US 20110054639A1

(19) United States
(12) Patent Application Publication
Pazzi
(10) Pub. No.: US 2011/0054639 A1
(43) Pub. Date: Mar. 3, 2011

# (54) METHOD FOR ENSURING SAFETY AND LIVENESS RULES IN A STATE BASED DESIGN

(76) Inventor: **Luca Pazzi**, Modena (IT)

(21) Appl. No.: **12/865,413**

(22) PCT Filed: **Feb. 1, 2008**

(86) PCT No.: **PCT/EP2008/051300**

§ 371 (c)(1),
(2), (4) Date: **Nov. 16, 2010**

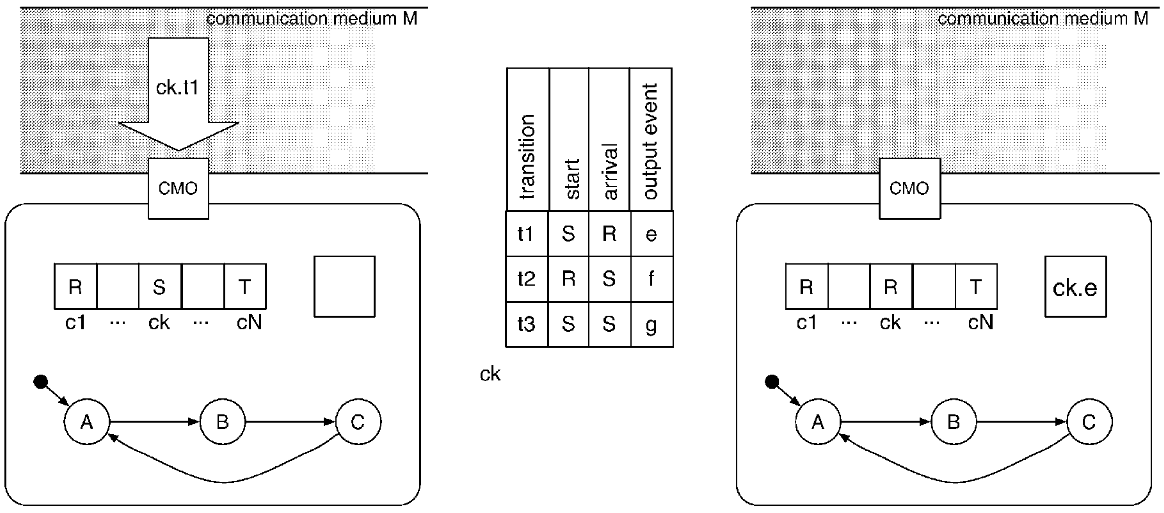
**Publication Classification**

(51) **Int. Cl.**
***G05B 9/02*** (2006.01)
***G05B 11/01*** (2006.01)

(52) **U.S. Cl.** ........................................................ **700/12**

(57) **ABSTRACT**

A method for controlling one or more physical machines for ensuring safety and liveness rules in a state based design of the physical machines, includes the steps of associating at least one logical state to at least one physical state that the physical machine or assemblage of physical machines may assume, providing state constraints for the logical states, and checking that a physical state assumed by the physical machines is associated to a logical state complying with the state constraints.



| transition | start | arrival | output event |
|---|---|---|---|
| t1 | S | R | e |
| t2 | R | S | f |
| t3 | S | S | g |

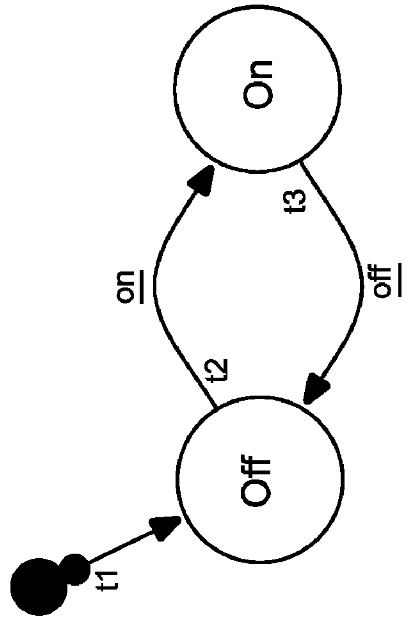


Fig. 1

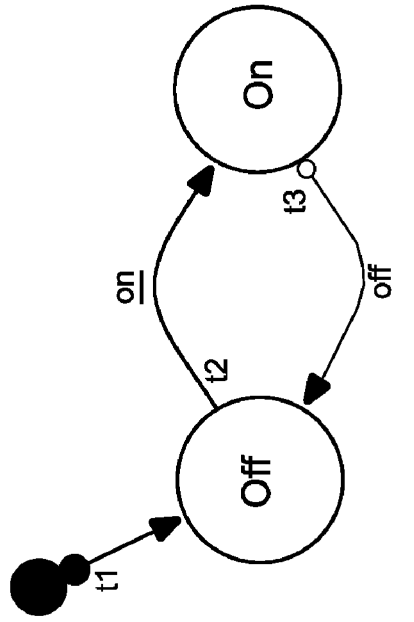


Fig. 2

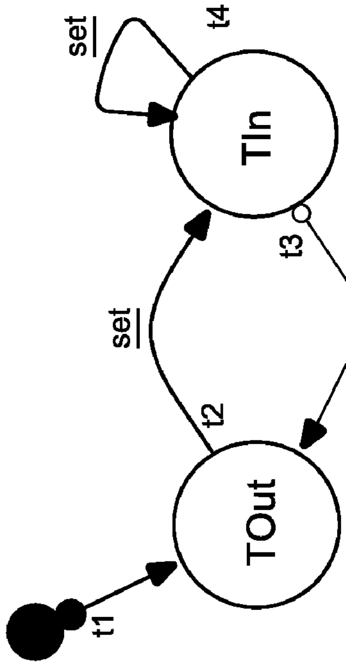


Fig. 3

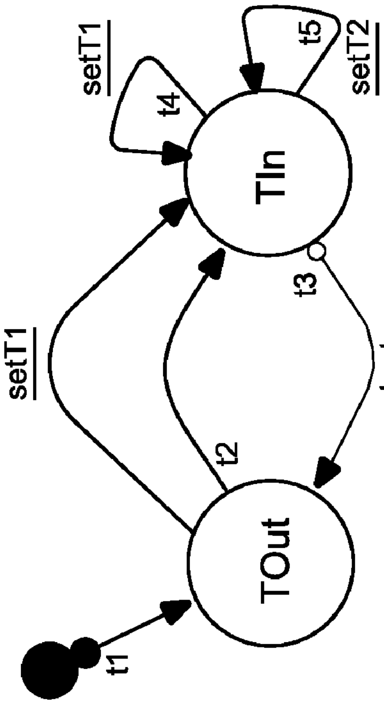


Fig. 4

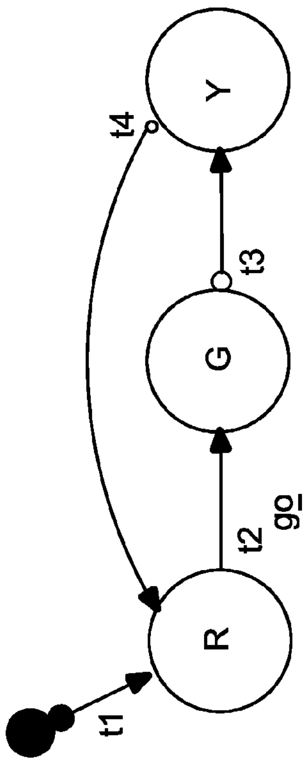


Fig. 5

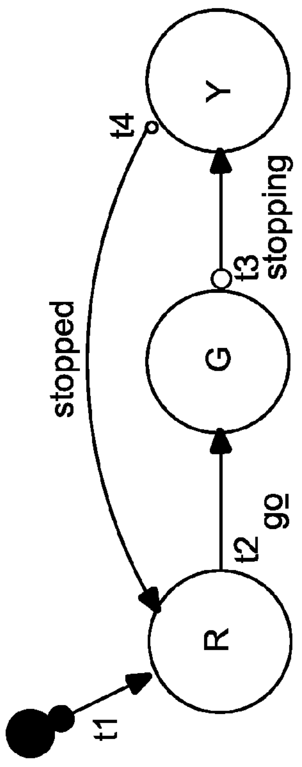


Fig. 6

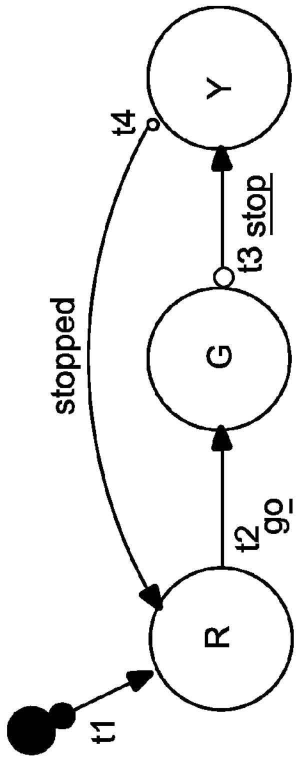


Fig. 7

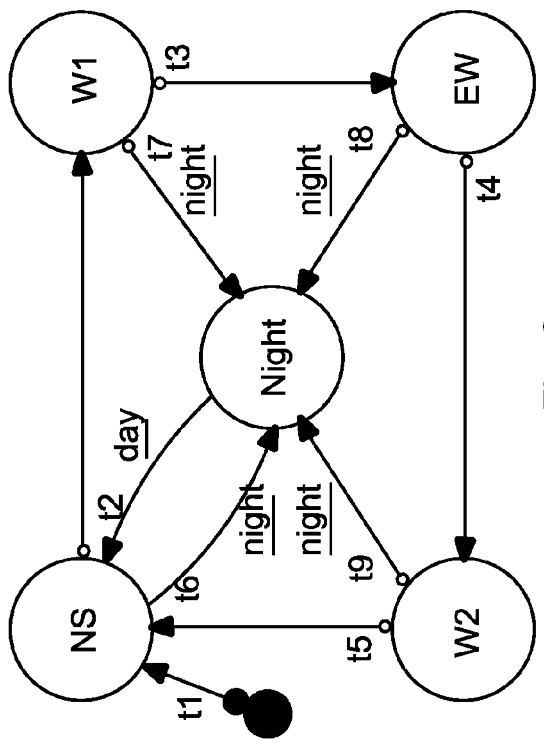


Fig. 8

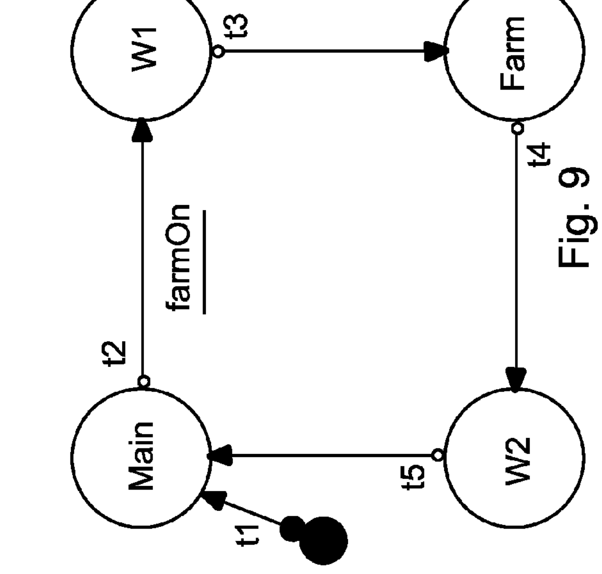


Fig. 9

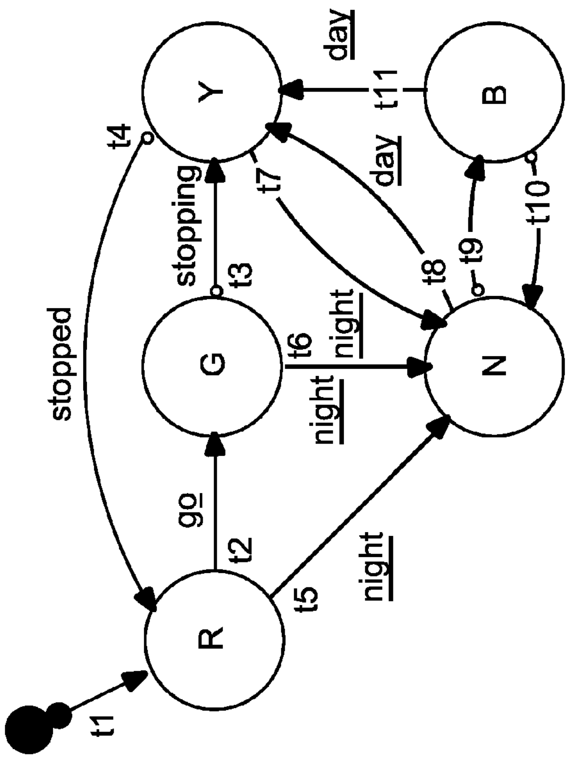


Fig. 10

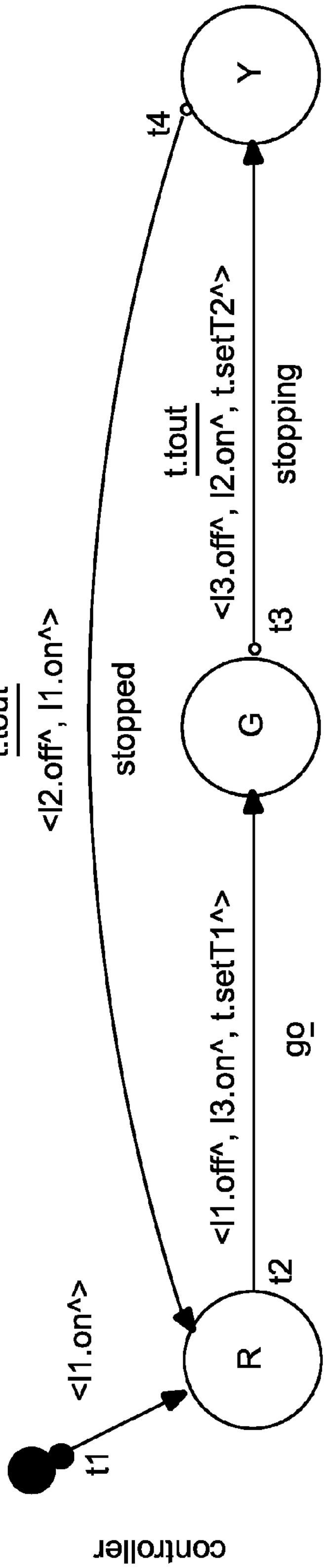


Fig. 11

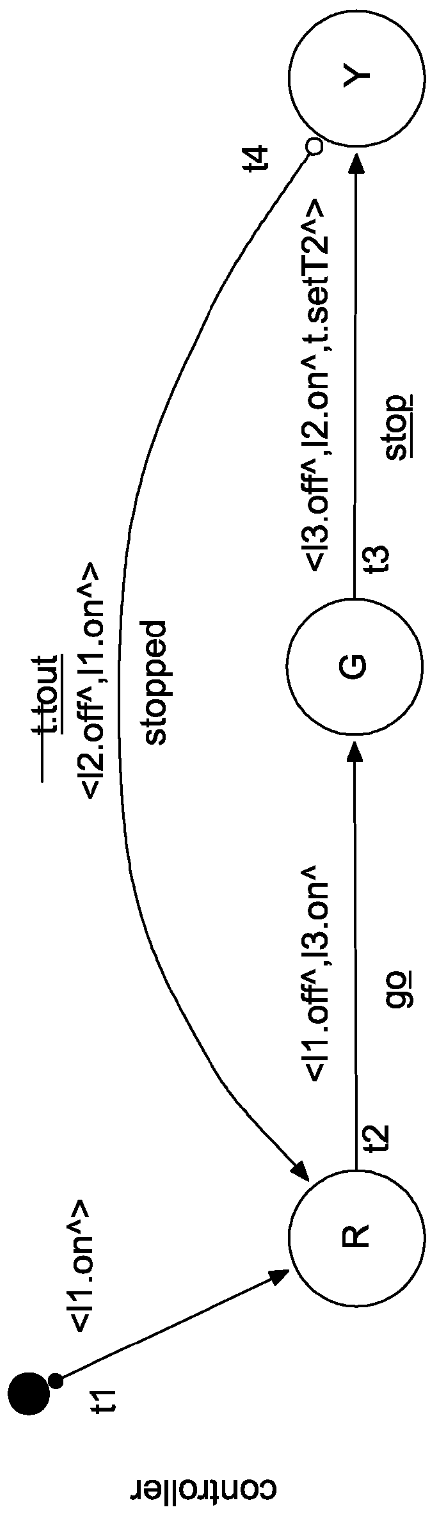

t1
<l1.on^>
R
t2
<l1.off^,l3.on^
go
G
t3
<l3.off^,l2.on^,t.setT2^>
stop
Y
t4
t.tout
<l2.off^,l1.on^>
stopped
controller

