=} & \{s \in S \mid \forall s' \text{ s.t. } s \rightarrow s' : s' \in Sat(\Phi)\} \\
Sat(\exists \mathbf{X}\, \Phi) & \stackrel{\text{def}}{=} & \{s \in S \mid \exists s' \text{ s.t. } s \rightarrow s' \text{ and } s' \in Sat(\Phi)\}
\end{array}
$$

MC Algorithm Specification

Given: $\mathcal{K} = (S, A_s, L, \rightarrow)$ and Φ, returns: $Sat(\Phi) \stackrel{\text{def}}{=} \{s \in S \mid s \models \Phi\}$.

$$
\begin{array}{lcl}
Sat(\mathrm{tt}) & \stackrel{\text{def}}{=} & S \\
Sat(a) & \stackrel{\text{def}}{=} & \{s \in S \mid a \in L(s)\} \\
Sat(\neg\Phi) & \stackrel{\text{def}}{=} & S \setminus Sat(\Phi) \\
Sat(\Phi_1 \wedge \Phi_2) & \stackrel{\text{def}}{=} & Sat(\Phi_1) \cap Sat(\Phi_2) \\
Sat(\Phi_1 \vee \Phi_2) & \stackrel{\text{def}}{=} & Sat(\Phi_1) \cup Sat(\Phi_2) \\
Sat(\forall \mathbf{X}\, \Phi) & \stackrel{\text{def}}{=} & \{s \in S \mid \forall s' \text{ s.t. } s \rightarrow s' : s' \in Sat(\Phi)\} \\
Sat(\exists \mathbf{X}\, \Phi) & \stackrel{\text{def}}{=} & \{s \in S \mid \exists s' \text{ s.t. } s \rightarrow s' \text{ and } s' \in Sat(\Phi)\} \\
Sat(\forall \Phi_1 \;\mathcal{U}\; \Phi_2) & \stackrel{\text{def}}{=} & Sat(\Phi_2) \cup \{s \in Sat(\Phi_1) \mid \forall s' \text{ s.t. } s \rightarrow s' : \\
 & & \qquad s' \in Sat(\forall \Phi_1 \;\mathcal{U}\; \Phi_2)\} \\
Sat(\exists \Phi_1 \;\mathcal{U}\; \Phi_2) & \stackrel{\text{def}}{=} & Sat(\Phi_2) \cup \{s \in Sat(\Phi_1) \mid \exists s' \text{ s.t. } s \rightarrow s' \text{ and} \\
 & & \qquad s' \in Sat(\exists \Phi_1 \;\mathcal{U}\; \Phi_2)\}
\end{array}
$$

Success stories

Classical FM have been successfully used for modeling and analyzing functional aspects of complex systems, for example:

Success stories

Classical FM have been successfully used for modeling and analyzing functional aspects of complex systems, for example:

Model-checking models of *trillions* of states (or more ... 10^{30})

Success stories

Classical FM have been successfully used for modeling and analyzing functional aspects of complex systems, for example:

Model-checking models of *trillions* of states (or more ... 10^{100})

Success stories

Classical FM have been successfully used for modeling and analyzing functional aspects of complex systems, for example:

Model-checking models of *trillions* of states (or more ... 10^{10000})

Success stories

Classical FM have been successfully used for modeling and analyzing functional aspects of complex systems, for example:

Model-checking models of *trillions* of states (or more ... 10^{10000})

- Complex control software for space applications (e.g. NASA Mars rovers [COMPUTER, Jan. 04])

Success stories

Classical FM have been successfully used for modeling and analyzing functional aspects of complex systems, for example:

Model-checking models of *trillions* of states (or more ... 10^{10000})

- Complex control software for space applications (e.g. NASA Mars rovers [COMPUTER, Jan. 04])
- Complex control software for civil applications (e.g. Rotterdam Storm Surge Barrier, The Netherlands)

Success stories

Classical FM have been successfully used for modeling and analyzing functional aspects of complex systems, for example:

Model-checking models of *trillions* of states (or more ... 10^{10000})

- Complex control software for space applications (e.g. NASA Mars rovers [COMPUTER, Jan. 04])
- Complex control software for civil applications (e.g. Rotterdam Storm Surge Barrier, The Netherlands)
- Automotive & Railways (e.g. train interlocking & on board control systems)

Success stories

Classical FM have been successfully used for modeling and analyzing functional aspects of complex systems, for example:

Model-checking models of *trillions* of states (or more ... 10^{10000})

- Complex control software for space applications (e.g. NASA Mars rovers [COMPUTER, Jan. 04])
- Complex control software for civil applications (e.g. Rotterdam Storm Surge Barrier, The Netherlands)
- Automotive & Railways (e.g. train interlocking & on board control systems)
- Low level device control (Intel, Siemens, Microsoft)

Success stories

Classical FM have been successfully used for modeling and analyzing functional aspects of complex systems, for example:

Model-checking models of *trillions* of states (or more ... 10^{10000})

- Complex control software for space applications (e.g. NASA Mars rovers [COMPUTER, Jan. 04])
- Complex control software for civil applications (e.g. Rotterdam Storm Surge Barrier, The Netherlands)
- Automotive & Railways (e.g. train interlocking & on board control systems)
- Low level device control (Intel, Siemens, Microsoft)

. . .

Success stories

Classical FM have been successfully used for modeling and analyzing functional aspects of complex systems, for example:

Model-checking models of *trillions* of states (or more ... 10^{10000})

- Complex control software for space applications (e.g. NASA Mars rovers [COMPUTER, Jan. 04])
- Complex control software for civil applications (e.g. Rotterdam Storm Surge Barrier, The Netherlands)
- Automotive & Railways (e.g. train interlocking & on board control systems)
- Low level device control (Intel, Siemens, Microsoft)

. . .

(Automatic) Theorem Proving, e.g.

Success stories

Classical FM have been successfully used for modeling and analyzing functional aspects of complex systems, for example:

Model-checking models of *trillions* of states (or more ... 10^{10000})

- Complex control software for space applications (e.g. NASA Mars rovers [COMPUTER, Jan. 04])
- Complex control software for civil applications (e.g. Rotterdam Storm Surge Barrier, The Netherlands)
- Automotive & Railways (e.g. train interlocking & on board control systems)
- Low level device control (Intel, Siemens, Microsoft)

. . .

(Automatic) Theorem Proving, e.g.

Avionics systems (e.g. Boeing)

Classical FM have been successfully used for modeling and analyzing functional aspects of complex systems, for example:

Model-checking models of *trillions* of states (or more ... 10^{10000})

- Complex control software for space applications (e.g. NASA Mars rovers [COMPUTER, Jan. 04])
- Complex control software for civil applications (e.g. Rotterdam Storm Surge Barrier, The Netherlands)
- Automotive & Railways (e.g. train interlocking & on board control systems)
- Low level device control (Intel, Siemens, Microsoft)

. . .

(Automatic) Theorem Proving, e.g.

Avionics systems (e.g. Boeing)

...

Success stories

Classical FM have been successfully used for modeling and analyzing functional aspects of complex systems, for example:

Model-checking models of *trillions* of states (or more ... 10^{10000})

- Complex control software for space applications (e.g. NASA Mars rovers [COMPUTER, Jan. 04])
- Complex control software for civil applications (e.g. Rotterdam Storm Surge Barrier, The Netherlands)
- Automotive & Railways (e.g. train interlocking & on board control systems)
- Low level device control (Intel, Siemens, Microsoft)

. . .

(Automatic) Theorem Proving, e.g.

Avionics systems (e.g. Boeing)

...

Success stories

Things like even software verification, this has been the Holy Grail of computer science for many decades but now in some very key areas, for example, driver verification we're building tools that can do actual proof about the software and how it works in order to guarantee the reliability.

Success stories

Things like even software verification, this has been the Holy Grail of computer science for many decades but now in some very key areas, for example, driver verification we're building tools that can do actual proof about the software and how it works in order to guarantee the reliability.

Bill Gates, April 18, 2002.
Keynote address at WinHEC 2002

Timed/Probabilistic/Stochastic Extensions of Process Algebraic System Modelling and Temporal Logic Requirement Specification

Extensions of FM

A substantial contribution to the design of dependable systems can be provided by extensions of FM for the integrated modeling and analysis of functional and non-functional aspects of complex systems, e.g.

Extensions of FM

A substantial contribution to the design of dependable systems can be provided by extensions of FM for the integrated modeling and analysis of functional and non-functional aspects of complex systems, e.g.

- High level model specification languages

Extensions of FM

A substantial contribution to the design of dependable systems can be provided by extensions of FM for the integrated modeling and analysis of functional and non-functional aspects of complex systems, e.g.

- High level model specification languages e.g.

  Timed-/Probabilistic-/Stochastic-Process Calculi

Extensions of FM

A substantial contribution to the design of dependable systems can be provided by extensions of FM for the integrated modeling and analysis of functional and non-functional aspects of complex systems, e.g.

- High level model specification languages e.g.

  Timed-/Probabilistic-/Stochastic-Process Calculi

- High level (non-)functional requirement specification languages,

Extensions of FM

A substantial contribution to the design of dependable systems can be provided by extensions of FM for the integrated modeling and analysis of functional and non-functional aspects of complex systems, e.g.

- High level model specification languages e.g.

  Timed-/Probabilistic-/Stochastic-Process Calculi

- High level (non-)functional requirement specification languages, e.g.

  Timed-/Probabilistic-/Stochastic-Temporal Logics

Extensions of FM

A substantial contribution to the design of dependable systems can be provided by extensions of FM for the integrated modeling and analysis of functional and non-functional aspects of complex systems, e.g.

- High level model specification languages e.g.

  Timed-/Probabilistic-/Stochastic-Process Calculi

- High level (non-)functional requirement specification languages, e.g.

  Timed-/Probabilistic-/Stochastic-Temporal Logics

- Efficient verification techniques,

Extensions of FM

A substantial contribution to the design of dependable systems can be provided by extensions of FM for the integrated modeling and analysis of functional and non-functional aspects of complex systems, e.g.

- High level model specification languages e.g.

  Timed-/Probabilistic-/Stochastic-Process Calculi

- High level (non-)functional requirement specification languages, e.g.

  Timed-/Probabilistic-/Stochastic-Temporal Logics

- Efficient verification techniques, e.g.