Fig. 12

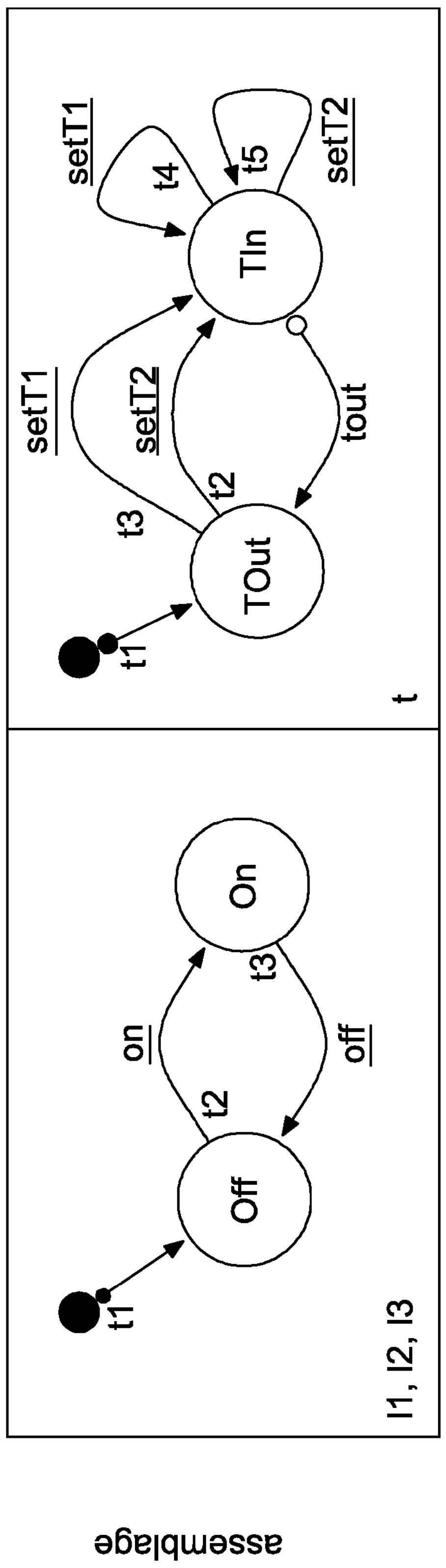
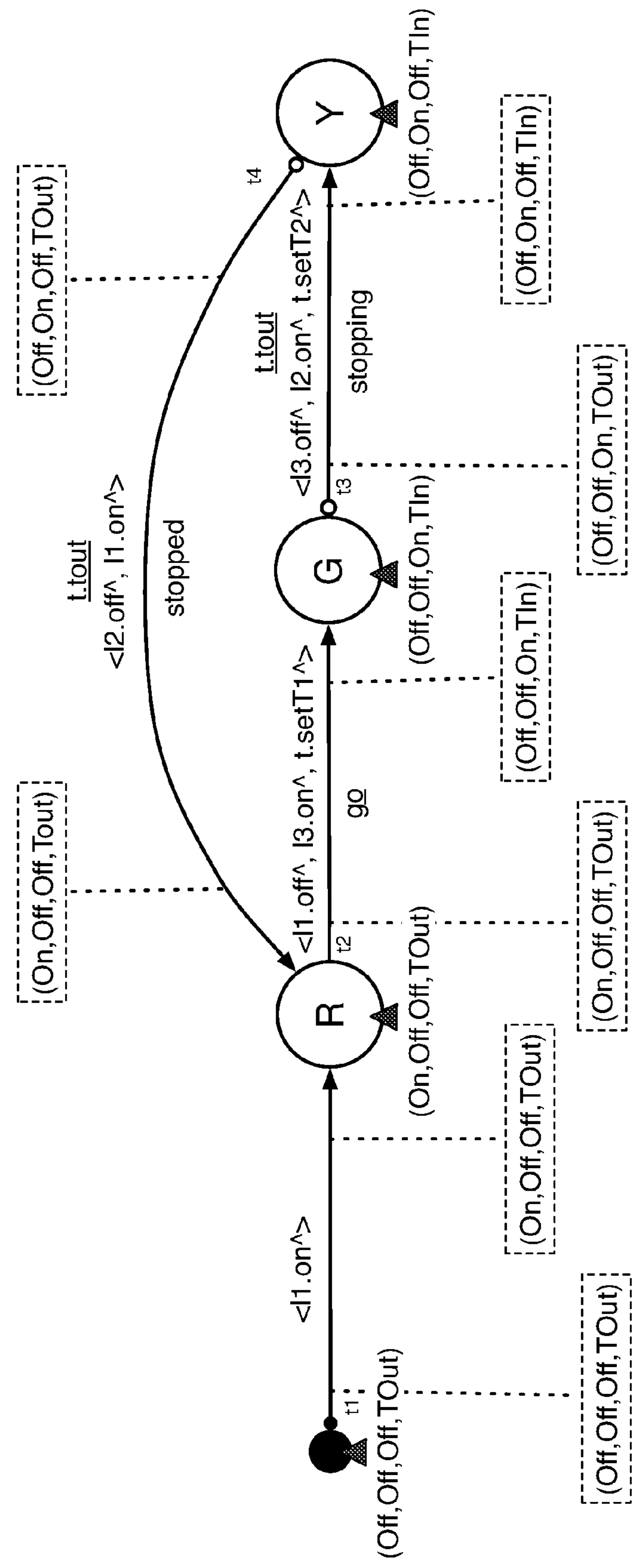

l1, l2, l3
t1
Off
t2
on
On
t3
off
t
t1
TOut
t3
setT1
t2
setT2
tout
TIn
t4
setT1
t5
setT2
assemblage

Fig. 13



(Off,Off,Off,TOut)
t1
<l1.on^>
(Off,Off,Off,TOut)
(On,Off,Off,TOut)
R
(On,Off,Off,TOut)
(On,Off,Off,TOut)
t2
<l1.off^, l3.on^, t.setT1^>
go
(Off,Off,On,TIn)
G
(Off,Off,On,TIn)
(Off,Off,On,TOut)
t3
t.tout
<l3.off^, l2.on^, t.setT2^>
stopping
(Off,On,Off,TIn)
Y
(Off,On,Off,TIn)
t4
t.tout
<l2.off^, l1.on^>
stopped
(Off,On,Off,TOut)
(On,Off,Off,Tout)

Fig. 14

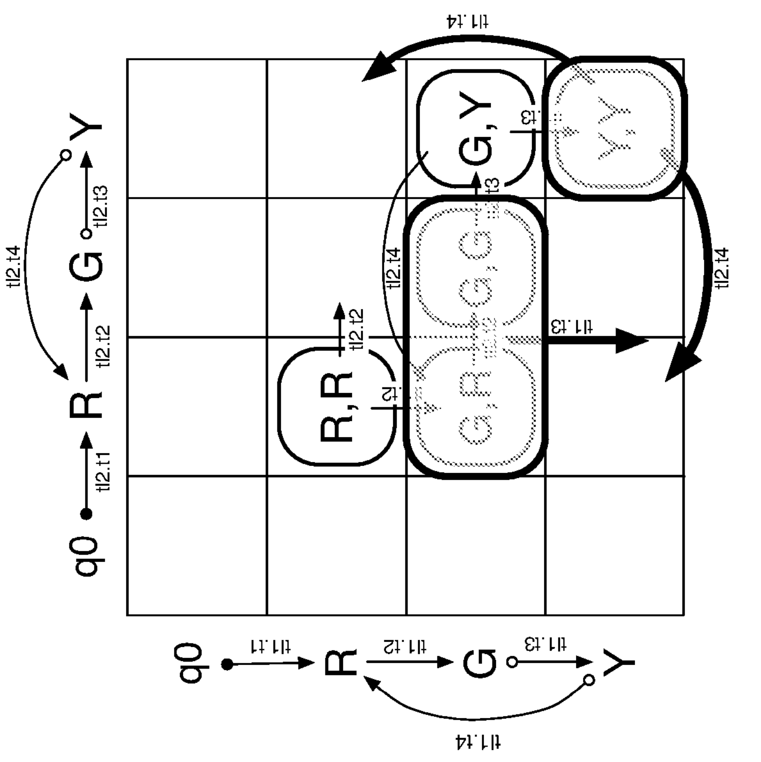

q0
R
G
Y
tl2.t1
tl2.t2
tl2.t3
tl2.t4
tl1.t1
tl1.t2
tl1.t3
tl1.t4
R,R
G,R
G,G
G,Y
Y,Y

Fig. 16

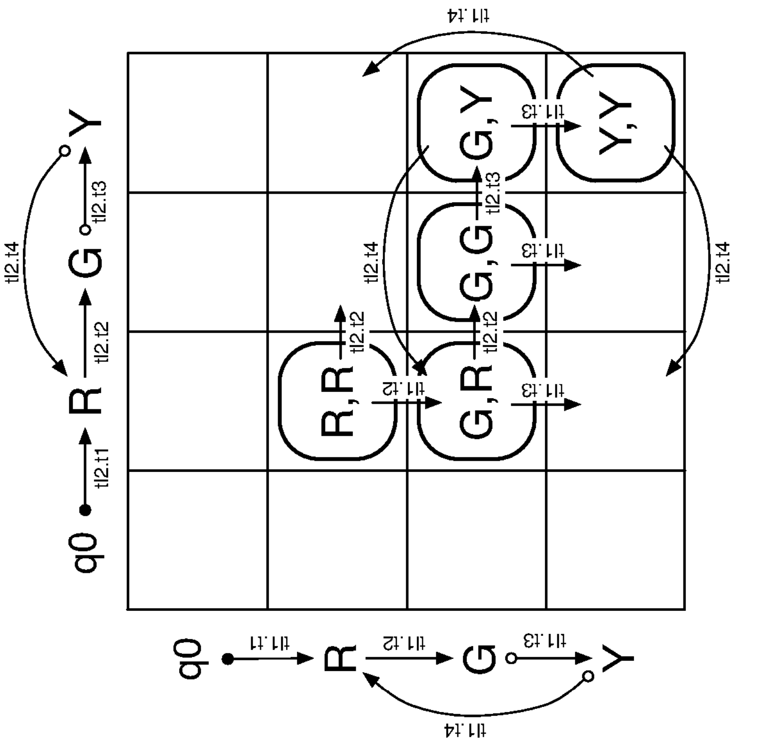

q0
R
G
Y
tl2.t1
tl2.t2
tl2.t3
tl2.t4
tl1.t1
tl1.t2
tl1.t3
tl1.t4
R,R
G,R
G,G
G,Y
Y,Y

Fig. 15

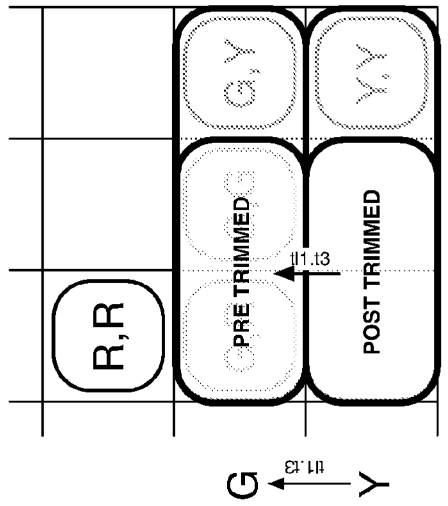


Fig. 19

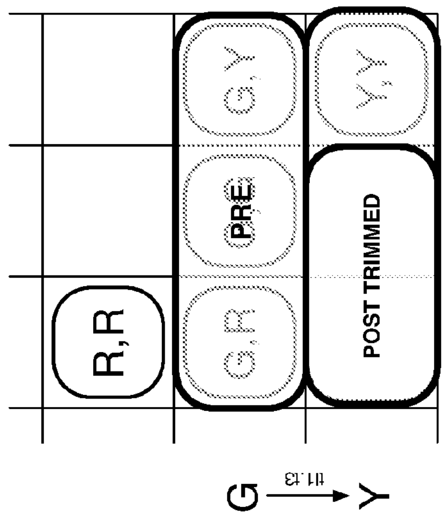


Fig. 18

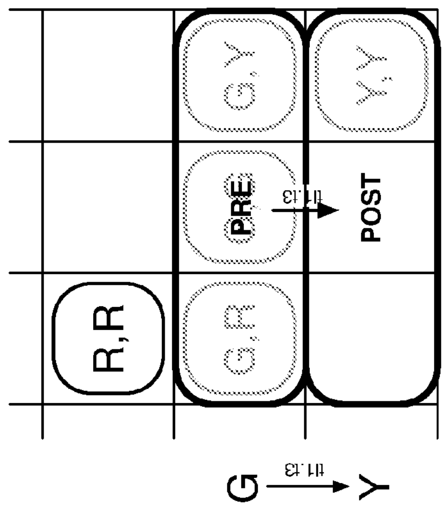


Fig. 17

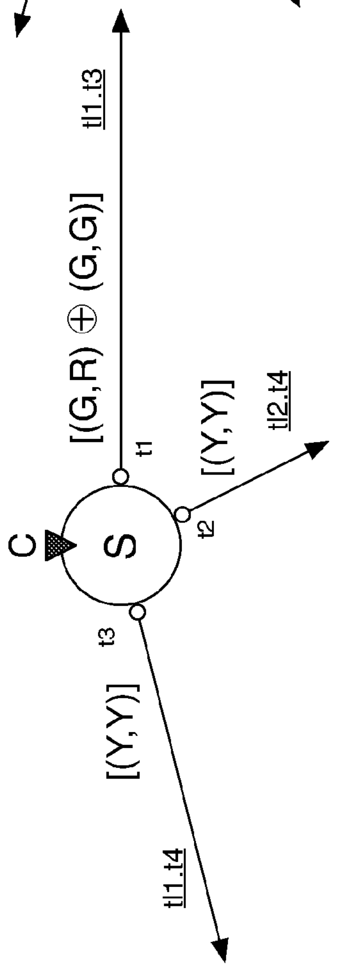

C
S
t1
t2
t3
[(G,R) ⊕ (G,G)]
tl1.t3
[(Y,Y)]
tl2.t4
[(Y,Y)]
tl1.t4

Fig. 20

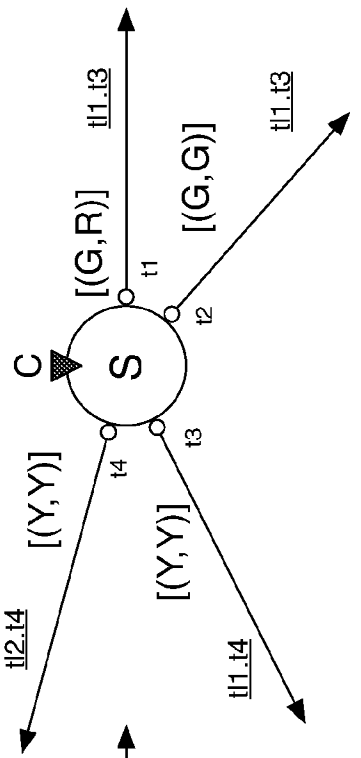
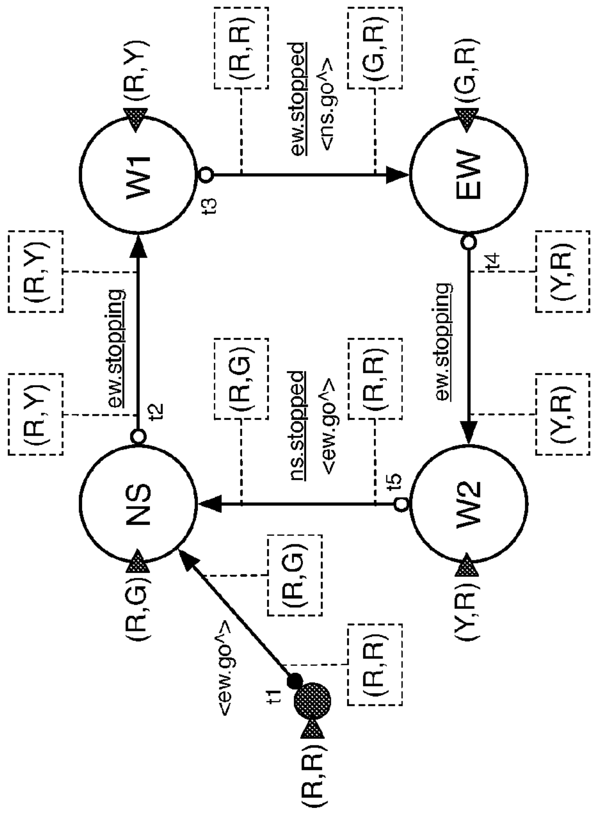
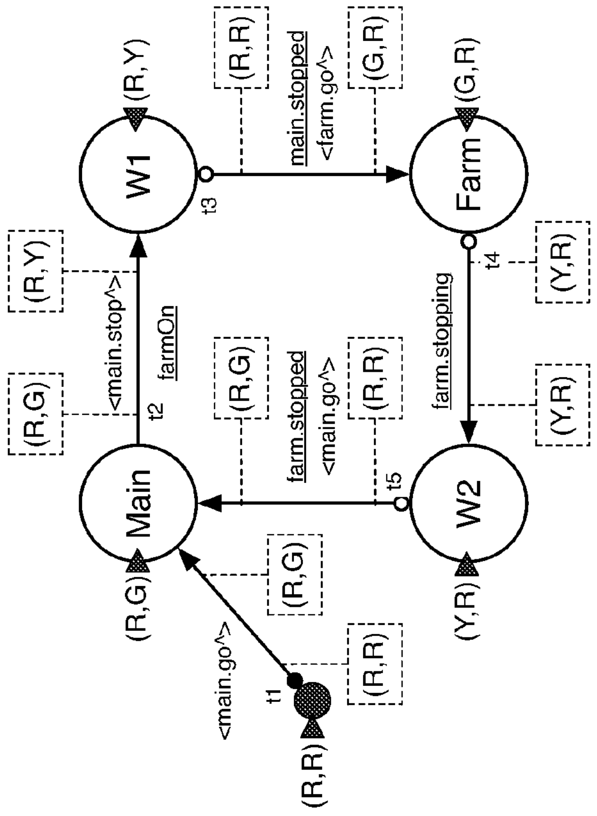

C
S
t1
t2
t3
t4
[(G,R)]
tl1.t3
[(G,G)]
tl1.t3
[(Y,Y)]
tl2.t4
[(Y,Y)]
tl1.t4

Fig. 21



NS
W1
EW
W2
(R,G)
(R,Y)
(G,R)
(Y,R)
(R,R)
t1
t2
t3
t4
t5
<ew.go^>
ew.stopping
ew.stopped
<ns.go^>
ns.stopped
<ew.go^>
(R,Y)
(R,Y)
(R,R)
(G,R)
(R,G)
(R,R)
(Y,R)
(Y,R)
(R,G)
(R,R)

Fig.22



Main
W1
Farm
W2
(R,G)
(R,Y)
(G,R)
(Y,R)
(R,R)
t1
t2
t3
t4
t5
<main.go^>
<main.stop^>
farmOn
main.stopped
<farm.go^>
farm.stopping
farm.stopped
<main.go^>
(R,G)
(R,Y)
(R,R)
(G,R)
(R,G)
(R,R)
(Y,R)
(Y,R)
(R,G)
(R,R)

Fig. 23

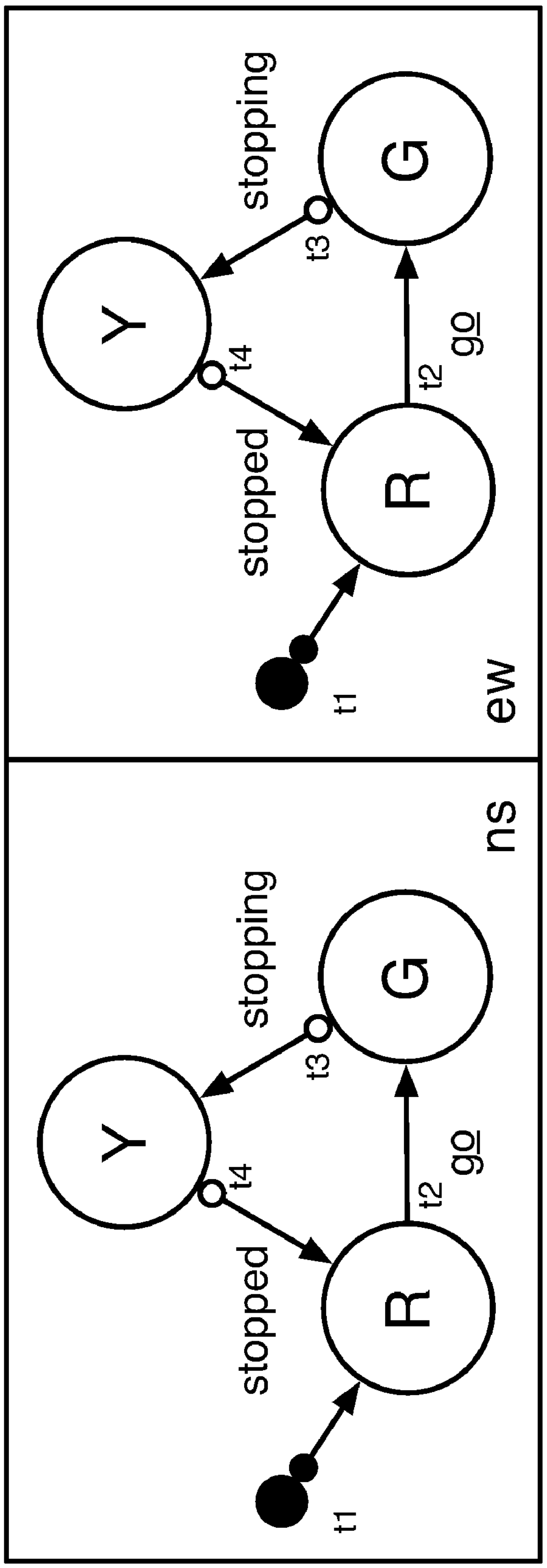

ns
R
Y
G
stopped
stopping
go
t1
t2
t3
t4
ew
R
Y
G
stopped
stopping
go
t1
t2
t3
t4