  Timed-/Probabilistic-/Stochastic-Model-checkers

Extensions of FM

A substantial contribution to the design of dependable systems can be provided by extensions of FM for the integrated modeling and analysis of functional and non-functional aspects of complex systems, e.g.

- High level model specification languages e.g.

  Timed-/Probabilistic-/Stochastic- Process Calculi

- High level (non-)functional requirement specification languages, e.g.

  Timed-/Probabilistic-/Stochastic- Temporal Logics

- Efficient verification techniques, e.g.

  Timed-/Probabilistic-/Stochastic- Model-checkers

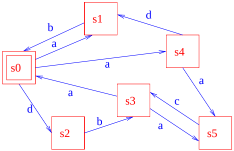
From algebraic terms to LTS via Formal Semantics.

Formal Syntax definition
(Grammar)

$$
\begin{aligned}
s0 &\triangleq a.s1 + d.s2 + a.s4\\
s1 &\triangleq b.s0\\
s2 &\triangleq b.s3\\
s3 &\triangleq a.s0 + a.s5\\
s4 &\triangleq d.s1 + a.s5\\
s5 &\triangleq c.s3
\end{aligned}
$$

Formal Semantics definition
(Logic deduction system)

Mathematical Objects
(LTS)



From algebraic terms to LTS via Formal Semantics.

$$
\begin{aligned}
s0 &\triangleq a.s1 + d.s2 + a.s4\\
s1 &\triangleq b.s0\\
s2 &\triangleq b.s3\\
s3 &\triangleq a.s0 + a.s5\\
s4 &\triangleq d.s1 + a.s5\\
s5 &\triangleq c.s3
\end{aligned}
$$

From algebraic terms to CTMC via Formal Semantics.

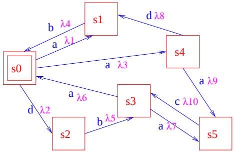
$$
\begin{aligned}
s0 &\triangleq a^{\lambda_1}.s1 + d^{\lambda_2}.s2 + a^{\lambda_3}.s4\\
s1 &\triangleq b^{\lambda_4}.s0\\
s2 &\triangleq b^{\lambda_5}.s3\\
s3 &\triangleq a^{\lambda_6}.s0 + a^{\lambda_7}.s5\\
s4 &\triangleq d^{\lambda_8}.s1 + a^{\lambda_9}.s5\\
s5 &\triangleq c^{\lambda_{10}}.s3
\end{aligned}
$$

Formal Syntax definition (Grammar)

$$
\begin{aligned}
s0 &\triangleq a^{\lambda_1}.s1 + d^{\lambda_2}.s2 + a^{\lambda_3}.s4\\
s1 &\triangleq b^{\lambda_4}.s0\\
s2 &\triangleq b^{\lambda_5}.s3\\
s3 &\triangleq a^{\lambda_6}.s0 + a^{\lambda_7}.s5\\
s4 &\triangleq d^{\lambda_8}.s1 + a^{\lambda_9}.s5\\
s5 &\triangleq c^{\lambda_{10}}.s3
\end{aligned}
$$

Formal Syntax definition
(Grammar)

$$
\begin{aligned}
s0 &\triangleq a^{\lambda_1}.s1 + d^{\lambda_2}.s2 + a^{\lambda_3}.s4\\
s1 &\triangleq b^{\lambda_4}.s0\\
s2 &\triangleq b^{\lambda_5}.s3\\
s3 &\triangleq a^{\lambda_6}.s0 + a^{\lambda_7}.s5\\
s4 &\triangleq d^{\lambda_8}.s1 + a^{\lambda_9}.s5\\
s5 &\triangleq c^{\lambda_{10}}.s3
\end{aligned}
$$

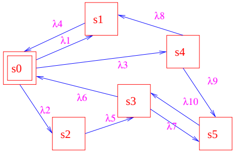


From algebraic terms to CTMC via Formal Semantics.

Formal Syntax definition
(Grammar)

$$
\begin{aligned}
s0 &\triangleq a^{\lambda_1}.s1 + d^{\lambda_2}.s2 + a^{\lambda_3}.s4\\
s1 &\triangleq b^{\lambda_4}.s0\\
s2 &\triangleq b^{\lambda_5}.s3\\
s3 &\triangleq a^{\lambda_6}.s0 + a^{\lambda_7}.s5\\
s4 &\triangleq d^{\lambda_8}.s1 + a^{\lambda_9}.s5\\
s5 &\triangleq c^{\lambda_{10}}.s3
\end{aligned}
$$

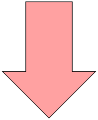
Mathematical Objects
(CTMC)



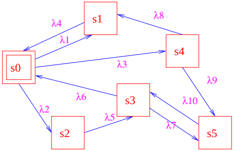
From algebraic terms to CTMC via Formal Semantics.

Formal Syntax definition
(Grammar)

$$
\begin{aligned}
s0 &\triangleq a^{\lambda_1}.s1 + d^{\lambda_2}.s2 + a^{\lambda_3}.s4\\
s1 &\triangleq b^{\lambda_4}.s0\\
s2 &\triangleq b^{\lambda_5}.s3\\
s3 &\triangleq a^{\lambda_6}.s0 + a^{\lambda_7}.s5\\
s4 &\triangleq d^{\lambda_8}.s1 + a^{\lambda_9}.s5\\
s5 &\triangleq c^{\lambda_{10}}.s3
\end{aligned}
$$



Mathematical Objects
(CTMC)



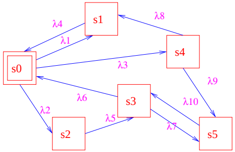
From algebraic terms to CTMC via Formal Semantics.