Fig. 24

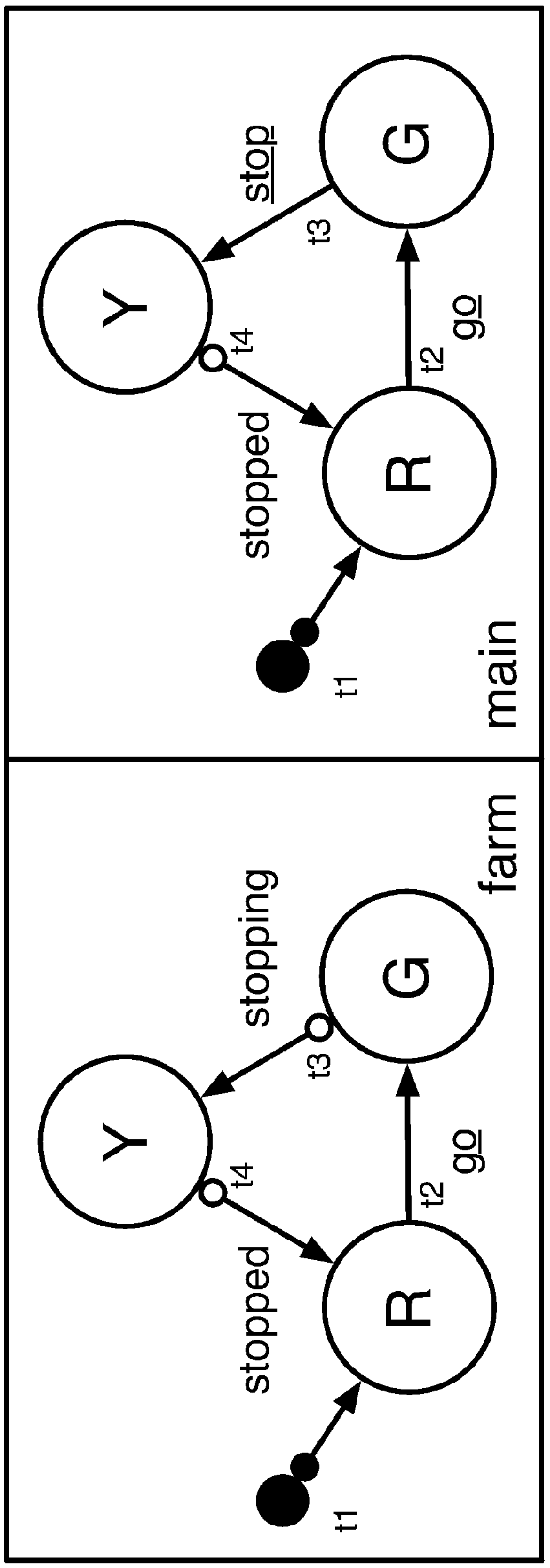

farm
t1
stopped
R
t4
Y
t2
go
t3
G
stopping
main
t1
stopped
R
t4
Y
t2
go
t3
stop
G

Fig. 25

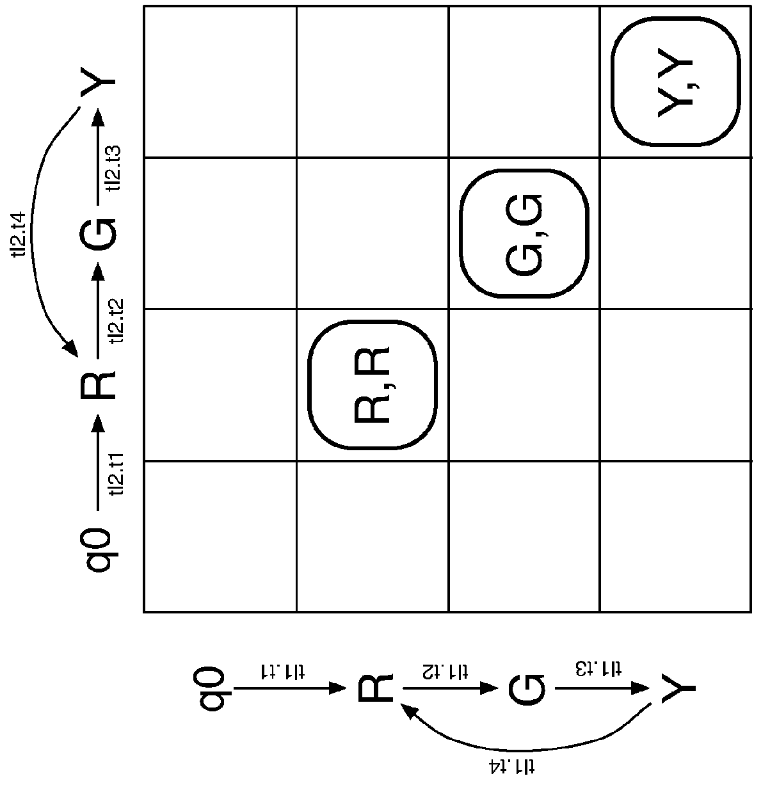

q0
R
G
Y
tl2.t1
tl2.t2
tl2.t3
tl2.t4
tl1.t1
tl1.t2
tl1.t3
tl1.t4
R,R
G,G
Y,Y

Fig. 27

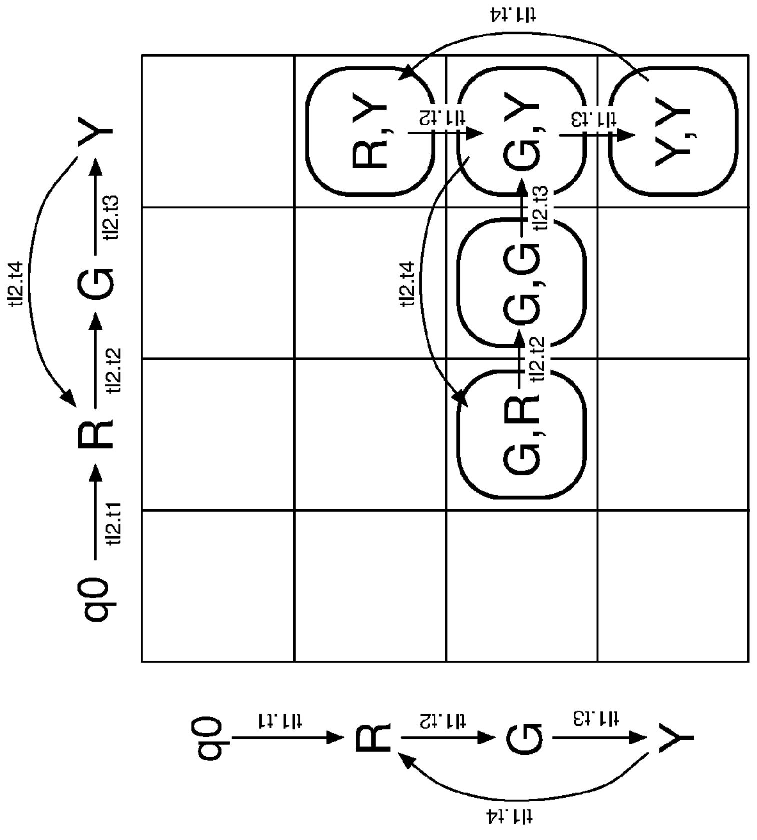

q0
R
G
Y
tl2.t1
tl2.t2
tl2.t3
tl2.t4
tl1.t1
tl1.t2
tl1.t3
tl1.t4
R,Y
G,Y
Y,Y
G,G
G,R

Fig. 26

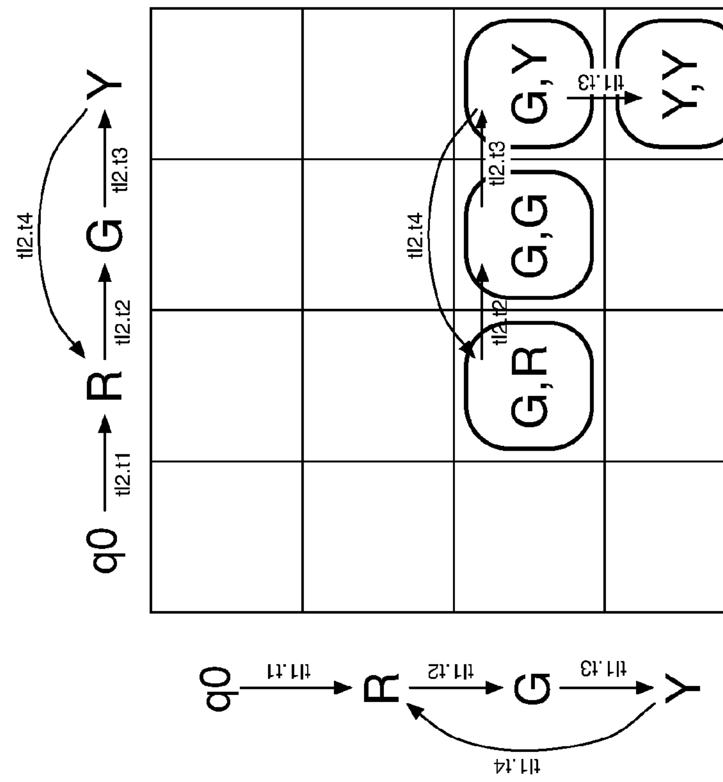

q0
R
G
Y
tl2.t1
tl2.t2
tl2.t3
tl2.t4
tl1.t1
tl1.t2
tl1.t3
tl1.t4
G,R
G,G
G,Y
Y,Y

Fig. 29

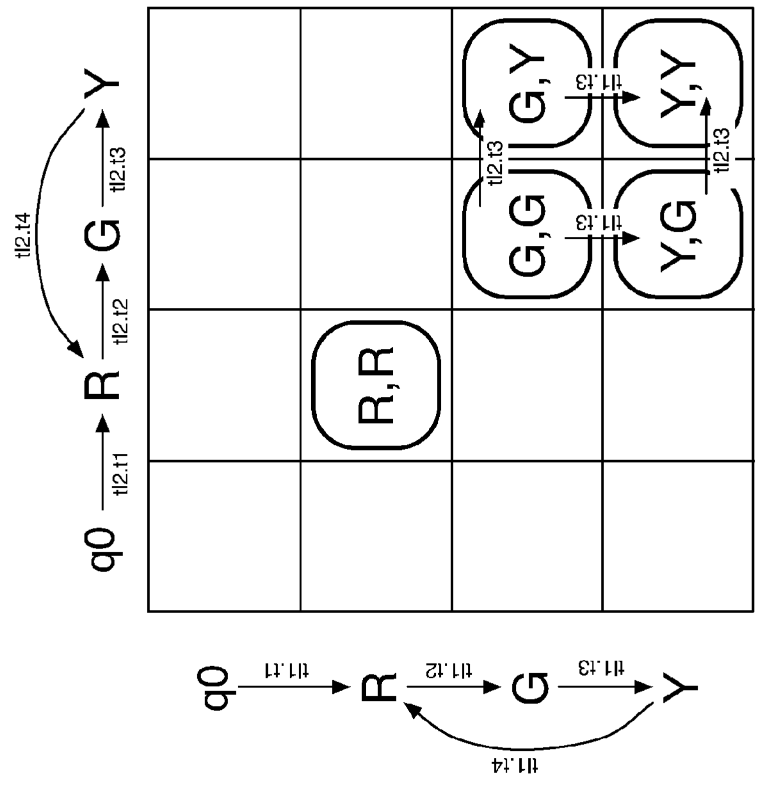

q0
R
G
Y
tl2.t1
tl2.t2
tl2.t3
tl2.t4
tl1.t1
tl1.t2
tl1.t3
tl1.t4
R,R
G,G
G,Y
Y,G
Y,Y

Fig. 28

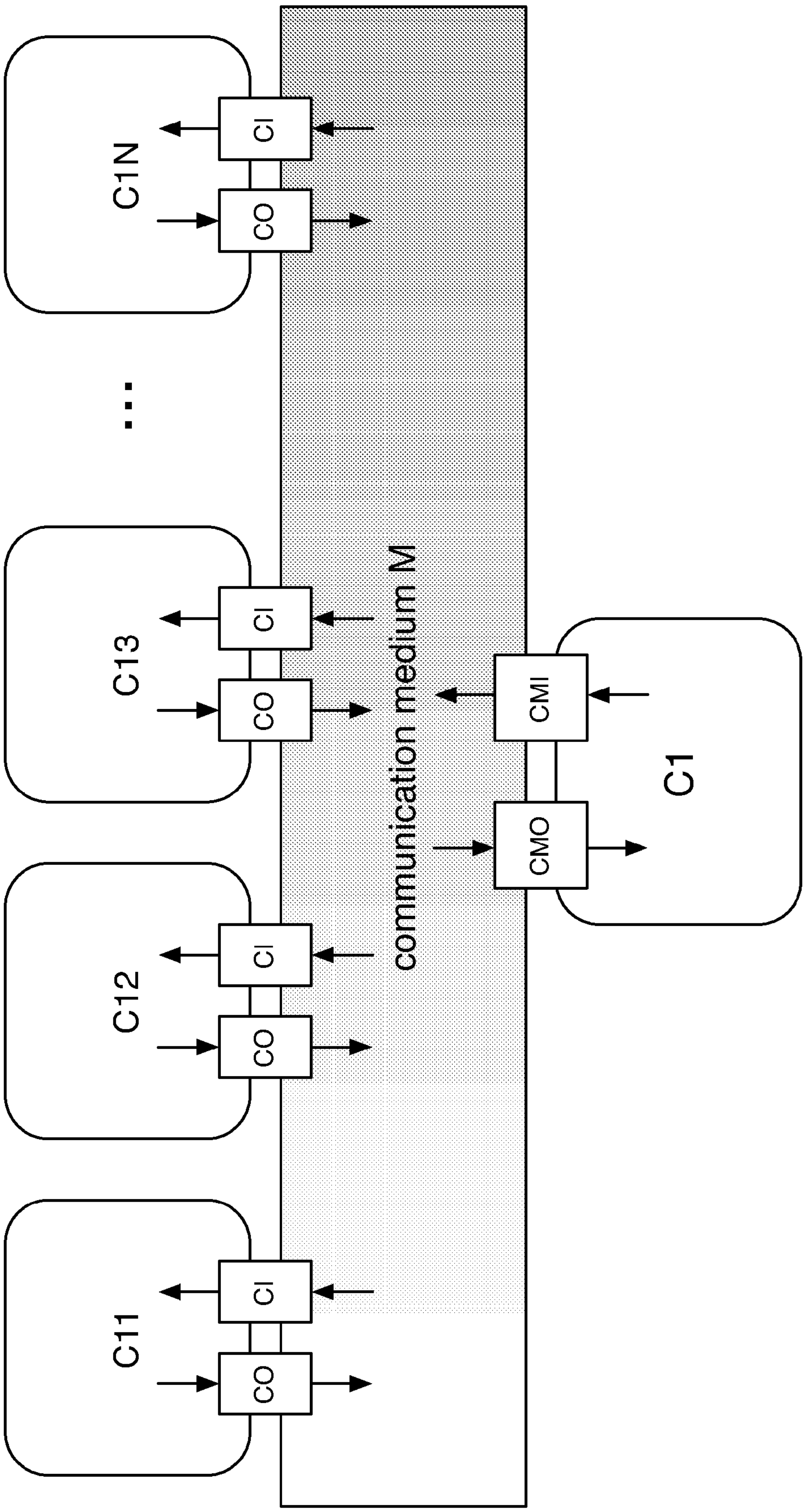

C11
CO
CI
C12
CO
CI
C13
CO
CI
C1N
CO
CI
...
communication medium M
CMI
CMO
C1

Fig. 30

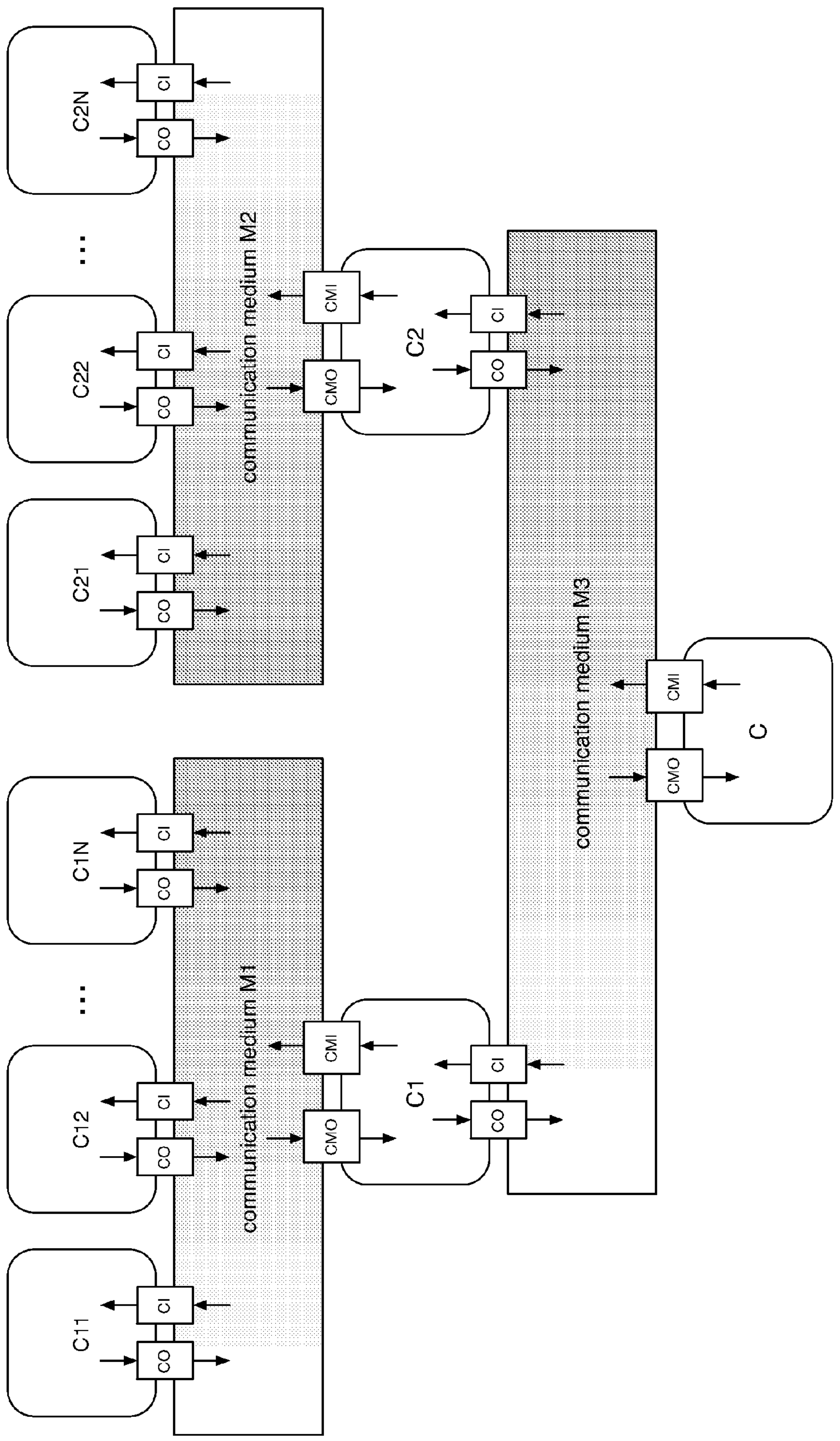

C11
C12
...
C1N
C21
C22
...
C2N
CO
CI
CMO
CMI
C1
C2
C
communication medium M1
communication medium M2
communication medium M3