Formal Syntax definition (Grammar)

Formal Semantics definition (Logic deduction system)

Mathematical Objects (CTMC)

$$
\begin{aligned}
s0 &\triangleq a^{\lambda_1}.s1 + d^{\lambda_2}.s2 + a^{\lambda_3}.s4\\
s1 &\triangleq b^{\lambda_4}.s0\\
s2 &\triangleq b^{\lambda_5}.s3\\
s3 &\triangleq a^{\lambda_6}.s0 + a^{\lambda_7}.s5\\
s4 &\triangleq d^{\lambda_8}.s1 + a^{\lambda_9}.s5\\
s5 &\triangleq c^{\lambda_{10}}.s3
\end{aligned}
$$



A *process algebraic* approach to *stochastic* system modelling

A *process algebraic* approach to *stochastic* system modelling

1. Algebraic terms

A *process algebraic* approach to *stochastic* system modelling

1. Algebraic terms, defined via a Formal syntax, e.g. Hillston PEPA-like:

   $S ::= \mathbf{nil} \;\Big|\; (\alpha, \lambda).S \;\Big|\; S + S \;\Big|\; (\alpha, \lambda).X \;\Big|\; S|[\alpha_1, \ldots, \alpha_n]|S$

   with $\alpha, \alpha_1 \ldots . \alpha_n \in A_t$, $\lambda > 0$,

   and constants defined via equations $X \triangleq S$

A *process algebraic* approach to *stochastic* system modelling

1. Algebraic terms, defined via a Formal syntax, e.g. Hillston PEPA-like:

   $S ::= \textbf{nil} \;\Big|\; (\alpha, \lambda).S \;\Big|\; S + S \;\Big|\; (\alpha, \lambda).X \;\Big|\; S|[\alpha_1, \ldots, \alpha_n]|S$
   with $\alpha, \alpha_1 \ldots . \alpha_n \in A_t$, $\lambda > 0$,

   and constants defined via equations $X \triangleq S$

   with **graphical tool support**;

A *process algebraic* approach to *stochastic* system modelling

1. Algebraic terms, defined via a Formal syntax, e.g. Hillston PEPA-like:

   $$S ::= \mathbf{nil} \;\Big|\; (\alpha, \lambda).S \;\Big|\; S + S \;\Big|\; (\alpha, \lambda).X \;\Big|\; S|[\alpha_1, \ldots, \alpha_n]|S$$

   with $\alpha, \alpha_1 \ldots . \alpha_n \in A_t$, $\lambda > 0$,

   and constants defined via equations $X \triangleq S$

   with **graphical tool support**;

2. CTMC

A *process algebraic* approach to *stochastic* system modelling

1. Algebraic terms, defined via a Formal syntax, e.g. Hillston PEPA-like:

   $S ::= \mathbf{nil} \mid (\alpha, \lambda).S \mid S + S \mid (\alpha, \lambda).X \mid S|[\alpha_1, \ldots, \alpha_n]|S$
   with $\alpha, \alpha_1 \ldots . \alpha_n \in A_t$, $\lambda > 0$,

   and constants defined via equations $X \triangleq S$

   with **graphical tool support**;

2. CTMC, the reference Mathematical Objects

A *process algebraic* approach to *stochastic* system modelling

1. Algebraic terms, defined via a Formal syntax, e.g. Hillston PEPA-like:

   $$S ::= \mathbf{nil} \;\Big|\; (\alpha, \lambda).S \;\Big|\; S + S \;\Big|\; (\alpha, \lambda).X \;\Big|\; S|[\alpha_1, \ldots, \alpha_n]|S$$

   with $\alpha, \alpha_1 \ldots . \alpha_n \in A_t$, $\lambda > 0$,

   and constants defined via equations $X \triangleq S$

   with **graphical tool support**;

2. CTMC, the reference Mathematical Objects, equipped with Behavioural Relations

A *process algebraic* approach to *stochastic* system modelling

1. Algebraic terms, defined via a Formal syntax, e.g. Hillston PEPA-like:

   $$S ::= \mathbf{nil} \;\Big|\; (\alpha, \lambda).S \;\Big|\; S + S \;\Big|\; (\alpha, \lambda).X \;\Big|\; S|[\alpha_1, \ldots, \alpha_n]|S$$

   with $\alpha, \alpha_1 \ldots . \alpha_n \in A_t$, $\lambda > 0$,

   and constants defined via equations $X \triangleq S$

   with **graphical tool support**;

2. CTMC, the reference Mathematical Objects, equipped with Behavioural Relations, i.e. Formal Equivalences, e.g. Lumping

A *process algebraic* approach to *stochastic* system modelling

1. Algebraic terms, defined via a Formal syntax, e.g. Hillston PEPA-like:

   $$S ::= \textbf{nil} \;\Big|\; (\alpha, \lambda).S \;\Big|\; S + S \;\Big|\; (\alpha, \lambda).X \;\Big|\; S|[\alpha_1, \ldots, \alpha_n]|S$$

   with $\alpha, \alpha_1 \ldots . \alpha_n \in A_t$, $\lambda > 0$,

   and constants defined via equations $X \triangleq S$

   with **graphical tool support**;