Fig. 31

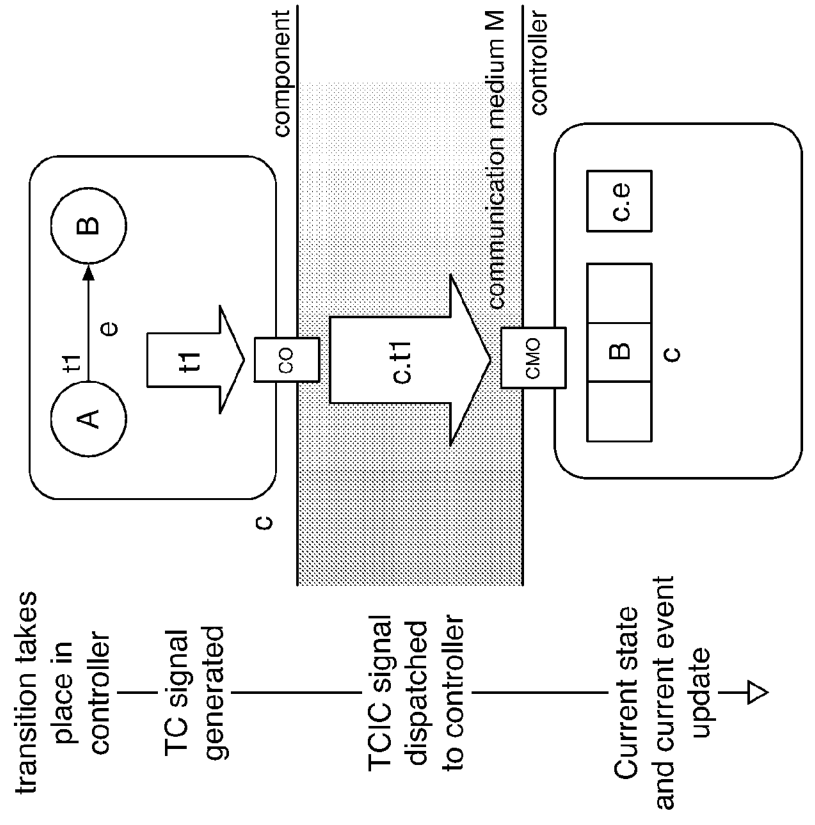

transition takes place in controller
TC signal generated
TCIC signal dispatched to controller
Current state and current event update
c
A
t1
e
B
t1
CO
c.t1
component
communication medium M
controller
CMO
B
c
c.e

Fig. 32

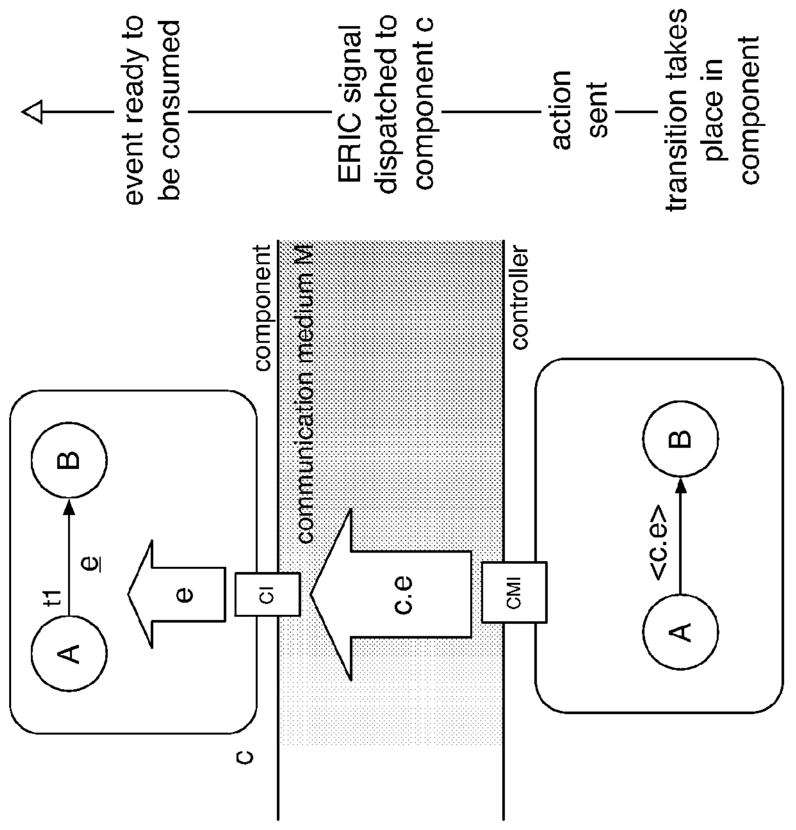
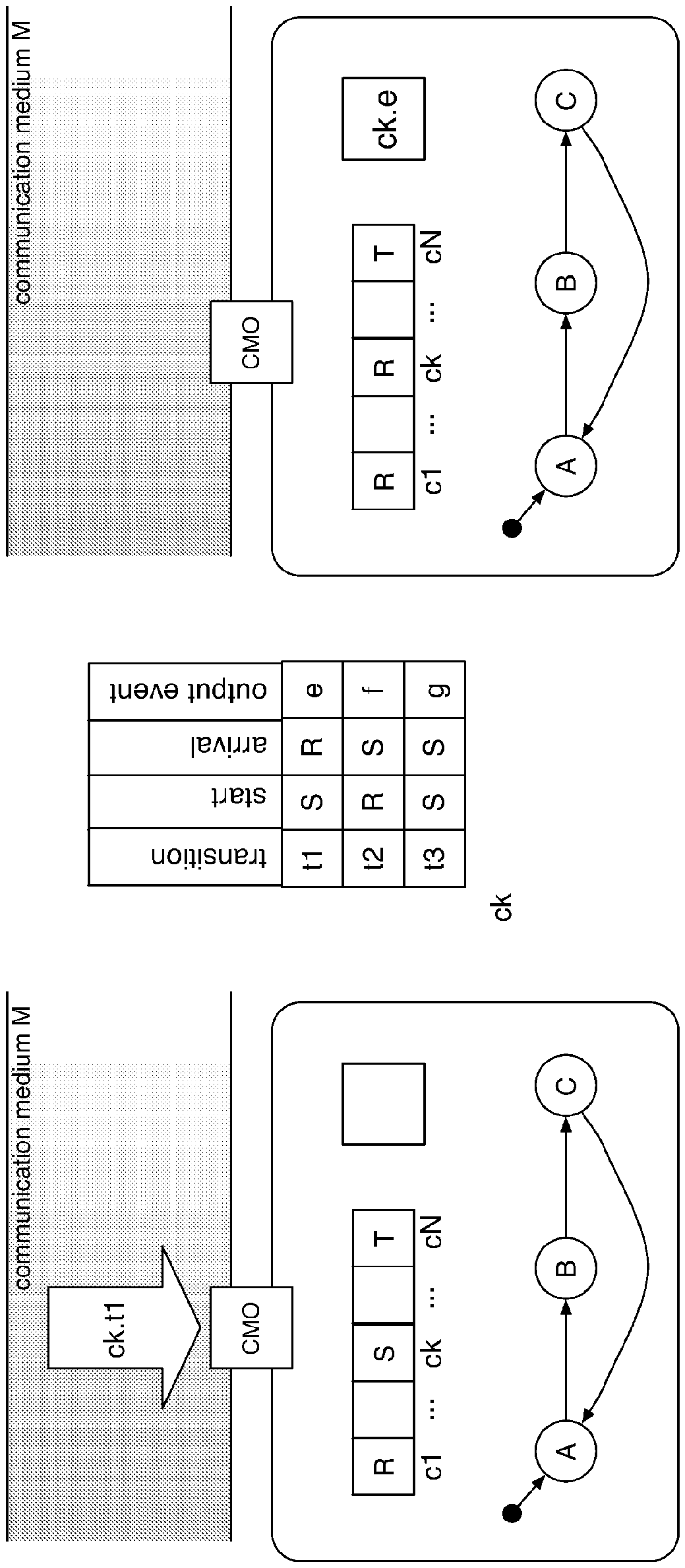

event ready to be consumed
ERIC signal dispatched to component c
action sent
transition takes place in component
c
A
t1
e
B
e
CI
c.e
component
communication medium M
controller
CMI
A
<c.e>
B

Fig. 33



| transition | start | arrival | output event |
|---|---|---|---|
| t1 | S | R | e |
| t2 | R | S | f |
| t3 | S | S | g |

Fig. 34

# METHOD FOR ENSURING SAFETY AND LIVENESS RULES IN A STATE BASED DESIGN

**[0001]** The present invention concerns a method which works upon an abstract operational and structural model of the control of one or more sets of state machines, named assemblages, by means of other state machines, named controllers. According to the model, such controllers may be further grouped into assemblages themselves and be controlled, on their turn, by other controllers, and so on.

**[0002]** The method is based on state constraints, which are propositions about the global state of an assemblage and enforces safety in a state based design, that is it checks that such constraints are always verified, that is when the controller is in a given state the controlled machines in the assemblage do not violate the constraint of such state. It moreover shows how to ensure that a reactive behavior is correctly implemented, that is when the assemblages moves, in an uncontrollable way, to a global state which violates the constraint of the current state of the controller, then there is a transition in the controller that is triggered and move the control out of the violated state. The method enforces also liveness in a state based design, by checking that any part of the constraint of a given state may be reached by the global state of the assemblage.

State machines

**[0003]** State machines as referred to in this invention are used in the control of physical machines. Such physical entities have a behavior, which is a sequence of physical states. In order to observe and control such a behavior, and therefore the machine itself, each of the physical states of the machine is converted into a logical state through a special device named sensor.

**[0004]** Conversely, given a set of logical states, a physical machine may be forced to move to a specific physical state corresponding to a given logical state, through a special device named actuator, that converts logical commands of a state machine into physical commands acting on a physical machine. It is also possible that a physical machine changes its state spontaneously.

**[0005]** The transition among the logical states is referred to as state transition. A transition among logical states in a logical machine corresponds therefore to a transition among physical states in the physical machine.

**[0006]** Sensor and actuators act therefore as an interface between a physical device and a computer, which manipulates indeed logical symbols. Since, as observed, there is a direct and given correspondence among physical and logical states, with related transitions among them, it is possible to use the term state machine to denote both the symbolic behavior of a device, as well as its physical counterpart.

**[0007]** State machines play a twofold role in a control model, since they have to represent both the behavior to be controlled as well as the behavior which exercises control over other state machines. Additionally, the two roles have to coexist in a single state machine, since controllers may be further grouped into assemblages which are controlled on their turn. The behavior of a state machine is represented by a succession of states and state transitions, originating from an initial state $q_0$. At each time the state machine is found in a state, named current state. State transitions take the state machine from one current state to another state and are triggered, that is activated, by either:

**[0008]** a direct, although non mandatory, request coming from a controller state machine;

**[0009]** a reaction to a specific state and behavior observed in one or more controlled state machines;

**[0010]** A state machine consists of an interface, which allows an external controller to observe and control the behaviour of the machine and of an implementation, which allows the machine acting as a controller to observe and control, in turn, the behavior of other state machines through their interfaces.

State Machine Interface

**[0011]** A state machine interface consists of:

**[0012]** a set of states, among which there is a distinguished state, named initial state;

**[0013]** a set of logical symbols, named events, which are partitioned into two sets, respectively named input and output events;

**[0014]** a set of state transitions, which are directed arrows linking states, each state transition comprising:

**[0015]** (a) an optional state transition trigger, named external trigger, which consists of a symbol taken from the input events of the machine;

**[0016]** (b) an optional output symbol, which consists of a symbol taken from the output events of the machine;

**[0017]** The interface of a state machine is therefore formally given by:

**[0018]** an initial state $q_0$,

**[0019]** a finite set of states $Q=\{q_1,q_2 \ldots, q_M\}$;

**[0020]** a finite set of state transitions $T=\{t_1,t_2 \ldots, t_M\}$ which is partitioned into two sets $T_1$ and $T_0$, which will be referred to, respectively, as input and output transitions;

**[0021]** a state transition function $\delta:Q\times T\rightarrow Q$, such that there is a transition t from state S to state T iff $\delta(S,t)=T$;

**[0022]** a finite set of identifiers $E=\{e_1,e_2, \ldots e_K\}$, named events, which is partitioned into two sets $E_1$ and $E_0$, which will be referred to as, respectively, of input and output events;

**[0023]** a transition labelling function event:$T\rightarrow E$. Output transitions must be labelled by output events, and input transitions must be labelled by input events.

**[0024]** It is useful to define the inverse of the transition labelling function trigger$^{-1}$:$E\rightarrow 2^T$, such that, for any $e\in E$, trigger$^{-1}$(e) denotes the set of state transitions which are labelled by event e,

**[0025]** Input transitions are controllable through the interface, that is their activation can be requested by the controller. In any case such a request is not mandatory, that is any transition request by the controller may either succeed or not. Output transitions are instead not controllable through the interface: as such, they will also be called automatic state transitions, in the sense that they happen with no intervention from the controller. We distinguish finally a subfamily of automatic state transitions, that is (automatic) instantaneous state transition, which are taken, as their name suggests, as soon as possible.

**[0026]** By convention, state transitions departing from the initial state $q_0$ have to be automatic instantaneous state transitions and do not have to be labelled by any output event.

## State Machine Assemblages

**[0027]** It is useful to group state machines into assemblages $A=\{c_1, c_2, \ldots c_M\}$, which are finite set of state machines, each univocally identified within the assemblage by the distinct identifiers $c_1, c_2, \ldots c_M$.

**[0028]** State machines within an assemblage have to exhibit a global coordinated behavior. Each state machine within an assemblage makes visible only its interface, hiding other details. We will refer to each of the $c_i \in A$ as assemblage component, or more simply to as component. Components may be additionally partitioned into two classes: asynchronous and synchronous devices. Such a distinction which will have an effect on the behavior of the state machine during its interaction with a controller. Synchrony issues are discussed more thoroughly here below.

## Global State of an Assemblage

**[0029]** Each state machine $C_i$ belonging to an assemblage A will be found, at any time, in a state belonging to its own set of states $q_{c_i} \in Q_{c_i}$ named the current state of $c_i$. The whole set of state machines in the assemblage A will be collectively found, at any time, in a global state $q=(q_{c_1}, q_{c_2}, \ldots q_{c_M})$, with $q_{c_i} \in Q_{c_i}$ for each $C_i \in A$, named the current state of the assemblage A, which in turn belongs to a set $Q_A$ of global states of the assemblage A given by the cartesian product of the sets of states of each device in A: $Q_A = Q_{c_1} \times Q_{c_2} \times \ldots \times Q_{c_M}$.

## Assemblage Commands

**[0030]** Since the behavior of an assemblage has to be controlled and sensed as a whole by means of a controller, we will refer to the state transitions and to the event symbols of its component machines through unique identifiers within the assemblage. We build such a set of assemblage global identifiers by prefixing with the assemblage component name $c_i$ each event symbol $e \in E_{c_i}$ in the event set and each transition name t belonging to the event set of $T_{c_i}$ of $c_i$. A univocally identifiable event is written as c.e, where e belongs to the set of events $E_c$while a univocally identifiable state transition is written as c.t, where t belongs to the set of transitions $E_c$, where c is the identifier of a state machine within the assemblage A.

**[0031]** Both the univocally identifiable event symbol c.e and transition c.t will be referred to as assemblage commands. Commands are additionally classified into the sets $I_A$ and $O_A$, respectively of input and output commands depending whether the original event symbol or transition identifier belongs to the set of, respectively, input or output events and transitions, in the original component machine.

## State Machine Implementation

**[0032]** The implementation details of a state machine consist of additional features associated to state transitions:

- **[0033]** an optional state transition trigger, named internal trigger, which consists of a symbol taken from the output events of a controlled machine;
- **[0034]** an optional guard condition associated to a state transition, consisting of a logical expressions involving the states of the controlled machines;
- **[0035]** an optional list of input events, each belonging to a controlled machine;

**[0036]** The implementation features of a state machine are given formally by:

- **[0037]** a transition labelling function trigger: $T \rightarrow I_A$ which associates input commands to state transitions. Each input command associated to a transition, if any, is named transition internal trigger;
- **[0038]** a transition labelling function actions: $T \rightarrow O^*_A$ which associates a (possibly empty) list of output commands to state transitions;
- **[0039]** a transition labelling function guard: $T \rightarrow \epsilon_A$ which associates a state proposition called transition condition (also transition guard or guard condition) to a state transition. When there is no state proposition associated to a state transition, it means that the state proposition $ANY_A$ (which will be shown to be always true) is by default associated to the state transition.

**[0040]** A state machine description may be succinctly written by a tuple:

$$S = \langle q_0 Q, T, \delta, E, \text{event}, I, O, \text{trigger}, \text{actions}, \text{guard} \rangle$$

**[0041]** An object of the present invention is to provide a method for ensuring that an assemblage of state machines does not reach a global configuration of states which may be harmful.

**[0042]** According to the present invention there is provided a method for controlling a physical machine or an assemblage of physical machines for ensuring safety and liveness rules in a state based design of said physical machine or assemblage of physical machines, characterized in that it comprises associating at least one logical state to at least one physical state said physical machine or assemblage of physical machines may assume, providing state constraints for said logical states, checking that a physical state assumed by said physical machine or assemblage of physical machines is associated to a logical state complying with said state constraints.

**[0043]** The invention will now be described, only by way of non-limitative examples, with reference to the attached drawings in which:

**[0044]** FIG. **1** shows an example a state machine, i.e a device, having two states, states On and Off, for instance a lamp, whose behavior can be totally controlled by means of input transitions t**2** and t**3** and input events on and off;

**[0045]** FIG. **2** shows a semiautomatic device, for instance a lamp like that of FIG. **1**, whose behavior can be partially controlled by means of an input transition t**2** labelled by an input event on. Transition t**3** will be instead taken automatically by the machine when in state On and the output event off generated;

**[0046]** FIG. **3** shows another semiautomatic device, a timer, which can be used in order to implement time intervals in the behavior of more complex machines;

**[0047]** FIG. **4** shows another version of the device of FIG. **3**, offering more input events and transitions;

**[0048]** FIGS. **5** and **6** show state machines that are two versions of an implementation of a traffic light having the canonical three lamps, Red, Green and Yellow;

**[0049]** FIG. **7** shows a different implementation of a traffic light like that of FIGS. **5** and **6**;

**[0050]** FIG. **8** shows an interface of a controller which coordinates the assemblage of two traffic lights of the same kind, whose interface has been shown in FIG. **6**;

**[0051]** FIG. **9** shows the interface of a controller which coordinates the assemblage of two traffic lights of different kind, whose interface has been shown in FIG. **6**;

**[0052]** FIG. **10** shows a version of a traffic light controller which implements a night mode flashing feature required by the controller of FIG. **8**;

**[0053]** FIG. **11**, **12**, **13** show the implementation of two traffic light controllers given the same assemblage of state machines;

**[0054]** FIG. **14** illustrates pre and post condition semantics of each transition in the implementation of a traffic light controller shown in FIG. **11**;

**[0055]** FIGS. **15** and **16** illustrates feasible state transitions associated to a fictional state proposition and the exit zones associated to a same condition;

**[0056]** FIGS. **17**, **18** and **19** illustrates a trimming process applied to the fictional state proposition of FIGS. **15** and **16**;

**[0057]** FIGS. **20** and **21** illustrate exit zones associated to a state constraint of a state of a controller;

**[0058]** FIGS. **22** to **25** illustrate an implementation and safety verification of cross road controllers;

**[0059]** FIGS. **26** to **29** illustrate four examples of assemblage propositions;

**[0060]** FIG. **30** illustrates a single level architecture involving an assemblage of component state machines;

**[0061]** FIG. **31** illustrates a multilevel architecture involving an assemblage of component and controller state machines;

**[0062]** FIG. **32** illustrates a communication flow controller-component;

**[0063]** FIG. **33** illustrates a communication flow component-controller;

**[0064]** FIG. **34** illustrate a current state array and incoming event computation.

**[0065]** FIGS. **1** to **10** illustrate examples of state machine interfaces.

**[0066]** In FIGS. **1** to **10** the following graphical conventions have been adopted: state machine interfaces are drawn as directed graphs, where input transitions are drawn as arrows, output automatic transitions are drawn as arrows which have a small white circle in correspondence of the transition initial state and an arrow in correspondence of the arrival state; finally, instantaneous state transition are distinguished from the other automatic transition by painting black the small circle.

**[0067]** Input events are underlined. State transition identifiers are drawn close to the beginning of the arrow, while input and output events are drawn instead near the middle of the arrow. The initial state $q_0$ is finally drawn as a black dot.

**[0068]** The meaning of boolean algebra operators and expressions used in the rest of this description is explained at paragraph "The boolean algebra of assemblage expressions" and subsequent paragraphs here below.

**[0069]** FIGS. **1** to **4** illustrate examples of state machines to which the method according to the invention may be applied.

**[0070]** FIG. **1** shows an example a state machine, i.e a device, having two states, states On and Off, for instance a lamp, whose behavior can be totally controlled by means of input transitions t**2** and t**3** and input events on and off.

**[0071]** FIG. **2** shows a semiautomatic device, for instance a lamp like that of FIG. **1**, whose behavior can be partially controlled by means of an input transition t**2** labelled by an input event on. Transition t**3** will be instead taken automatically by the machine when the machine is in state On and the output event off is generated.

**[0072]** FIG. **3** shows another semiautomatic device, a timer, which can be used in order to implement time intervals in the behavior of more complex machines. A timer starts and rests in the TOut state, until it is forced to move to the TIn state by the controllable (input) transition t**2** on the receipt of the input event Set. It then remains in the TIn state for a definite and fixed amount of time, that is until the automatic (output) transition t**3** is taken and the corresponding (output) event tout is generated. A self loop is provided by the input transition t**4** which starts and ends in the state TIn, meaning that, when the timer is in state TIn and the input event is received by the machine, then the measurement of the time interval is restarted.

**[0073]** FIG. **4** shows another version of the device of FIG. **3**, offering more input events and transitions, namely setT**1** and setT**2**, each meaning that a different time interval has to pass before the timer returns to the timeout state.

**[0074]** FIGS. **5** and **6** show state machines that are two versions of an implementation of a traffic light having the canonical three lamps, Red, Green and Yellow.

**[0075]** Referring to FIG. **5**, each of said lamps, once lit, corresponds to the three states of the state machine, respectively state R, G and Y. Both of the two versions of the traffic light controller have an input/controllable transition t**2** labelled by the input event go and two automatic output transitions, which happen automatically. Such a device is then controlled by a go event, after that it cycles automatically (by taking automatic transitions t**3** and t**4**) through the other two states until it returns to the R state.

**[0076]** FIG. **6** shows a more readable version of the traffic light controller, which emits output events stopping and stopped when the two automatic transitions are taken.

**[0077]** FIG. **7** illustrate a different version of the traffic light controller since it rests in the R state until a stop command is issued through the input event stop.

**[0078]** FIG. **8** shows the interface of a controller which coordinates the assemblage of two traffic lights of the same kind, whose interface has been shown in FIG. **6**.

**[0079]** The two traffic lights are placed on the crossing of two roads, one running from North to South and the other from East to West. The controller has four states: state NS, which means that the road traffic is enabled from North to South and vice versa, state W**1**, which means that traffic is being stopped in such a road, state EW, which means that the road traffic is enabled from East to West and vice versa and finally state W**2**, which means, as in the other case, that traffic is being stopped in such a road. The basic cycles happens automatically. A different working mode, corresponding to state Night (both the roads have yellow flashing light), may be reached by issuing a command through the input event night when the state machine is in any state of the basic cycle. From the night mode it is possible to restart the basic cycle starting from the NS state, by issuing a command through the input event day.

**[0080]** FIG. **9** shows the interface of a controller which coordinates the assemblage of two traffic lights of different kind, whose interface has been shown in FIGS. **6** and **7**, which are placed, respectively, on a main road ad on secondary farm road.

**[0081]** The controller starts on the Main state, meaning that the traffic on the main road is enabled to flow and the farm road is stopped, and it rests on such state until a command is issued to the controller through the event farm. The controller then moves automatically to state W**1**, where the main road is stopping and the farm road is still blocked, then to the state Farm, where the traffic on the farm road is enabled to flow and the main road is stopped. After some fixed time interval the controller moves automatically to state VV**1**, where the farm road is stopping and the main road is still blocked, and to the Main state again, where it rests waiting for the next command.

**[0082]** FIG. **10** shows finally a version of the traffic light controller which implements the night mode flashing feature required by the previous cross road controller of FIG. **8**. Such a feature is realized by two additional states, N and B, respectively having the yellow lamp lit and no lamp lit, which alternate themselves in order to obtain the flashing mode. By the input events night and day it is possible to switch from the regular cycle mode to the flashing mode, and back.

**[0083]** FIGS. **11** to **13** shows the implementation of the two controller state machines whose interface has been shown in FIGS. **6** and **7** by using an assemblage of four synchronous state machines, that is three lamp state machines, identified by I**1**, I**2** and I**3** whose interface is depicted in FIG. **1** and the timer state machine, identified by t, whose interface is depicted in FIG. **4**.

**[0084]** The following graphical conventions are adopted: state machines are drawn as directed graphs and retain all the details of the interface.

**[0085]** Guard conditions are enclosed in square brackets and drawn near the beginning of the arrow, internal triggers are underlined and command lists are enclosed in angular brackets. The default state proposition guard $\text{ANY}_A$ is by convention not drawn.

**[0086]** Actions in the action list which are directed towards assemblage synchronous components are distinguished by postponing an upper arrow to the command.

**[0087]** Finally, when it is shown, the controlled assemblage is drawn by reporting the component state machines above a dotted line, each identified by the assemblage identifier and separated by a solid line (as in FIGS. **11** to **13**).

## State Transition Typologies

**[0088]** Internal and external features of state transitions can not be arbitrarily mixed. The following rules must be followed, dictated by the rationale that it is not possible to specify both an internal trigger and an external trigger for a single state transition.

**[0089]** State transitions can be thus classified into three families:

**[0090]** internally triggerable state transitions, which have an internal trigger, instantaneous state transitions, which have no triggers at all and externally triggerable state transitions, which have an external trigger; such a classification very easily maps to the interface distinction amongst controllable and automatic (non controllable) transitions:

**[0091]** Internally triggerable state transitions are those which react to changes in the controlled machines. An example is given in FIG. **11** by transitions $t_3$ and $t_4$ and in FIG. **12** by transition $t_4$. Internally triggerable state transitions give rise to automatic transition when the state machine is seen through its interface.

**[0092]** Instantaneous state transitions are those which do not specify any trigger, hence are taken as soon as their guard condition becomes true. An example is given in FIGS. **11** and **12** by transition $t_1$, which is triggered instantaneously and in any case it is guarded by the state proposition $\text{ANY}_A$ which is always true and is not drawn by convention. They give rise to automatic instantaneous transition when the state machine is seen through its interface.

**[0093]** Externally triggerable state transitions are those which react to commands sent to the interface of the machines. An example is given in FIG. **11** by transition $t_2$ and in FIG. **12** by transitions $t_2$ and $t_3$.

**[0094]** By comparing the interface of the controller in FIG. **2** and its implementation in FIGS. **11** to **13** it is clear that a state machine interface can be obtained from its implementation; vice versa it is possible to design the behavior by specifying its interface and then specify its implementation accordingly.

**[0095]** The method that is the object of the present invention will now be described in detail.

## Verification Method

**[0096]** It is important to ensure that an assemblage of state machines does not reach a global configuration of states which it may be harmful.

**[0097]** The problem regards in general having full knowledge about the configuration of states which an assemblage of state machine can assume. According to the method of the present invention it is possible to design a controller in such a way that the controlled assemblage is always under such a control. By this method it is possible both to ensure that an assemblage of state machine does not reach a forbidden configuration as well as that any allowed configuration of states may be reached by the assemblage.

## State Constraints

**[0098]** The method according to the invention works by first associating to each state S of the controller an assemblage state proposition, which is a formula which denotes the exact set of states that the modeler wants to be assumed by the assemblage when the controller is in a state S. Such a state proposition is named state constraint and denoted by vinc(S).

## State Transition Precondition Semantics

**[0099]** State transitions denote a set of global states in which the assemblage must be found in order for the state transition to be taken. Such set of states is denoted by a state proposition which is called precondition semantics. It can be computed according to the features of the transition and to the different typologies in which it may be classified:

**[0100]** externally triggerable transition: let t be a transition originating from a state S of the controller constrained by state proposition vinC(S). Necessary condition for state transition t to be executed is that its guard condition guard(t) holds, that is, the assemblage must be in state q such that q satisfies guard(t). Since, at the same time, the state transition t originates from S we have that q has necessarily to be contained within the maximal set of states the assemblage may assume when the controller is in S, that is q satisfies vinC(S). The precondition semantics pre(.) of state transition t, guarded by state proposition c, is then given by the intersection of the transition guard and of the state constraint of the state from which the transition originates:

$$\text{pre}(t)=\text{guard}(t)\odot\text{vinc}(S)$$

**[0101]** internally triggerable transition: let t be a transition originating from a state S of the controller constrained by state proposition vinC(S) and triggered by transition t in component c. As in the case above, necessary condition for state transition t to be executed is that its guard condition guard(t) holds, that is, the assemblage must be in state q such that q satisfies guard(t). Since, at the same time, the state transition t originates from S we have that q has necessarily to be contained within the maximal set of states the assemblage may assume when the controller is in S, that is q satisfies vinc(S) after either a triggering transition c.t happened within the assemblage or the triggering command c.e is received from the assemblage. Two cases then arise:

**[0102]** (a) In the former case we know that such a state transition happened and therefore the state proposition vinc(S) has been transformed into transf(vinc(S),c,t), where transf(vinc(S),c,t) is a state proposition p' which means "the assemblage was in a state p and a state transition t happened. The precondition semantics pre(.) is therefore in this case given by:

$$\mathrm{pre}(t)=\mathrm{guard}(t)\odot\mathrm{transf}(\mathrm{vinc}(S),c,t) \tag{2}$$

**[0103]** (b) In the latter case we know that a command of the kind c.e may have triggered any of the transitions which have e as input trigger. We have therefore that the state proposition vinc(S) has been transformed into transfE(vinc(S),c.e), where

$$transfE(vinc(S),\, c.e) = \bigoplus_{t\in T} transf(vinc(S),\, c,\, t),$$

T being the set of transitions that are triggerable by event e in component c, in symbols T=trigger$^{-1}$ (e).

**[0104]** The precondition semantics pre(.) is therefore in this case given by:

$$\mathrm{pre}(t)=\mathrm{guard}(t)\odot\mathrm{transfE}(\mathrm{vinc}(S),c.e) \tag{3}$$

**[0105]** instantaneous transitions: as in the case of externally triggerable transitions, the precondition is then given by the intersection of the transition guard and of the state constraint of the state from which the transition originates, hence the precondition semantics pre(.) is given by Equation 1 above.

**[0106]** It may be observed that a state transition such that pre(t)=NONE$_A$ will never be executed, hence it may be removed from the state diagram. It may be further observed that, necessary condition in order to have at most one external transition chosen for execution on the receipt of a triggering event e, if T(e,S)is the set of external transitions having S as starting state and e as trigger, for any $t_1,t_2\in T(e,S)$ than pre($t_1$) and pre($t_2$) must be disjoint. We observe, finally, that, in order to have exactly one external transition taken on the receipt of e, the set of the transition preconditions in T(e,S) must form a partition of C(S).

EXAMPLE 1

Calculation of Precondition Semantics

**[0107]** In this example the semantic precondition of the transitions of the two versions the traffic light controller implementation shown in FIGS. **11** and **12** are calculated.

**[0108]** 1. First controller (FIG. **11**):

**[0109]** A constraint is first assigned to each of the four states of the controller:

(a) $\mathrm{vinc}(q_0)=\mathrm{ON}_{=}^{I1}\odot\mathrm{Off}_{=}^{I2}\odot\mathrm{Off}_{=}^{I3}\odot\mathrm{TOut}_{=}^{t}$

(b) $\mathrm{vinc}(R)=\mathrm{On}_{=}^{I1}\odot\mathrm{Off}_{=}^{I2}\odot\mathrm{Off}_{=}^{I3}\odot\mathrm{TOut}_{=}^{t}$

(c) $\mathrm{vinc}(G)=\mathrm{Off}_{=}^{I1}\odot\mathrm{Off}_{=}^{I2}\odot\mathrm{On}_{=}^{I3}\odot\mathrm{TIn}_{=}^{t}$

(d) $\mathrm{vinc}(Y)=\mathrm{Off}_{=}^{I1}\odot\mathrm{On}_{=}^{I2}\odot\mathrm{Off}_{=}^{I3}\odot\mathrm{TIn}_{=}^{t}$

**[0110]** We have that $t_1$ is an instantaneous transition starting from $Q_0$: we have therefore to compute its precondition according to Equation (1):

$\mathrm{pre}(t_1)=\mathrm{vinc}(q_0)\odot\mathrm{guard}\ (t_0)$
$=\mathrm{On}_{=}^{I1}\odot\mathrm{Off}_{=}^{I2}\odot\mathrm{Off}_{=}^{I3}\odot\mathrm{TOut}_{=}^{t}$

**[0111]** We have that $t_2$ is an externally triggered transition, hence its precondition is again calculated through Equation (1):

$\mathrm{pre}(t_2)=\mathrm{vinc}(R)\odot\mathrm{guard}\ (t_0)$
$=\mathrm{On}_{=}^{I1}\odot\mathrm{Off}_{=}^{I2}\odot\mathrm{Off}_{=}^{I3}\odot T\mathrm{Out}_{=}^{t}$

**[0112]** The next three transitions are instead internally triggered by an assemblage state transition, hence, it is necessary to calculate the transformation induced on the respective starting state constraint by such transition. We have then that their precondition semantics is therefore given by Equation (2):

$\mathrm{pre}(t_3)=\mathrm{transf}(\mathrm{vinc}(G),t,\mathrm{tout})\odot\mathrm{guard}(t_3)$
$=\mathrm{Off}_{=}^{I1}\odot\mathrm{Off}_{=}^{I2}\odot\mathrm{On}_{=}^{I3}\odot T\mathrm{Out}_{=}^{t}$

$\mathrm{pre}(t_4)=\mathrm{transf}(\mathrm{vinc}(Y),t,\mathrm{tout})\odot\mathrm{guard}\ (t_3)$
$=\mathrm{Off}_{=}^{I1}\odot\mathrm{On}_{=}^{I2}\odot\mathrm{Off}_{=}^{I3}\odot T\mathrm{Out}_{=}^{t}$

**[0113]** 2. Second controller (FIG. **12**)

**[0114]** We assign first a constraint to each of the four states of the controller:

(a) $\mathrm{vinc}(q_0)=\mathrm{On}_{=}^{I1}\odot\mathrm{Off}_{=}^{I2}\odot\mathrm{Off}_{=}^{I3}\odot T\mathrm{Out}_{=}^{t}$

(b) $\mathrm{vinc}(R)=\mathrm{On}_{=}^{I1}\odot\mathrm{Off}_{=}^{I2}\odot\mathrm{Off}_{=}^{I3}\odot T\mathrm{Out}_{=}^{t}$

(c) $\mathrm{vinc}(G)=\mathrm{Off}_{=}^{I1}\odot\mathrm{Off}_{=}^{I2}\odot\mathrm{On}_{=}^{I3}\odot T\mathrm{Out}_{=}^{t}$

(d) $\mathrm{vinc}(Y)=\mathrm{Off}_{=}^{I1}\odot\mathrm{On}_{=}^{I2}\odot\mathrm{Off}_{=}^{I3}\odot T\mathrm{In}_{=}^{t}$

**[0115]** We have then that $t_1$ is an instantaneous transition starting from $q_0$: we have therefore to compute its precondition according to Equation (1):

$\mathrm{pre}(t_1)=\mathrm{vinc}(q_0)\odot\mathrm{guard}(t_0)$
$=\mathrm{On}_{=}^{I1}\odot\mathrm{Off}_{=}^{I2}\odot\mathrm{Off}_{=}^{I3}\odot T\mathrm{Out}_{=}^{t}$

**[0116]** We have that, in this case, both $t_2$ and $t_3$ are externally triggered transition, hence its precondition is again calculated through Equation (1):

$\mathrm{pre}(t_2)=\mathrm{vinc}(R)\odot\mathrm{guard}(t_0)$
$=\mathrm{On}_{=}^{I1}\odot\mathrm{Off}_{=}^{I2}\odot\mathrm{Off}_{=}^{I3}\odot T\mathrm{Out}_{=}^{t}$

$\mathrm{pre}(t_3)=\mathrm{vinc}(G)\odot\mathrm{guard}(t_0)=\mathrm{Off}_{=}^{I1}\odot\mathrm{Off}$
$_{=}^{I2}\odot\mathrm{On}_{=}^{I3}\odot T\mathrm{Out}_{=}^{t}$

**[0117]** The last transition is instead internally triggered by an assemblage state transition, hence, it is necessary to calculate the transformation induced on the respective starting state constraint by such transition. We have then that its precondition semantics is given by Equation (2):

$$\mathrm{pre}(t_4)=\mathrm{transf}(\mathrm{vinc}(Y),t,\mathrm{tout})\odot \mathrm{guard}(t_3)$$
$$=\mathrm{Off}_{=}^{I1}\odot \mathrm{On}_{=}^{I2}\odot \mathrm{Off}_{=}^{I3}\odot T\mathrm{Out}_{=}^{t}$$

## State Transition Postcondition Semantics

**[0118]** Let t be a transition in the controller having a list of associated actions l=actions(t). When the transitions is started, the assemblage is in any global state Q which satisfies pre(t). When the state transition is executed, each action directed towards a synchronous component of the assemblage modifies such a state q before the state transition ends its execution. Let Q' be the state of the assemblage after the last action is executed. When the state transition is terminated the assemblage is therefore found in a global state q', which satisfies the state proposition originating from the precondition pre(t) of the transition transformed by the occurrence of the list of actions, let it be called postcondition semantics post(,) then given by:

$$\mathrm{post}(t)=\mathrm{transfL}(\mathrm{pre}(t),l_s) \tag{4}$$

**[0119]** Where $l_s$ is the sublist of l containing only actions directed towards synchronous components and transfL(•,•) is the function which transforms a state proposition according to a list of actions, which will be defined later.