2. CTMC, the reference Mathematical Objects, equipped with Behavioural Relations, i.e. Formal Equivalences, e.g. Lumping

3. A mapping of terms to CTMC

A *process algebraic* approach to *stochastic* system modelling

1. Algebraic terms, defined via a Formal syntax, e.g. Hillston PEPA-like:

   $S ::= \mathbf{nil} \;\Big|\; (\alpha, \lambda).S \;\Big|\; S + S \;\Big|\; (\alpha, \lambda).X \;\Big|\; S|[\alpha_1, \ldots, \alpha_n]|S$
   with $\alpha, \alpha_1 \ldots . \alpha_n \in A_t$, $\lambda > 0$,

   and constants defined via equations $X \triangleq S$

   with **graphical tool support**;

2. CTMC, the reference Mathematical Objects, equipped with Behavioural Relations, i.e. Formal Equivalences, e.g. Lumping

3. A mapping of terms to CTMC, the Formal Semantics definition

A *process algebraic* approach to *stochastic* system modelling

1. Algebraic terms, defined via a Formal syntax, e.g. Hillston PEPA-like:

   $S ::= \mathbf{nil} \; \Big| \; (\alpha, \lambda).S \; \Big| \; S + S \; \Big| \; (\alpha, \lambda).X \; \Big| \; S|[\alpha_1, \ldots, \alpha_n]|S$
   with $\alpha, \alpha_1 \ldots . \alpha_n \in A_t$, $\lambda > 0$,
   and constants defined via equations $X \triangleq S$

   with **graphical tool support**;

2. CTMC, the reference Mathematical Objects, equipped with Behavioural Relations, i.e. Formal Equivalences, e.g. Lumping

3. A mapping of terms to CTMC, the Formal Semantics definition, e.g.:

$$(\alpha, \lambda).S \xrightarrow{(\alpha,\lambda)} S \qquad \frac{S_1 \xrightarrow{(\alpha,\lambda)} S}{S_1 + S_2 \xrightarrow{(\alpha,\lambda)} S} \qquad \ldots$$

A *process algebraic* approach to *stochastic* system modelling

1. Algebraic terms, defined via a Formal syntax, e.g. Hillston PEPA-like:

   $S ::= \textbf{nil} \;\Big|\; (\alpha, \lambda).S \;\Big|\; S + S \;\Big|\; (\alpha, \lambda).X \;\Big|\; S|[\alpha_1, \ldots, \alpha_n]|S$
   with $\alpha, \alpha_1 \ldots . \alpha_n \in A_t$, $\lambda > 0$,
   and constants defined via equations $X \triangleq S$

   with **graphical tool support**;

2. CTMC, the reference Mathematical Objects, equipped with Behavioural Relations, i.e. Formal Equivalences, e.g. Lumping

3. A mapping of terms to CTMC, the Formal Semantics definition, e.g.:

$$(\alpha, \lambda).S \xrightarrow{(\alpha,\lambda)} S \qquad \frac{S_1 \xrightarrow{(\alpha,\lambda)} S}{S_1 + S_2 \xrightarrow{(\alpha,\lambda)} S} \qquad \ldots$$

4. Algebraic terms manipulation rules

A *process algebraic* approach to *stochastic* system modelling

1. Algebraic terms, defined via a Formal syntax, e.g. Hillston PEPA-like:

   $$S ::= \mathbf{nil} \;\Big|\; (\alpha, \lambda).S \;\Big|\; S + S \;\Big|\; (\alpha, \lambda).X \;\Big|\; S|[\alpha_1, \ldots, \alpha_n]|S$$

   with $\alpha, \alpha_1 \ldots . \alpha_n \in A_t$, $\lambda > 0$,

   and constants defined via equations $X \triangleq S$

   with **graphical tool support**;

2. CTMC, the reference Mathematical Objects, equipped with Behavioural Relations, i.e. Formal Equivalences, e.g. Lumping

3. A mapping of terms to CTMC, the Formal Semantics definition, e.g.:

   $$(\alpha, \lambda).S \xrightarrow{(\alpha,\lambda)} S \qquad \frac{S_1 \xrightarrow{(\alpha,\lambda)} S}{S_1 + S_2 \xrightarrow{(\alpha,\lambda)} S} \qquad \ldots$$

4. Algebraic terms manipulation rules, i.e. Axiomatizations of Equivalences

A *process algebraic* approach to *stochastic* system modelling

1. Algebraic terms, defined via a Formal syntax, e.g. Hillston PEPA-like:

   $S ::= \textbf{nil} \mid (\alpha, \lambda).S \mid S + S \mid (\alpha, \lambda).X \mid S|[\alpha_1, \ldots, \alpha_n]|S$

   with $\alpha, \alpha_1 \ldots . \alpha_n \in A_t$, $\lambda > 0$,

   and constants defined via equations $X \triangleq S$

   with **graphical tool support**;

2. CTMC, the reference Mathematical Objects, equipped with Behavioural Relations, i.e. Formal Equivalences, e.g. Lumping

3. A mapping of terms to CTMC, the Formal Semantics definition, e.g.:

$$(\alpha, \lambda).S \xrightarrow{(\alpha,\lambda)} S \qquad \frac{S_1 \xrightarrow{(\alpha,\lambda)} S}{S_1 + S_2 \xrightarrow{(\alpha,\lambda)} S} \qquad \ldots$$

4. Algebraic terms manipulation rules, i.e. Axiomatizations of Equivalences plus standard CTMC analysis techniques

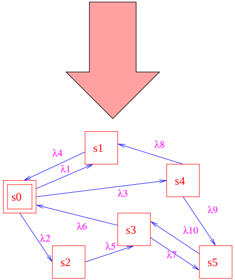
From logic formulae to CTMC via Formal Semantics.