## EXAMPLE 2

### Calculation of Postcondition Semantics

**[0120]** Referring now to FIG. **14**, we calculate the postcondition semantics of the four transitions of the first traffic light controller example of FIG. **11**; in all cases we simply transform the precondition semantics previously calculated in Example 1 by the operator transfL (•,•) given the list of commands that label each transition. We have therefore that:

post $(t_1)=\mathrm{transfL}(\mathrm{pre}(t),\langle\rangle)=\mathrm{On}_{=}^{I1}\odot \mathrm{Off}_{=}^{I2}\odot \mathrm{Off}_{=}^{I3}\odot \mathrm{TOut}_{=}^{t}$ where the precondition semantics is: $\mathrm{pre}(t_1)=\mathrm{On}_{=}^{I1}\odot \mathrm{Off}_{=}^{I2}\odot \mathrm{Off}_{=}^{I3}\odot \mathrm{TOut}_{=}^{t}$ In other words, since state transition $t_1$ is labelled with no commands, post and precondition semantics are the same.

$$\mathrm{post}\ (t_2)=\mathrm{transfL}(\mathrm{pre}(t_2),\langle 1.\mathrm{off}\uparrow,I3.\mathrm{on}\uparrow,t.\mathrm{set}T1|$$
$$\rangle)=\mathrm{Off}_{=}^{I1}\odot \mathrm{Off}_{=}^{I2}\odot \mathrm{On}_{=}^{I3}\odot T\mathrm{In}_{=}^{t}$$

where $\mathrm{pre}(t_2)=\mathrm{On}_{=}^{I1}\odot \mathrm{Off}_{=}^{I2}\odot \mathrm{Off}_{=}^{I3}\odot \mathrm{TOut}_{=}^{t}$

**[0121]** In other words the effect of the commands I**1**.off↑, I**3**.on↑ and t.SetT**1**↑is, respectively, to turn off lamp I**1**, to turn on lamp I**3** and to set the timer t**1** to state Tin. Since the three commands are synchronous, their effect is achieved before the end of the controller transition. We calculate accordingly the semantics of the last two state transitions:

$$\mathrm{post}(t_3)=\mathrm{transfL}(\mathrm{pre}(t_3),\langle 3.\mathrm{off}\uparrow,\mathrm{I}\mathbf{2}.\mathrm{on}\uparrow,\mathrm{t.set\ T}\mathbf{2}\uparrow$$
$$\rangle)=\mathrm{Off}_{=}^{I1}\odot \mathrm{On}_{=}^{I2}\odot \mathrm{Off}_{=}^{I3}\odot T\mathrm{In}_{=}^{t}$$

where $\mathrm{pre}(t_3)=\mathrm{Off}_{=}^{I1}\odot \mathrm{Off}_{=}^{I2}\odot \mathrm{On}_{=}^{I3}\odot T\mathrm{Out}_{=}^{t}$;

$$\mathrm{post}(t_4)=\mathrm{transfL}(\mathrm{pre}(t_4),\langle I1.\mathrm{on}\uparrow,I2.\mathrm{off}\uparrow$$
$$\rangle)=\mathrm{On}_{=}^{I1}\odot \mathrm{Off}_{=}^{I2}\odot \mathrm{Off}_{=}^{I3}\odot T\mathrm{out}_{=}^{1}\ \text{where}$$

$$\mathrm{pre}(t_4)=\mathrm{Off}_{=}^{I1}\odot \mathrm{On}_{=}^{I2}\odot \mathrm{Off}_{=}^{I3}\odot T\mathrm{Out}_{=}^{t}.$$

## State Transition Semantic Safety

**[0122]** A state transition t ending in a state T of the controller is said to be semantically safe iff:

$$\mathrm{post}(t)\preceq\epsilon_A \mathrm{vinc}(t) \tag{8}$$

## EXAMPLE 3

**[0123]** It can be easily checked that in all four cases the postcondition semantics for the four transitions of the example presented in FIGS. **11** to **13** satisfy Equation above. Hence the implementation of the state transitions is safe.

## Feasible Transitions Associated to a State Proposition

**[0124]** The feasible transitions associated to a state proposition C are the set of transitions that can be taken by the component machines of the assemblage when the system is in any global state that satisfies C and can be found by Algorithm 1 shown here below, which examines all the state transitions in all the state machine of the assemblage, determines for each of such transitions the respective starting state, then checks the proposition which states that the assemblage is in such starting state and at the same time in C: if such proposition is true the transition is added to the set of feasible transitions.

Algorithm 1: Feasible Transitions Algorithm

```
input: An assemblage of state machines A and an assemblage
proposition C
output: The set F of feasible state transition of A under C
foreach state machine c in the assemblage do
    foreach transition t in c do
        take the start state of the transition, say S;
        form the state proposition p = "state machine c is
        in state S";
        compute the state proposition p⊙C;
        if p⊙C is not empty then
            add transition t to the set F;
        end
    end
end
```

**[0125]** The set of feasible state transitions F can be further partitioned into two sets, $F_0$ and $F_1$, respectively of output and input feasible transitions, by trivially examining whether they belong, respectively, to the set $T_0$ or $T_1$ of output and input transitions of the state machine to which they belong. $F_0$ is called also the set of feasible non controllable transitions associated to C.

## Notation

**[0126]** We write (X, Y) for the state proposition $X_{=}^{c1}\odot Y_{=}^{c2}$ which denotes a single global state, once the assemblage having components C**1** and C**2** is known from the context.

## EXAMPLE 4

**[0127]** Let Cross be an assemblage composed by a pair of a traffic light of the kind depicted in FIG. **1**. Since each component state machine has four states, the whole assemblage may be found in sixteen states, which are depicted in FIGS. **15** and **16**, where traffic light states are laid linearly, together with the existing transitions, which can distinguished among controllable or not controllable by the graphical symbology used so far. Consider the totally fictional assemblage proposition traffic light **1** is in state Green and traffic light **2** is not in state q**0** or traffic lights are both in state Red or both in state Yellow, which can be written as $C=(((G_{=}^{tl1}\odot(\neg)\ q\mathbf{0}_{=}^{tl2})\oplus$

$(R_{=}^{tI1} \odot R_{=}^{tI2})) \oplus (Y_{=}^{tI1} \odot Y_{=}^{tI2}))$. State proposition C can be easily shown to be satisfied by any of the global states depicted as round squares in FIG. **15**, which also depicts the transitions which start from such states, that is the set of feasible transitions F associated to state proposition C. It can be observed that in some cases a transition leads to a state which still satisfies proposition C, while in other cases it does not. For example two transitions can be taken when the assemblage is in the global state (R, R), namely transition tI**2**.t**2** which leads to state (R, G), which does not belong to proposition p and transition tI**1**.t**2** which leads to state (G,R), which belongs instead to proposition p.

## Exit Zones

**[0128]** Given a state proposition C, there exist subpropositions of C such that, when the assemblage is within one of such subpropositions, there are non controllable transitions such that, if and when taken, move the assemblage to a global state such that C is no more satisfied. Such subpropositions are called "exit subpropositions" and play an important role in detecting when a state constraint may be violated. Given in fact a proposition C, we are interested in finding all the couples (p,t), where p is an exit subproposition and t is the non controllable transition, named "exit transition", that takes the assemblage out of C when the assemblage is in p. Algorithm 2 shown here below finds all the exit zones associated to a state proposition C given the set $F_0$ of feasible non controllable transitions associated to C. The algorithm works by forming, for each state transition t in $F_0$, the state proposition pre="state machine c is in the start state of t and satisfies C", then transform it according to the transition t. The resulting state proposition, named post="state machine c was in the start state of t and satisfied C then state transition t happened", may be only partially outside of the original proposition C, then it is trimmed, that is the state predicate postTrimmed="the assemblage satisfies post and does not satisfy C" is calculated. By reversing the direction of state transition t and by transforming postTrimmed according to such reversed transition, we found the proposition preTrimmed, which is the larger subproposition of pre such that: the assemblage is in a global state which satisfies pre and the happening of t brings the assemblage in a global state such that C is no more satisfied.

Algorithm 2: Exit Zones Algorithm

```
input:     The set $F_o$ of feasible non controllable state
           transition of assemblage A under proposition C
output:    The set Ez of exit zones associated to C
foreach transition t in the set $F_o$ do
    take the start state of transition t, say S;
    form the state proposition p = "state machine c is in
    state S";
    compute the state proposition pre = $p \odot C$;
    transform the state proposition pre according to t, let
    it be post;
    compute state proposition postTrimmed = $post \odot \neg C$;
    if postTrimmed is not empty then
        transform postTrimmed according to t reversed, let
        it be preTrimmed;
        add (preTrimmed, t) to the set Ez of exit zones of
        the assemblage;
    end
end
```

## Exit Zone Coverage

**[0129]** An exit zone $(p,t_k)$ associated to a state constraint C of a state S of the controller is said to be covered iff there exists in the controller a set of internally triggerable state transition $T_S$ such that each transition t in $T_S$ has state S as start state, is triggered by $t_k$ and the set of preconditions of the transitions in $T_S$ forms a partition of p.

## State Safety

**[0130]** The method ensures that, if all the state transitions in the controller are semantically safe and if each exit zone in the diagram has been covered by a state transition set in the controller then the following proposition holds:

## Proposition 1 (State Safety Invariant)

**[0131]** When the controller is in any state S, the assemblage is in a state q which satisfies vinc(S).

**[0132]** The truth of the proposition above can be inductively checked by ensuring that

- **[0133]** it is true in the initial state of the controller;
- **[0134]** it is true after any state transition in the controller; we have in fact that if the controller moves from state R to S by transition t, and if the state transition t is semantically correct, then the Proposition 1 above is verified since the postcondition semantics is included within the constraint of T;
- **[0135]** it is true after any state transition in the assemblage; In this case, either the state of the assemblage after the transition still satisfies the constraint for the state in the controller or it does not. In case it does not satisfies such constraint for the state, the assemblage transition necessarily originates from an exit subproposition of the constraint and the transition has therefore been previously classified as an exit transition. Since any exit zone has been covered by a state transition in the controller, a state transition, say tk, will be triggered and move the current state of the controller to another state of the controller, say T. Since the postcondition of tk is such that it satisfies the constraint of state T (observe that T may also coincide with S), the state safety invariant enounced above holds in this case also.

## Atomic Proposition

**[0136]** Given any current state q of the assemblage A it is possible to form the state proposition "assemblage A is in global state q". Such a state proposition is called the atomic state proposition associated to global state q. When it is clear form the context and no ambiguity arises, we use the notation q to denote both the assemblage global state as well as the associated atomic proposition.

## Connected Atomic Propositions

**[0137]** Given two assemblage atomic state propositions p and q, we say that they are connected iff there exist an assemblage transition, say c.t, such that p is transformed in q by c.t, in symbol q=transf (p, c, t)

## State Liveness

**[0138]** By state liveness we mean that:

**[0139]** 1. any state in the controller must be externally reachable, that is there must be a path of state transitions from the initial state $q_o$ to any state S in the controller;

**[0140]** 2. any atomic subproposition p of the state semantics vinc(S) of each externally reachable state S of the controller must in turn be internally reachable by the global state of the assemblage, that is, there must exists a path of assemblage state transitions from any subproposition of post(t) to subproposition p, where t is one of the incoming transitions to state S.

**[0141]** A state proposition is made of parts, that is subpropositions of the main state proposition, which are in general not connected, that is it does not exist any assemblage transition, either controllable or not, such that the assemblage current state will move from a subproposition to another. That means that some parts of the state constraint may not be reachable, and therefore, even if the state constraint is not violated, some expected properties may not be satisfied (safety and liveness are indeed orthogonal concepts).

**[0142]** In order to ensure that any part of a state proposition be reachable, control to each isolated part must be brought directly by different state transition of the controller. It may be observed that the state semantics vinc(S) may be viewed as a set of directed graphs made of subproposition arranged as Strongly Connected Components (SCC). Any atomic proposition within a SCC is such that it is reachable by any other atomic proposition within the same SCC. An atomic subproposition is itself a SCC. SCCs may be connected, and in such a case the resulting graph is acyclic, since if there were a cycle among two directed SCCA, then any atomic proposition in the former will be reachable from atomic proposition in the latter, and the two SCCs will become, by definition, a single SCC. Any state proposition can be therefore seen as a set of directed graphs having SCCs as nodes and assemblage state transitions as arcs. Any direct graph has one or more sources, that is elements of the digraph such that that there are not incoming arcs. Starting from the sources of a graph it is possible to reach any other part of the graph.

**[0143]** In order therefore to reach all the subpropositions of a state proposition, like the semantics vinc(S) of a state S in the controller, we have therefore to ensure that all the sources in any graph associated to such a proposition be reachable by at least one transition in the controller. As observed, each state transition denotes a set of states, named transition postcondition semantics, such that the assemblage is in one of such states when the transition is completed. If t is a transition in the controller such that its ending state is S, by checking that post(t) $\preceq \epsilon_{A}$s, for any source SCC s of any directed graphs associated to vinc(S) ensures that all the subproposition of the state semantics are reachable.

EXAMPLE 5

**[0144]** FIGS. **26** to **29** shows four state proposition (suppose they represent the semantics vinc(S) of state S in the controller) in the same layout of FIGS. **15** and **16** In FIG. **26** the proposition forms a single Strongly Connected Component (SCC). In this case, in order to ensure that any subproposition is reachable, we have simply to have one incoming transition covers at least one of the atomic propositions which make it such as, for example post(t)=(G, R) or post(t)=(G, G), and so on. In FIG. **27** it is instead shown a condition which consists of three different graphs, each one consisting of a single node, which is at the same time, the source of the graph. In order to ensure reachability, at least three transition $\{t_1, t_2, t_3\}$ must reach T in case the condition is the semantics of such a state, each one being such that post($t_1$)=(R, R), post($t_2$)=(G, G) and post($t_3$)=(Y, Y). The state proposition shown in FIG. **28** gives rise to two graphs, each consisting of a single SCC node. We show finally that the condition shown in FIG. **29** consists of a single graph with two nodes, where the SCC formed by the atomic propositions (G, R), (G, G) and (G, Y) is the source node.

State Proposition Boolean Algebra

**[0145]** It is possible to express logical propositions, that is statements which are either true or false, with respect to the state of a single device in the assemblage as well as propositions with respect to the global state of the entire assemblage. The former kind of propositions will be named state machine propositions, while the latter will be named assemblage state propositions. They will be collectively referred to as state propositions.

**[0146]** State machine and assemblage propositions are employed both to formulate operational aspects of the controller (like transition guard conditions) as well as to express constraints on the global behavior of the assemblage during the process of verification.

**[0147]** State machine and assemblage propositions will be denoted by symbolic expressions, named state machine and assemblage expressions.

EXAMPLE 6

**[0148]** Let "light**1**" be a traffic light device, whose behavior can be depicted by a state machine, which in turn can be found in state "Red", "Yellow" or "Green". Then "light**1** is in state Red", "light**1** is in state Yellow" and "light**1** is in state Green" are state machine propositions, which are true or false depending on the state of the device. Other state machine propositions can be built from simpler ones by ordinary propositional connectives, such as "light**1** is in state Red and (or) light**1** is in state Green", as well as "light**1** is not in state Red".

EXAMPLE 7

**[0149]** Let "light**1**" and "light**2** " be traffic light devices within an assemblage, such that each traffic light may be in state "Red", "Yellow" or "Green". Then "light**1** is in state Red and light**2** is in state Green" is an example of an assemblage state propositions, which is either true or false depending on the global state of the assemblage. Observe that any state machine proposition is also an assemblage proposition. Assemblage state propositions can be built from simpler ones by ordinary propositional connectives, such as "light**1** is in state Red and light**2** is in state Green (or) light**2** is in state Red".

Syntax

**[0150]** Let d∈A be any state machine of the assemblage A. We define basic state machine expressions the union of constant and atomic state machine expressions, defined as:

**[0151]** 1. $\text{none}_{=}^{d}$ and $\text{any}_{=}^{d}$ are state machine expression, which are said constant state machine expressions;

**[0152]** 2. If $S \in Q_d$ is any state of the state machine d, then $S_{=}^{d}$ is a state machine expression, which is said atomic state machine expression.

EXAMPLE 8

**[0153]** Let "light**1**" and "light**2**" be traffic light devices within an assemblage, such that each traffic light may be in state "Red", "Yellow" or "Green". Constant state machine expression $\text{any}_=^{light1}$ denotes the state proposition which always holds, that is the proposition that "says" that device "light1" may be found in state "Red", "Yellow" or "Green". Conversely, constant device expression $\text{none}_=^{light1}$ denote the state proposition which never holds. In other words, in "any state of the world", that is in any global state of the assemblage, the first proposition is always true and the latter is always false.

EXAMPLE 9

**[0154]** Let "light1" and "light2" be the traffic light devices within the assemblage of the example above. Then examples of atomic expressions are $\text{Red}_=^{light1}$, $\text{Red}_=^{light2}$ and $\text{Green}_=^{light1}$, which denote, respectively, the propositions which hold, respectively, when traffic light "light1" is in state "Red", when traffic light "light2" is in state "Red" and when traffic light "light1" is in state "Green".

Assemblage expressions

**[0155]** Assemblage expressions are built starting from state machine expressions and special assemblage constant expressions, which are combined, by means of the operators $\odot$, $\oplus$ and $\neg$ into more complex assemblage expressions.

**[0156]** 1. Any state machine expression is also an assemblage expressions;

**[0157]** 2. If A is a device assemblage, then $\text{ANY}_A$ and $\text{NONE}_A$ are assemblage expressions, named assemblage constant expressions;

**[0158]** 3. If $e_1$,$e_2$ are assemblage expressions, then $e_1 \odot e_2$, $e_1 \oplus e_2$ and $\neg e_1$ are assemblage expressions, named assemblage compound expressions.

**[0159]** If $e_1$ and $e_2$ are expressions which denote state propositions, then the compound state expression $e_1 \odot e_2$ denotes the conjunction of the two state propositions, that is the state proposition that holds when both the original state propositions hold.

**[0160]** In the same way, the expression $e_1 \oplus e_2$ denotes the disjunction of the two state propositions, that is the state proposition that holds when at least one of the original state propositions holds.

**[0161]** Finally, if e is an expression which denotes a state proposition, then the expression $\neg$ e denotes correspondingly the negation of the state proposition, that is the proposition which does not hold iff the original proposition holds.

EXAMPLE 10

**[0162]** Let $e_1$ be the expression $\text{Red}_=^{light1}$. Then $e_1$ denotes the state proposition which holds if and only if the device "light1" within an assemblage is in the state "Red". If, moreover, $e_2$ is the expression $\text{Green}_=^{light2}$, then the expression $\text{Red}_=^{light1} \odot \text{Green}_=^{light2}$ denotes the compound state proposition which holds iff device "light1" is in state "Red" and device "light2" is in state "Green". Similarly the expression $\text{Red}_=^{light1} \oplus \text{Green}_=^{light2}$ denotes the compound state proposition which holds iff either device "light1" is in state "Red" or device "light2" is in state "Green". Expression $\neg \text{Red}_=^{light1}$ denotes finally the proposition which does not hold if device "light1" is in state "Red" and holds in all the other states.

State Machine Expression Semantics

**[0163]** Given any state machine $d \in A$, and given any state q in which state machine d can be found, there exists a method for telling whether the proposition denoted by any device expression holds or does not hold.

**[0164]** There exists indeed a boolean valued function $[[e]]_{boot}: Q_d \rightarrow \{\text{true,false}\}$, where $Q_d$ is the set of states of the device, which assigns a truth value to any state machine expression e depending on the current state of the state machine. Such a boolean valued function represents the boolean semantics of device expressions and is recursively defined as follows:

**[0165]** For any device $d \in A$ and for any choice of states $q,S \in Q_d$:

**[0166]** 1. Constant state machine expressions:

$$[[\text{none}_=^c]]_{boot}(q)=\text{true and } [[\text{any}_=^c]]_{boot}(q)=\text{false for any } q \in Q_c$$

**[0167]** 2. Atomic state machine expressions:

$$[[S_=^c]]_{boot}(q) \text{ is true if and only if } q=S.$$

Assemblage Expression Semantics

**[0168]** The semantics of an assemblage expression e is a boolean valued function $[[e]]_{boot}: Q_A \rightarrow \{\text{true,false}\}$, where $Q_A$ is the set of global states of the assemblage A, which assigns a truth value to any assemblage expression e depending on the global states of the assemblage. Such a boolean valued function represents the boolean semantics of state machine expressions and is recursively defined as follows:

**[0169]** 1. if e is a state machine expression then: $[[e]]_{boot}(q)=[[e]]_{boot}(q)$, where q is the component of the global state q corresponding to the state machine to which expression e is referred to;

**[0170]** 2. $[[S_=^c \odot T_=^c]]_{boot}(q)=\text{false}$ if $S \neq T$, where c is any state machine of the assemblage;

**[0171]** 3. $[[e_1 \odot e_2]]_{boot}(q)=[[e_1]]_{boot}(q) \wedge e_2]]_{boot}(q)$ otherwise;

**[0172]** 4. $[[e_1 \oplus e_2]]_{boot}(q)=[[e_1]]_{boot}(q) \vee [e_2]]_{boot}(q)$

**[0173]** $\vee e_1]]_{boot}(q)$ is true iff$[[e_1]]_{boot}(q)$ is false;

where $\neg$ and $\wedge$ are the usual boolean logical operators.