Formal Syntax definition
(Grammar)

Formal Semantics definition
(Logic deduction system)

Mathematical Objects
(CTMC, Cones, Cilynders)

$$s1 \models \exists((\underline{in}(s1) \vee \underline{in}(s2) \vee \underline{in}(s3)) \;\; \mathcal{U} \;\; \underline{in}(s4))$$



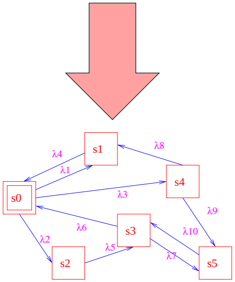
From logic formulae to CTMC via Formal Semantics.

Formal Syntax definition
(Grammar)

Formal Semantics definition
(Logic deduction system)

Mathematical Objects
(CTMC, Cones, Cilynders)

$s1 \models \mathcal{P}_{>0.8}((\underline{in}(s1) \vee \underline{in}(s2) \vee \underline{in}(s3))\ \ \mathcal{U}^{6.34}\ \underline{in}(s4))$



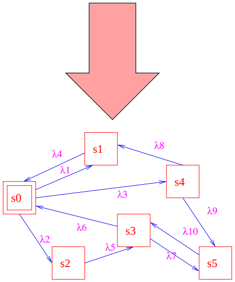
From logic formulae to CTMC via Formal Semantics.

Formal Syntax definition
(Grammar)

Formal Semantics definition
(Logic deduction system)

Mathematical Objects
(CTMC, Cones, Cilynders)

$s1 \models \mathcal{S}_{>0.6}(\underline{in}(s0) \vee \underline{in}(s3) \vee \underline{in}(s5))$



A *stochastic logics* approach to *non-functional* Requirement specification

A *stochastic logics* approach to *non-functional* Requirement specification

1. Logic formulae

A *stochastic logics* approach to *non-functional* Requirement specification

1. Logic formulae, defined via a Formal syntax, e.g. Baier et al. CSL-like:

$$\mathcal{A} ::= \mathrm{tt} \;\Big|\; a \;\Big|\; \ldots$$
$$\Phi ::= \mathcal{A} \;\Big|\; \neg\Phi \;\Big|\; \Phi \wedge \Phi \;\Big|\; \Phi \vee \Phi \;\Big|\; \mathcal{S}_{\bowtie p}(\Phi) \;\Big|\; \mathcal{P}_{\bowtie p}(\varphi)$$
$$\varphi ::= \mathbf{X}^{t}\,\Phi \;\Big|\; \Phi \;\; \mathcal{U}^{t}\,\Phi$$

A *stochastic logics* approach to *non-functional* Requirement specification

1. Logic formulae, defined via a Formal syntax, e.g. Baier et al. CSL-like:

$$\mathcal{A} ::= \mathrm{tt} \;\Big|\; a \;\Big|\; \ldots$$
$$\Phi ::= \mathcal{A} \;\Big|\; \neg\Phi \;\Big|\; \Phi \wedge \Phi \;\Big|\; \Phi \vee \Phi \;\Big|\; \mathcal{S}_{\bowtie p}(\Phi) \;\Big|\; \mathcal{P}_{\bowtie p}(\varphi)$$
$$\varphi ::= \mathbf{X}^t\, \Phi \;\Big|\; \Phi \;\; \mathcal{U}^t\, \Phi$$

2. CTMC

A *stochastic logics* approach to *non-functional* Requirement specification

1. Logic formulae, defined via a Formal syntax, e.g. Baier et al. CSL-like:

$$\mathcal{A} ::= \mathrm{tt} \;\Big|\; a \;\Big|\; \ldots$$
$$\Phi ::= \mathcal{A} \;\Big|\; \neg\Phi \;\Big|\; \Phi \wedge \Phi \;\Big|\; \Phi \vee \Phi \;\Big|\; \mathcal{S}_{\bowtie p}(\Phi) \;\Big|\; \mathcal{P}_{\bowtie p}(\varphi)$$
$$\varphi ::= \mathbf{X}^{t}\, \Phi \;\Big|\; \Phi \;\; \mathcal{U}^{t}\, \Phi$$

2. CTMC, the reference Mathematical Objects

A *stochastic logics* approach to *non-functional* Requirement specification

1. Logic formulae, defined via a Formal syntax, e.g. Baier et al. CSL-like:

$$\mathcal{A} ::= \text{tt} \;\Big|\; a \;\Big|\; \ldots$$
$$\Phi ::= \mathcal{A} \;\Big|\; \neg\Phi \;\Big|\; \Phi \wedge \Phi \;\Big|\; \Phi \vee \Phi \;\Big|\; \mathcal{S}_{\bowtie p}(\Phi) \;\Big|\; \mathcal{P}_{\bowtie p}(\varphi)$$
$$\varphi ::= \mathbf{X}^t\; \Phi \;\Big|\; \Phi \;\; \mathcal{U}^t\; \Phi$$