**[0174]** It may be observed that:

- **[0175]** the semantics of the assemblage expression consisting of a single state machine expression (Rule 1) corresponds to the value of the semantics for such device expression, as defined in State machine expression semantics; in other words the semantics of device expressions is embedded within the semantics of assemblage expressions;
- **[0176]** the semantics of the meet ($\odot$) operator between two atomic device expressions is given by two different cases.
  - **[0177]** Rule 2 applies in the case in which both operands are atomic state machine expressions and the two expressions are referred to distinct states of the same device: it should be noted that it evaluates to false independently of the current state of the device, since the device may be found, at any time, in state S or in state T, but not in both;
  - **[0178]** Rule 3 applies in all the other cases.

The Boolean Algebra of Assemblage Expressions

**[0179]** It may be shown that the set of assemblage expressions form a Boolean Algebra, and as such it enjoys the related theorems. By analogy, we name the three operators as their boolean algebra counterparts, that is conjunction ($\odot$), disjunction ($\oplus$) and negation ($\neg$). We define also a minimal and a maximal element, respectively given by the expressions $NONE_A$ and $ANY_A$.

Containment Relationship Among Assemblage Expressions

**[0180]** We define a containment relationship among assemblage expressions, written as $\preceq\epsilon_A$, as follows: for any $e_1,e_2\in\epsilon_A$, we say that $e_1$ is contained within $e_2$, in symbols $e_1 \preceq\epsilon_A e_2$ iff $e_1\odot e_2=e_1$ or iff $e_1\oplus e_2=e_2$. We also define ad equivalence relationship $\equiv_{\epsilon_A}$ among state proposition by saying that $a\equiv_{\epsilon_A} b$ iff $a\preceq_{\epsilon_A} b$ and $b\preceq_{\epsilon_A} a$.

**[0181]** Deciding whether any assemblage expression is contained within any other any assemblage expression will be carried out by:

**[0182]** 1. transforming both assemblage expressions into the equivalent normal form called sum of products (see the paragraph Sum of products here below), consisting of the join, named sum, of a finite number of special assemblage expressions named products, consisting in turn of a fixed number of device expressions;

**[0183]** 2. comparing such sums of products by the algorithm in paragraph Containment among sums of products (see below), which in turn relies on the comparability among products of paragraph Containment among products (see below), which in turn relies on the comparability among device expressions of paragraph Containment among state machine expression (see below).

Transformation of a State Proposition Induced by a State Transition

**[0184]** Given an assemblage state proposition p and given a state transition t in a component c of the assemblage A, we say that p is transformed by transition t of component c into the state proposition p' after the only transition t happened, in symbols p'=transf(p, c, t). State proposition p' is the state proposition which means "the assemblage was in state p and the state transition t happened".

Transformation of a State Proposition Induced by an Assemblage Event

**[0185]** Given an assemblage state proposition p and given a transition request e directed towards a component c of the assemblage, namely, a command c.e, we say that, once such a transition request has been successfully accomplished, the assemblage is found in a state transfE (p, c.e). In general there are different transitions which are labelled by the same input event e in component c; let T be the set of such transitions. We define:

$$transfE(p,\ c.e) = \bigoplus_{t\in T} transf(p,\ c,\ t)$$

where T is the set of transitions which are triggerable by event e in component c, in symbols $T=trigger^{-1}(e)$.

Transformation of a State Proposition Induced by a List of Actions

**[0186]** Let l be a list of actions, let $\langle\rangle$, according to the usual notation be the empty list, let head(l) be the first element of the list and let tail(l) be the list containing the remaining elements when head(l) is removed from l. Then the transformation induced by l on an assemblage state proposition p is given by transfL(p,l), which is computed recursively as follows:

**[0187]** 1. if $l=\langle\rangle$, i.e, l is empty, then transfL(p,$\langle\rangle$)=p;

**[0188]** 2. if head(l)=c.e , i.e, the trigger is an assemblage event, then transfL(p,l)=transfL(transfE(p,c.e), tail(l));

**[0189]** 3. if head(l)=c.t , i.e, the trigger is an assemblage transition, then transfL(p,l)=transfL(transf(p,c,t), tail (l));

that is, the empty list induces no transformation at all, the list which has at least one element induces a transformation on the state proposition depending on whether the command is an assemblage event or an assemblage transition.

Sum of Products

**[0190]** Let $A=\{m_1,m_2, \ldots m_N\}$ be an assemblage of state machines. A sum of products is a canonical form which is given by the join (that is, the OR of the algebra of state propositions) of a finite number of terms $\pi_i$, called products:

$$s=\pi_1\oplus\pi_2\oplus\ldots\oplus\pi_M$$

where each product

$$\pi_1=b_1(m_1)\odot b_2(m_2)\odot b_N(m_N)$$

is the meet (that is, the AND of the algebra of state propositions) of exactly N state machine propositions $b_i(m_i)$, which is a basic state machine proposition related to state machine $m_i\in A$.

**[0191]** Sums of products are a canonical form of the algebra of state propositions. A sum of product is a special expression which denotes a state proposition. An effective method for transforming any state expression into a sop expression is described in paragraph Containment Among Products (see below); in the paragraph The Boolean Algebra of Sum of Products (see below) it is observed that sum of products form a boolean algebra themselves.

**[0192]** It is possible moreover to describe effective algorithms for computing basic operations in the algebra of sum of products (sop algebra) as well as to decide whether one expression is contained within another. It is therefore possible to compute any operation or make any comparison among state propositions by:

**[0193]** 1. first transforming the operand state propositions into their equivalent sop normal form by the Algorithm in paragraph Containment Among Products;

**[0194]** 2. applying the corresponding computable operation or comparison operator of the sop algebra to the sop operands.

EXAMPLE 11

**[0195]** Let "light**1**" and "light**2**" be traffic light devices within an assemblage, such that each traffic light may be in state "Red", "Yellow" or "Green". It is then possible to write, among the others, the following products:

$$\pi_1=Red_=^{light1}\odot Red_=^{light2};$$

$$\pi_2=Red_=^{light1}\odot Green_=^{light2};$$

$$\pi_3=Red_=^{light1}\odot any_=^{light2};$$

Notation

**[0196]** In order to improve readability:

**[0197]** 1. we enclose the operands making a sum within braces and separate them by commas: we write thus

$s=\{\pi_1,\pi_2, \ldots, \pi_M\}$ in place of $s=\pi_1\oplus\pi_2\oplus \ldots \oplus\pi_M$; we also say that the sum s "contains" products $\pi_1$, $\pi_2$ and so on;

**[0198]** 2. we enclose the operands making a product within square brackets and separate them by commas: we write thus $[\text{Red}_{=}^{light1}, \text{Red}_{=}^{light2}]$ in place of $\text{Red}_{=}^{light1}\odot\text{Red}_{=}^{light2}$;

**[0199]** 3. we omit constant device expressions of the kind $\text{any}_{=}^{d}$: we write thus $[\text{Red}_{=}^{light\ 1}]$ in place of $\text{Red}_{=}^{light1}\odot\text{any}_{=}^{light2}$.

**[0200]** 4. the term "sum of products" will be shortened either to simply "sum" or to the acronym SOP (Sum Of Products) or "sop"; we will speak also of "sop normal form" when referring to sop expressions.

**[0201]** Finally, products which are equivalent to the empty state proposition $\text{NONE}_A$ can be omitted from a sum s, yielding a sum s' which can be easily proven equivalent to the original one.

EXAMPLE 12

**[0202]** The products of the example 11 above may then be written as:

**[0203]** $\pi_1=[\text{Red}_{=}^{light1}, \text{Red}_{=}^{light2}]$;

**[0204]** $\pi_2=[\text{Red}_{=}^{light1}, \text{Green}_{=}^{light2}]$

**[0205]** $\pi_3=[\text{Red}_{=}^{light1}]$

**[0206]** A sum containing the three products above then may be written as

**[0207]** $\{[\text{Red}_{=}^{light1}, \text{Red}_{=}^{light2}],[\text{Red}_{=}^{light1}, \text{Green}_{=}^{light2}],[\text{Red}_{=}^{light1}]\}$

The Boolean Algebra of Sum of Products

**[0208]** Sum of products form a Boolean algebra which is isomorphic to the algebra of state propositions. It can be observed that:

**[0209]** 1. for any operator of the algebra of state propositions there exists a correspondent operator in the algebra of sum of products:

**[0210]** (a) meet operator $\odot'$, described in paragraph Meet Among Sum of Products (see below) and used to compute the intersection of two sums;

**[0211]** (b) join operator $\oplus'$, described in paragraph Join Among Sum of Products (see below), used to compute the union of two sums;

**[0212]** (c) complement operator $\neg$, described in paragraph Complement of a Sum of Products (see below), used to compute the complement of a sum;

**[0213]** 2. there exist two special sums of products which act as the zero and unity elements of the algebra:

**[0214]** (a) an empty sum s of products denotes the zero value of the algebra of assemblage state propositions associated with assemblage A, that is $s\equiv\in_A\text{NONE}_A$. We use therefore the empty sum { } in order to denote such value;

**[0215]** (b) different sums s may denote the unity value of the algebra of state propositions associated with assemblage A, that is, for any of such sums, $s\equiv\in_A\text{ANY}_A$.

We refer collectively to such equivalence class by introducing a special sum symbol, that is $\{\text{ANY}_A\}$.

Algorithm for Transforming State Propositions into Sum of Products: sop

**[0216]** It is possible to transform any assemblage expression e denoting a state proposition into a sum of products SOp(e) by the following recursive algorithm:

**[0217]** Algorithm 3 computes the sum of products corresponding to any assemblage expression e by recursive calls to itself (SOp(e), Algorithm 3) as well as by calls to Algorithms which calculate the meet (meetSums, Algorithm 6), the join (joinSums, Algorithm 5) and the complement (complementSum, Algorithm 7) of sums.

**[0218]** Algorithm 3 works by assuming that the sum of products corresponding to the meet of two expressions $e_1$, $e_2$ is the meet of the sums of the two operand expressions $e_1$, $e_2$; the sum of products corresponding to the join of two expressions $e_1$, $e_2$ is the join of the sums of the two operand expressions $e_1$, $e_2$ and finally the sum of products corresponding to the complement of an expressions e is the complement of the sum of the operand expression e. The base case is given by the assemblage expression which consists only of a state machine expression d about component machine m: in this case sop(d) returns a sum $\{\pi_d\}$ containing a single product $\pi_d$, which is built as a meet of the state machine expression d and of a constant state machine expression $\text{any}_{=}^{c}$ for any other component c different from m.

Algorithm 3: sop(e) Sum of Products Generation Algorithm

```
input  :   An assemblage expression e of the form $e = e_1\oplus e_2$,
           $e = e_1\odot e_2$ or $e = \neg e_1$ or e=d, where d is a state machine
           expression
output:  A sum of products
if $e = e_1\odot e_2$ then
    return: meetSums(sop($e_1$), sop($e_2$));
end
if $e = e_1\oplus e_2$ then
    return: joinSums(sop($e_1$), sop($e_2$));
end
if $e = \neg e_1$ then
    return: complementSum($e_1$);
end
if e=d then
    return: $\{\pi_d\}$;
end
```

EXAMPLE 13

**[0219]** Suppose we want to transform the compound expression

$$\neg(\text{Red}^{tl1}\odot(\text{Green}_{=}^{Tl1}\oplus\text{Red}_{=}^{tl2}))$$

into a sum of products. Then:

**[0220]** 1. We take the complement of the sum obtained by transforming the expression $\neg(\text{Red}_{=}^{tl1}\odot(\text{Green}_{=}^{tl1}\oplus\text{Red}_{=}^{tl2}))$ into a sum (see step 2) below); let such a sum be $s_1$;

**[0221]** 2. transforming the expression $\text{Red}_{=}^{tl1}\odot(\text{Green}_{=}^{tl1}\oplus\text{Red}_{=}^{Tl2})$ into a sum involves transforming the operands $\text{Red}_{=}^{Tl1}$ and $(\text{Green}_{=}^{tl1}\oplus\text{Red}_{=}^{tl2})$ into the corresponding sums and then taking their intersection (see steps 3 and 4 below): let such a sum be $s_2$;

**[0222]** 3. the expression $\text{Red}_{=}^{tl1}$ is a device expressions and then it is transformed into the sum $s_3=\{[\text{Red}_{=}^{tl1}]\}$

**[0223]** 4 . transforming the expression $\text{Green}_{=}^{tl1}\oplus\text{Red}_{=}^{tl2}$ into a sum, requires to transform the operands $\text{Green}_{=}^{tl1}$ and $\text{Red}_{=}^{tl2}$ into the corresponding sums and then taking their union (see steps 5 and 6 below) : let such a sum be $s_4$;

**[0224]** 5. the expression $\text{Green}_{=}^{tl1}$ is a device expressions and then itnsformed into the sum $s_5=\{[\text{Green}_{=}^{Tl1}]\}$;

**[0225]** **6**. the expression $Red_{=}^{tI2}$ is a device expressions and then it is transformed into the sum $s_6=\{[Red_{=}^{tI2}]\}$;

**[0226]** We have therefore that

**[0227]** $s_4=s_5\odot' s_6=\{[Green_{=}^{TI1}],[Red_{=}^{tI2}]\}$

**[0228]** $s_2=s_3\odot' s_4=\{[Red_{=}^{tI1}]\}\odot'\{[Green_{=}^{tI1}],[Red_{=}^{tI2}]\}$

**[0229]** We then take the meet of the product in the first sum and the first product of the second sum

**[0230]** $\pi_1=[Red_{=}^{tI1}]\odot''[Green_{=}^{TI1}]=[Red_{=}^{tI1},Green_{=}^{tI1}]$ and we then take the meet of the product in the first sum and the second product of the second sum $\pi_2=[Red_{=}^{tI1}]\odot''[Red_{=}^{tI2}]=[Red_{=}^{tI1}, Red_{=}^{tI2}]$

**[0231]** Sum $s_2$ will contain only product $\pi_2$ since product $\pi_1$ denotes the empty state proposition: $s_2=\{[Red_{=}^{tI1},Red_{=}^{tI2}]\}$

**[0232]** Finally: $s_1=\neg' s_2=\{[Green_{=}^{tI2}],[Green_{=}^{tI1}],[Yellow_{=}^{tI2}],[Yellow_{=}^{tI1}]\}$

## Addition of a Product to a Sum of Products: AddProductToSum

**[0233]** Given a sum s and a product π' we want to obtain the sum s' such that s' denotes the state proposition which is equivalent to the state proposition $s\oplus\pi'$. See Algorithm 4.

Algorithm 4: addProductToSum (s,π')

```
input    :  A sum of product s and a product $\pi$
output   :  A sum of products s'
if s = { } then
    return: s' = {$\pi'$};
end
if head(s)$\preceq\epsilon_A\pi'$ then
    return: s' = addProductToSum (tail (s), $\pi'$);
end
if $\pi\preceq\epsilon_A$head(s) then
    return: s' = s;
end
if $\pi'\not\preceq\not\epsilon_A$ head(s) and head(s) $\not\preceq\not\epsilon_A$ $\pi'$ then
    return: s' = joinSums (s, {$\pi'$}) ;
end
```

## Join Among Sums of Products: JoinSums

**[0234]** The products in the two sums are added sequentially, by Algorithm 4, to a sum s' initially empty. See Algorithm 5.

Algorithm 5: joinSums($s_1$,$s_2$)

```
input:   Two sum of products $s_1$ and $s_2$
output:  A sum of products s
s' = { };
foreach $\pi_1 \in s_1$ do
    s' = addProductTo Sum(s',$\pi_i$);
end
foreach $\pi_j \in s_2$ do
    s' = addProductTo Sum(s',$\pi_j$);
end
return: s';
```

## Meet Among Sums of Products: MeetSums

**[0235]** For any pair of products in the two sum a new product is created, through Algorithm 9, then such product is added, by Algorithm 4, to a sum s initially empty. See Algorithm 6.

Algorithm 6: meetSums($s_1$, $s_2$) Algorithm

```
input:   Two sum of products $s_1$ and $s_2$
output:  A sum of products s
s = { };
foreach $\pi_j \in s_1$
    foreach $\pi_j \in s_2$ do
        $\pi$ = meetProducts($\pi_i$,$\pi_j$);
        s = addProductTo Sum(s,$\pi$)
    end
end
return: s;
```

## Complement of a Sum of Products: ComplementSum

**[0236]** For each product $\pi_i$ in sum s we compute the sum s which denotes the negation of the product through Algorithm 10 and add each product $\pi_j$ in s to the sum s' through Algorithm 4. See Algorithm 7.

Algorithm 7: complementSum($s_1$)

```
input :   A sum of products $s_1$
output :  A sum of products s'
s' = { };
foreach $\pi_i \in s_1$ do
    s = complementProduct($\pi_i$);
    foreach $\pi_i \in$ s do
    S' = addProductTo Sum(s',$\pi_j$);
    end
end
return: s';
```

## Containment Among Sums of Products: ContainmentAmongSums

**[0237]** We compare for containment each product $\pi_i$ in the first sum with each product $\pi_j$ in the second sum through Algorithm 11. It returns true iff any comparison for containment among products $\pi_i$ and $\pi_j$ is true. See Algorithm 8.

Algorithm 8: containmentAmongSums($s_1$,$s_2$)

```
input:   Two sums of products $s_1$,$s_2$
output:  A boolean value
foreach $\pi_i \in s_1$ do
    foreach $\pi_j \in s_1$ do
        if containmentAmongProducts($\pi_i$,$\pi_j$) = false then
            return: false;
        end
    end
end
return: true;
```

## Meet Among Products: MeetProducts

**[0238]** Let $b_m(\pi)$ be the operand of a product which is the state machine proposition related to state machine m, given an assemblage A. The meet of two products is then given by the product which has the state machine proposition given by the meet of the two state propositions related to state machine m in the two operand products. See Algorithm 9.