2. CTMC, the reference Mathematical Objects, equipped with proper theory

A *stochastic logics* approach to *non-functional* Requirement specification

1. Logic formulae, defined via a Formal syntax, e.g. Baier et al. CSL-like:

$$\mathcal{A} ::= \mathrm{tt} \;\Big|\; a \;\Big|\; \ldots$$
$$\Phi ::= \mathcal{A} \;\Big|\; \neg\Phi \;\Big|\; \Phi \wedge \Phi \;\Big|\; \Phi \vee \Phi \;\Big|\; \mathcal{S}_{\bowtie p}(\Phi) \;\Big|\; \mathcal{P}_{\bowtie p}(\varphi)$$
$$\varphi ::= \mathbf{X}^t\,\Phi \;\Big|\; \Phi\;\;\mathcal{U}^t\,\Phi$$

2. CTMC, the reference Mathematical Objects, equipped with proper theory

3. A relation between formulae and CTMC

A *stochastic logics* approach to *non-functional* Requirement specification

1. Logic formulae, defined via a Formal syntax, e.g. Baier et al. CSL-like:

$$\mathcal{A} ::= \mathrm{tt} \;\Big|\; a \;\Big|\; \ldots$$
$$\Phi ::= \mathcal{A} \;\Big|\; \neg\Phi \;\Big|\; \Phi \wedge \Phi \;\Big|\; \Phi \vee \Phi \;\Big|\; \mathcal{S}_{\bowtie p}(\Phi) \;\Big|\; \mathcal{P}_{\bowtie p}(\varphi)$$
$$\varphi ::= \mathbf{X}^t\,\Phi \;\Big|\; \Phi \;\; \mathcal{U}^t\,\Phi$$

2. CTMC, the reference Mathematical Objects, equipped with proper theory

3. A relation between formulae and CTMC, the Formal Semantics definition

A *stochastic logics* approach to *non-functional* Requirement specification

1. Logic formulae, defined via a Formal syntax, e.g. Baier et al. CSL-like:

$$\mathcal{A} ::= \mathrm{tt} \;\Big|\; a \;\Big|\; \ldots$$
$$\Phi ::= \mathcal{A} \;\Big|\; \neg\Phi \;\Big|\; \Phi \wedge \Phi \;\Big|\; \Phi \vee \Phi \;\Big|\; \mathcal{S}_{\bowtie p}(\Phi) \;\Big|\; \mathcal{P}_{\bowtie p}(\varphi)$$
$$\varphi ::= \mathbf{X}^t\, \Phi \;\Big|\; \Phi \;\; \mathcal{U}^t\, \Phi$$

2. CTMC, the reference Mathematical Objects, equipped with proper theory

3. A relation between formulae and CTMC, the Formal Semantics definition

$$\gamma \models \mathbf{X}^t\, \Phi \text{ iff } \gamma[1] \text{ is reached by time } t \text{ and } \gamma[1] \models \Phi.$$

A *stochastic logics* approach to *non-functional* Requirement specification

1. Logic formulae, defined via a Formal syntax, e.g. Baier et al. CSL-like:

$$\mathcal{A} ::= \mathrm{tt} \;\Big|\; a \;\Big|\; \ldots$$
$$\Phi ::= \mathcal{A} \;\Big|\; \neg\Phi \;\Big|\; \Phi \wedge \Phi \;\Big|\; \Phi \vee \Phi \;\Big|\; \mathcal{S}_{\bowtie p}(\Phi) \;\Big|\; \mathcal{P}_{\bowtie p}(\varphi)$$
$$\varphi ::= \mathbf{X}^t\,\Phi \;\Big|\; \Phi\;\;\mathcal{U}^t\,\Phi$$

2. CTMC, the reference Mathematical Objects, equipped with proper theory

3. A relation between formulae and CTMC, the Formal Semantics definition

$\gamma \models \Phi_1\;\;\mathcal{U}^t\,\Phi_2$ iff there exists $j \geq 0$ s.t. $\gamma[j]$ is is reached by time t, $\gamma[j] \models \Phi_2$, and $\gamma[i] \models \Phi_2$, for all $0 \leq i < j$

A *stochastic logics* approach to *non-functional* Requirement specification

1. Logic formulae, defined via a Formal syntax, e.g. Baier et al. CSL-like:

$$\mathcal{A} ::= \mathrm{tt} \;\Big|\; a \;\Big|\; \ldots$$
$$\Phi ::= \mathcal{A} \;\Big|\; \neg\Phi \;\Big|\; \Phi \wedge \Phi \;\Big|\; \Phi \vee \Phi \;\Big|\; \mathcal{S}_{\bowtie p}(\Phi) \;\Big|\; \mathcal{P}_{\bowtie p}(\varphi)$$
$$\varphi ::= \mathbf{X}^t\, \Phi \;\Big|\; \Phi \;\; \mathcal{U}^t\, \Phi$$