Algorithm 9: meetProducts($\pi_1$,$\pi_2$)

```
input: Two products $\pi_1$ and $\pi_2$
output: A product $\pi$
$\pi$ = any$_=^m$;
foreach m $\in$ A do
    $b_m(\pi)$ = meetAmongStateMachinePropositions($b_m(\pi_1)$,$b_m(\pi_2)$);
end
return: $\pi$;
```

Complement of a Product: ComplementProduct

**[0239]** See Algorithm 10.

Algorithm 10: complementProduct($\pi_1$,$\pi_2$)

```
input: A product $\pi$
output: A sum s
s = { };
foreach m $\in$ A do
    if $b_m(\pi)$ = any$_=^m$ then
        return: s = { };
    endif
    else if $b_m(\pi)$ = none$_=^m$then
        return: s = {ANY$_A$};
    endif
    else if $b_m(\pi)$ = T$_=^m$ then
        $\pi$ = ANY$_A$;
        foreach S $\in$ $Q_m$ do
            if S $\neq$ T then
                $b_m(\pi)$ = S$_=^m$;
            endif
        end
        s = addProductTo Sum(s,$\pi$);
    endif
end
return: s;
```

EXAMPLE 14

**[0240]** Let $\pi_1$ $\mathrm{Red}_=^{light1}\odot\mathrm{Green}_=^{light2}$. Its negation $\neg\pi_1$ is then computed by taking the sum resulting from the negation of the basic device expressions $\mathrm{Red}_=^{light1}$ and $\mathrm{Green}_=^{light2}$. In the former case we have that $s_l=\mathrm{Yellow}_=^{light1}\oplus\mathrm{Green}_=^{light1}$, while the latter results in the sum $s_2=\mathrm{Yellow}_=^{light2}\oplus\mathrm{Red}_=^{light2}$. The final sum is then given by $s=s_1\oplus s_2=\mathrm{Yellow}_=^{light1}\oplus\mathrm{Green}_=^{light1}\oplus\mathrm{Yellow}_=^{light2}\oplus\mathrm{Red}_=^{light2}$.

Containment Among Products: ContainmentAmongProducts

**[0241]** See Algorithm 11.

Algorithm 11: containmentAmongProducts($\pi_1$,$\pi_2$)

```
input: Two sums of products $\pi_1$,$\pi_2$
output: A boolean value
foreach m $\in$ A
    if containmentAmongStateMachinePropositions($b_m(\pi_1)$,$b_m(\pi_2)$) =
    false;
    then
        return: false;
    end
end
return: true;
```

Transformation of a Product: TransformStateMachineProposition

**[0242]** We take each state machine expression in the product and we transform it . The result is a sum. Let $\pi_1=\odot_{m\in A}b_1(m)$ where $b_1(m)$ is the basic state machine expression corresponding to the component m of the assemblage A. Then, $\pi_1$ is transformed by the transition t of component c into the sum s as follows : let $s_c$ be the sum corresponding to the transformation of the state machine expression $b_1(c)$ induced by transition t of component c. Let $s_c=\oplus_{i=1\ldots N}\pi_i$. Then the transformation is given by the sum $s_c=\oplus_{i=1\ldots N}\pi'_i$ where each product $\pi'_i=\odot_{m\in A}b'(m)$ is obtained by the corresponding product $\pi_i=\odot_{m\in A}b(m)$ of $s_c$ by the rules: $b'(m)=b(m)$ if c=m and $b'(m)=b_1(m)$ if c≠m.

Intersection Among State Machine Expressions

**[0243]** The algorithm is expressed in tabular form as shown below; p and q are distinct states belonging to state machine having rolename r.

| $\odot'''$ | none$_=^r$ | p$_=^r$ | q$_=^r$ | any$_=^r$ |
|---|---|---|---|---|
| none$_=^r$ | none$_=^r$ | none$_=^r$ | none$_=^r$ | none$_=^r$ |
| p$_=^r$ | none$_=^r$ | p$_=^r$ | none$_=^r$ | p$_=^r$ |
| q$_=^r$ | none$_=^r$ | none$_=^r$ | q$_=^r$ | q$_=^r$ |
| any$_=^r$ | none$_=^r$ | p$_=^r$ | q$_=^r$ | any$_=^r$ |

Containment Among State Machine Expressions

**[0244]** The algorithm is expressed in tabular form as shown below; p and q are distinct states belonging to state machine having rolename r.

| $\leqq'''\epsilon_A$ | none$_=^r$ | p$_=^r$ | q$_=^r$ | any$_=^r$ |
|---|---|---|---|---|
| none$_=^r$ | true | false | false | false |
| p$_=^r$ | false | true | false | true |
| q$_=^r$ | false | false | true | true |
| any$_=^r$ | false | false | false | true |

Communication and Synchronization

**[0245]** The control model provides two very general models of synchrony.

**[0246]** 1. state machines operate through an internal, never ending cycle, which iterates a basic computation step. During a computation step, signals sent to the machine are evaluated, and one, if any, state transition is chosen for execution. A state transition execution consists in computing a new current state and in sending out signals directed to other state machines;

**[0247]** 2. state machines communicate through some communication medium, which again operates through one or more never ending cycles. Such cycles iterate basic communication computations, which consist, essentially, in delivering signals from one machine to the another.

**[0248]** 3.

**[0249]** In the asynchronous model each machine is driven by a separate thread (by using a software oriented language) or a separate processor (by using an hardware oriented language). The communication medium is again driven by one or more (different) threads or processors. The three main entities of the control model (controller and controlled machines, communication medium) are therefore behaviorally independent and synchronize only through communication ports. A communication port is a block of memory which is shared among the different processes. The processes read and write control signals by a typical producer consumer pattern of execution. To prevent processes from reading or writing the shared data at the same time, one or more mutex or read-write locks are employed. Finally, the shared block of memory can be structured as a FIFO list, in order to have the producer not to stop in case a new control signal is produced before a previously produced control message has been consumed.

**[0250]** In the synchronous model both the controller and the controlled state machines, as well as the communication medium, are driven by a unique thread or processor. By using the software oriented language, that means that while executing its internal cycle, the controller state machine stops and starts executing both computations of the communication medium as well as, sequentially, the internal cycle of each controlled machine. By using an hardware oriented language, either the controlled machines are part of the processor and execute in its main cycle, or some sort of time driven, or master-slave synchronization is implemented through different processors.

**[0251]** The control model will be able not only to provide both kinds of synchronization, but also to host a mix of them. In other words, given a controller state machine and a set of controlled state machines, it may be the case that some machines in the set are controlled through the asynchronous model, and the others through the synchronous one. As an example, the same computer processor may control other machines asynchronously through a field bus and, at the same time control other internal state machines synchronously, like timers or adders, by the internal motherboard communication bus.

**[0252]** Finally, a simple, yet abstract, way of differentiating the two models independently of implementation issues, consist in contrasting them by comparing execution times:

- **[0253]** Asynchronous behavior: each request of behavior is served within a given delay, due to the internal work of the controlled machine and to the propagation time taken by the request in travelling from a controller to a controlled machine; in the same way, notification of behavior takes some time to travel back from the controlled to the controller machine.
- **[0254]** Synchronous behavior: each request of behavior is served instantaneously.

Current-State Array

**[0255]** Each state machine is equipped with an array of symbols, each denoting the currently known state of a component machine of the assemblage under control of the machine.

**[0256]** Such an array of symbol is kept up to date with the current state of the component machines as part of the workflow of the state machine and of the communication medium, by means of the messages exchanged, as explained below in paragraph Current State Array and Incoming Event Computation (see below).

Incoming Internal Event

**[0257]** Each state machine is equipped with a variable holding the event symbol (if any) associated to the last transition which took place in a component machine. The content of such variable is kept up to date as part of the workflow of the state machine and of the communication medium, by means of the messages exchanged, as explained below in paragraph Current State Array and Incoming Event Computation.

Incoming Internal Transition Symbol

**[0258]** Each state machine is equipped with a variable, which coincides with the CMO port of paragraph CMOP Communication Medium Output Port (see below), holding the symbol which denotes the last transition which happened within the assemblage in the form of a TCIC signal (see below paragraph TCIC—Transition Completed in Component).

Control Signals

**[0259]** Control signals are used in order to coordinate the joint behavior of the assemblage and of the controller. They are generated by either one of the assemblage components or by the controller, and processed by the communication medium.

TC—Transition Completed

**[0260]** A transition completed signal is generated by a component state machine in order to notify that a specific transition happened within the machine. It consists of the bare transition identifier and is sent to the communication medium.

TCIC—Transition Completed in Component

**[0261]** A TCIC signal identifies univocally a transition within the whole assemblage of components. A TCIC signal is generated by the communication medium as part of its workaround: once a TC signal t generated by a state machine c is received by the communication medium, the TCIC signal (c, t) is sent to the controller.

ERIC—Event Required in Component

**[0262]** This signal, which will be referred to as command, is generated by the controller in order to ask a specific component state machine to undertake some state transition labelled by a specific input event. A command consists of the identifier of the machine plus an event symbol belonging to the machine input events. For example, by sending the command (c, e) the controller asks the state machine c to undertake a state transition, if any, departing from the current state of c and labeled by the input event e. Such a transition t is such that event (t)=e.

Communication Ports

**[0263]** We distinguish the different kinds of ports by the typology of message exchanged and by the producer and the consumer of such signals.

COP Component Output Port.

**[0264]** This port is placed between a component state machine and the communication medium and hosts a queue of TC signals, which are produced by the component state machine and consumed by the communication medium.

CMOP Communication Medium Output Port.

**[0265]** This port is placed between the communication medium and the controller and hosts a queue of TCIC signals, which are produced by the communication medium and consumed by the controller.

CIP Component Input Port.

**[0266]** This port is placed between the communication medium and the component state machine and hosts a queue of event signals, which are produced by the communication medium and consumed by the component state machine.

CMIP Communication Medium Input Port.

**[0267]** This port is placed between the communication medium and the controller and hosts a queue of ERIC signals, which are produced by the controller and consumed by the communication medium.

Communication Architecture

**[0268]** By communication architecture we mean the global arrangement of component and controller state machine and of the communication medium by means of communication ports.

**[0269]** A typical communication architecture involving an assemblage of component state machines $A=\{C_1,C_2, \ldots C_n\}$ and a controller state machine C is shown in FIG. **30**. Each component state machine is connected to the communication medium M by two ports, respectively a component input (CIP) and output (COP) port. The communication medium M is on its turn connected to the controller C by two ports, respectively a communication medium input (CMI) and output (CMO) port.

**[0270]** It is also possible to have a multilevel arrangement of component and controller state machines, since each controller can be attached to another communication medium as if it were a component on its turn. A typical case of multilevel arrangement is shown in FIG. **31**, where two assemblages, namely $A_1=\{C_{11},C_{12}, \ldots , C_{1N}\}$ and $A_2=\{C_{21},C_{22}, \ldots , C_{2N}\}$ are controlled, respectively, by the controller state machines $C_1$ and $C_2$, through the communication media $M_1$ and $M_2$ and the respective communication ports.

**[0271]** Controller state machines $C_l$ and $C_2$ are moreover attached on their turn to a third communication medium $M_3$, and as such they become component of a third assemblage $A_3=\{C_1,C_2\}$. The communication medium $M_3$ is finally connected to the controller C by a communication medium input (CMI) and output (CMO) port.

**[0272]** It is possible to observe that state machines may be grouped into different typologies, depending on the control they exercise upon other machines or, vice versa, the control other machines exercise upon them:

**[0273]** 1. state machines which are simply controlled, like the state machines in the assemblages $A_1$ and $A_2$;

**[0274]** 2. state machine which exercise control and are at the same time controlled, like the state machines $C_1$ and $C_2$ making the assemblage $A_3$;

**[0275]** 3. state machines which simply exercise control other state machines like state machine C.

Communication Medium Workflow

**[0276]** The communication between the controller and the controlled state machines happens through a communication medium which is an operating entity whose aim is to bring control signals from the components to the controller and vice versa. We present an abstract operational model of the behavior (whose main tasks are depicted in FIGS. **32** and **33**) which may be implemented in a variety of different ways upon different communication technologies.

**[0277]** controller-component communication: this task (depicted in FIG. **32**) is aimed at notifying the controller that a transition happened within a specific controlled machine c of the assemblage. The component machine does not contain any information regarding neither the existence of any controller nor that the state machine itself is identified by c within the assemblage. It therefore simply emits a transition completed (TC) signal towards the communication medium, which has the duty of completing it with the additional information that such a signal was produced by c. The communication medium therefore wraps the TC signal within a TCIC signal, which indeed embeds the additional information about the source of the TC signal. The TCIC is finally dispatched to the controller, which then updates its knowledge about both the current state of c and the output event optionally emitted;

**[0278]** component-controller communication: this task (depicted in FIG. **33**) is aimed at notifying the component c that an action, say c.e has been sent to it from the controller. This time the controller is aware of the existence of component c, therefore the communication medium simply unwraps the ERIC signal by using the destination part to deliver the message to the component and by depositing the event part to the component input port (CI) for being consumed and processed.

Current State Array and Incoming Event Computation

**[0279]** When a TRIC signal c.t is dispatched to the state machine, the state machine updates the array of the current states and the incoming output event variable, as shown in FIG. **34**:

**[0280]** 1. the last state observed in component c is known by the current array entry;

**[0281]** 2. given the state transition identifier t it becomes possible to fetch the new current state of the component and to update such an array; we assume that a state machine acting as a controller for another state machine c is equipped with a description (say in tabular form) of the controlled machine;

**[0282]** 3. in the same way it becomes possible to know the output event associated with the transition, and to update the incoming output event variable accordingly.

State Transition Selection

**[0283]** Given the current state s of a state machine, the current state array, a current internal incoming event (if any), a current external incoming event (if any) and the array of current component states, we say that a (possibly empty) set of state transitions is selected for being executed iff for each state transition in such a set the following conditions are verified:

**[0284]** 1. the transition has the state s as its departing state; and

**[0285]** 2. the guard condition is satisfied; and

**[0286]** (a) the internal incoming event matches the transition internal trigger (if any); Or

**[0287]** (b) the external incoming event matches the transition external trigger (if any); or

**[0288]** (c) the transition is automatic (it has neither an internal nor an external trigger);

State Transition Execution

**[0289]** Given a state transition belonging to a state machine the state transition is executed when:

**[0290]** 1. the target state of the state transition becomes the current state;

**[0291]** 2. the actions in the action list which labels, if any, the state transition are sent to the communication medium input (CMI) port;

**[0292]** 3. a transition completed (TC) signal is sent to the component output (CO) port.

State Machine Behavior

**[0293]** The behavior of a state machine consists in performing an initialization phase, then in repeating indefinitely an execution cycle.

**[0294]** In the initialization phase:

**[0295]** 1. it is requested that each component of the assemblage under control of the state machine (if any) communicates at least once its internal status and current event;

**[0296]** 2. a subset of the automatic state transitions which have the initial state as initial state are selected for execution and the first transition in the subset is chosen for execution.

**[0297]** The execution cycle consists of the alternate fetch of signals coming from both the components and the controller and on the execution of either the transitions which have those signals as triggers or are automatic:

**[0298]** 1. the component input port (CI) is looked up;

**[0299]** (a) in case a transition completed in component

**[0300]** (TCIC) signal is present, it is fetched and removed from the port, then the current state array and the current event are updated;

**[0301]** (b) a set of transition are selected for execution;

**[0302]** (c) the first transition in the set of transition obtained at the previous point is chosen for execution;

**[0303]** 2. the communication medium input (CMI) port is looked up;

**[0304]** (a) in case an input event signal is present, it is fetched and removed from the port;

**[0305]** (b) a set of transition are selected for execution;

**[0306]** (c) the first transition in the set of transition obtained at the previous point is chosen for execution.

**1-27**. (canceled)

**28**. A method for controlling a physical machine or an assemblage of physical machines for ensuring safety and liveness rules in a state based design of said physical machine or assemblage of physical machines, comprising the steps of associating at least one logical state to at least one physical state that said physical machine or assemblage of physical machines may assume, providing state constraints for said logical states, and checking that a physical state assumed by said physical machine or assemblage of physical machines is associated to a logical state complying with said state constraints.

**29**. A method according to claim **28**, and further comprising the steps of moving said physical machine or assemblage of physical machines out of a physical state if said physical state is not associated to a logical state that complies with said state constraints, and forcing said physical machine or assemblage of physical machines to assume a physical state associated to a logical state that complies with said state constraints.

**30**. A method according to claim **28**, wherein said state constraints comprise a set of logical states associated to physical states that said physical machine or assemblage of physical machines is allowed to assume.

**31**. A method according to claim **30**, and further comprising the step of verifying whether each state of said set of logical states may be reached as a result of a transition starting from another logical state of said set of logical states.

**32**. A method according to claim **30**, and further comprising the step of verifying that said physical machine or assemblage of physical machines is capable of reaching any physical state associated to a logical state of said set of logical states.

**33**. A method according to claim **30**, and further comprising the step of verifying that each physical state assumed by said physical machine or assemblage of physical machines during a transition from an initial physical state to a final physical state is associated to a logical state of said set of logical states.

**34**. A method according to claim **28**, and further comprising the step of associating to a physical machine a state machine that is a logical machine having logical states, each of which corresponds to at least one physical state of said physical machine.

**35**. A method according to claim **28**, and further comprising the step of associating to an assemblage of physical machines an assemblage of state machines, said assemblage of state machines having logical states each of which corresponds to at least one physical state of said assemblage of physical machines.

**36**. A method according to claim **35**, wherein each state machine of said assemblage of state machines is associated with a respective physical machine of said assemblage of physical machines, each logical state of each state machine corresponding to a physical state of the respective physical machine.

**37**. A method according to claim **34**, wherein a state machine is associated to a respective physical machine by means of an interface, said interface comprising a sensor capable of converting a physical state of said physical machine into a logical state of said state machine and an actuator capable of converting logical commands of said state machine into physical commands acting on said physical machine.

**38**. A method according to claim **34**, wherein said state constraints comprise a set of logical states that a state machine is allowed to assume.

**39**. A method according to claim **35**, wherein said state constraints comprise a set of logical states that an assemblage of state machines is allowed to assume.

**40**. A method according to claim **35**, wherein a state machine may act as a controller of another state machine or assemblage of state machines, or be controlled by another state machine or assemblage of state machines.

**41**. A method according to claim **40**, wherein said state constraints comprise a set of logical states a state machine or an assemblage of state machines controlled by a controller may assume when said controller is in a given state.

**42**. A method according to claim **38**, and further comprising the step of verifying whether a state machine is in a logical state complying with said constraints before a transition from said logical state to a second logical state occurs.

**43**. A method according to claim **42**, and further comprising the step of verifying that said second logical state complies with said constraints.

**44**. A method according to claim **39**, and further comprising the step of verifying whether an assemblage of state machines is in a logical state complying with said constraints before a transition from said logical state to a second logical state occurs.

**45**. A method according to claim **44**, further comprising verifying that said second logical state complies with said constraints.

**46**. A method according to claim **42**, wherein a transition from a logical state to said second logical state in said state machine is associated with a logical command that may be converted into a physical command for generating a physical transition in a physical machine associated with said state machine.

**47**. A method according to claim **44**, wherein a transition from a logical state to said second logical state in said assemblage of state machines is associated with a logical command that may be converted into a physical command for generating a physical transition in an assemblage of physical machines associated with said assemblage of state machines.

**48**. A method according to claim **34**, wherein a transition from a physical state to a second physical state in said physical machine generates a transition from a logical state to a second logical state in a state machine associated with said physical machine.

**49**. A method according to claim **48**, wherein said state machine verifies whether said transition complies with said state constraints.

**50**. A method according to claim **49**, wherein said state machine generates a logical command that may be converted into a physical command for forcing said physical machine to assume a physical state associated to a logical state of said state machine that complies with said state constraints, if said transition does not comply with said state constraints.

**51**. A method according to claim **35**, wherein a transition from a physical state to a second physical state in said assemblage of physical machine generates a transition from a logical state to a second logical state in an assemblage of state machines associated with said assemblage physical machines.

**52**. A method according to claim **51**, wherein said assemblage of state machines verifies whether said transition complies with said state constraints.

**53**. A method according to claim **52**, wherein said assemblage of state machines generates a logical command that may be converted into a physical command for forcing said assemblage of physical machines to assume a physical state associated to a logical state of said assemblage of state machines that complies with said state constraints, if said transition does not comply with said state constraints.

**54**. A method according to claim **40**, and further comprising the step verifying whether only one transition is activated by an external event, if there exist a plurality of transitions that may be activated by said external event under respective given conditions.

* * * * *