2. CTMC, the reference Mathematical Objects, equipped with proper theory

3. A relation between formulae and CTMC, the Formal Semantics definition

$$s \models \mathcal{P}_{\geq p}(\varphi) \text{ iff } \mathbb{P}\{\gamma \mid \gamma \models \varphi\} \geq p$$

A *stochastic logics* approach to *non-functional* Requirement specification

1. Logic formulae, defined via a Formal syntax, e.g. Baier et al. CSL-like:

$$\mathcal{A} ::= \mathrm{tt} \;\Big|\; a \;\Big|\; \ldots$$
$$\Phi ::= \mathcal{A} \;\Big|\; \neg\Phi \;\Big|\; \Phi \wedge \Phi \;\Big|\; \Phi \vee \Phi \;\Big|\; \mathcal{S}_{\bowtie p}(\Phi) \;\Big|\; \mathcal{P}_{\bowtie p}(\varphi)$$
$$\varphi ::= \mathbf{X}^t\, \Phi \;\Big|\; \Phi \;\; \mathcal{U}^t\, \Phi$$

2. CTMC, the reference Mathematical Objects, equipped with proper theory

3. A relation between formulae and CTMC, the Formal Semantics definition

$$s \models \mathcal{P}_{<p}(\varphi) \text{ iff } \mathbb{P}\{\gamma \mid \gamma \models \varphi\} < p$$

A *stochastic logics* approach to *non-functional* Requirement specification

1. Logic formulae, defined via a Formal syntax, e.g. Baier et al. CSL-like:

$$\mathcal{A} ::= \mathrm{tt} \;\Big|\; a \;\Big|\; \ldots$$
$$\Phi ::= \mathcal{A} \;\Big|\; \neg\Phi \;\Big|\; \Phi \wedge \Phi \;\Big|\; \Phi \vee \Phi \;\Big|\; \mathcal{S}_{\bowtie p}(\Phi) \;\Big|\; \mathcal{P}_{\bowtie p}(\varphi)$$
$$\varphi ::= \mathbf{X}^t\, \Phi \;\Big|\; \Phi \;\; \mathcal{U}^t\, \Phi$$

2. CTMC, the reference Mathematical Objects, equipped with proper theory

3. A relation between formulae and CTMC, the Formal Semantics definition

   $s \models \mathcal{S}_{\geq p}(\Phi)$ iff the probability to be in a state s' s.t. $s' \models \Phi$, in the *long run* starting from s, is $\geq p$.

A *stochastic logics* approach to *non-functional* Requirement specification

1. Logic formulae, defined via a Formal syntax, e.g. Baier et al. CSL-like:

$$\mathcal{A} ::= \mathrm{tt} \;\Big|\; a \;\Big|\; \ldots$$
$$\Phi ::= \mathcal{A} \;\Big|\; \neg\Phi \;\Big|\; \Phi \wedge \Phi \;\Big|\; \Phi \vee \Phi \;\Big|\; \mathcal{S}_{\bowtie p}(\Phi) \;\Big|\; \mathcal{P}_{\bowtie p}(\varphi)$$
$$\varphi ::= \mathbf{X}^t\,\Phi \;\Big|\; \Phi \;\; \mathcal{U}^t\,\Phi$$

2. CTMC, the reference Mathematical Objects, equipped with proper theory

3. A relation between formulae and CTMC, the Formal Semantics definition

4. **Automatic verification**

A *stochastic logics* approach to *non-functional* Requirement specification

1. Logic formulae, defined via a Formal syntax, e.g. Baier et al. CSL-like:

$$\mathcal{A} ::= \text{tt} \mid a \mid \ldots$$
$$\Phi ::= \mathcal{A} \mid \neg\Phi \mid \Phi \wedge \Phi \mid \Phi \vee \Phi \mid \mathcal{S}_{\bowtie p}(\Phi) \mid \mathcal{P}_{\bowtie p}(\varphi)$$
$$\varphi ::= \mathbf{X}^t\, \Phi \mid \Phi \;\; \mathcal{U}^t\, \Phi$$

2. CTMC, the reference Mathematical Objects, equipped with proper theory
3. A relation between formulae and CTMC, the Formal Semantics definition
4. **Automatic verification, i.e. Stochastic Model Checking**

THANK YOU!!

- Jan A. Bergstra, Alban Ponse, and Scott A. Smolka (editors). *Handbook of Process Algebra.*
  Elsevier, ISBN: 0-444-82830-3, 2001.
- Christel Baier and Joost-Pieter Katoen. *Principles of Model Checking*
  MIT Press 2008
- C. A. R. Hoare. *Communicating Sequential Processes.*
  Prentice Hall International. 2004
- Robin Milner. *A Calculus of Communicating Systems*
  Springer Verlag ISBN 0-387-10235-3. 1980.
- Robin Milner. *Communication and Concurrency*
  Prentice Hall, International Series in Computer Science
  ISBN 0-131-15007-3. 1989
- Matthew Hennessy. *Algebraic Theory of Processes*
  MIT Press 1988.
- Edmund Clarke, Orna Grumberg, Doron Peled. *Model Checking*
  The MIT Press 2000.
- Gerald Holzmann. *The SPIN Model Checker. Primer and Reference Manual*
  Addison-Wesley 2003.