15. A Mechanical Press Controller

Jean-Raymond Abrial (ETHZ)

March 2008

1. Informal presentation of the example
2. Presentation of some design patterns
3. Writing the requirement document
4. Proposing a refinement strategy
5. Development of the model using refinements and design patterns
6. Demos

1. Informal Presentation of the Example

Presentation of the Example

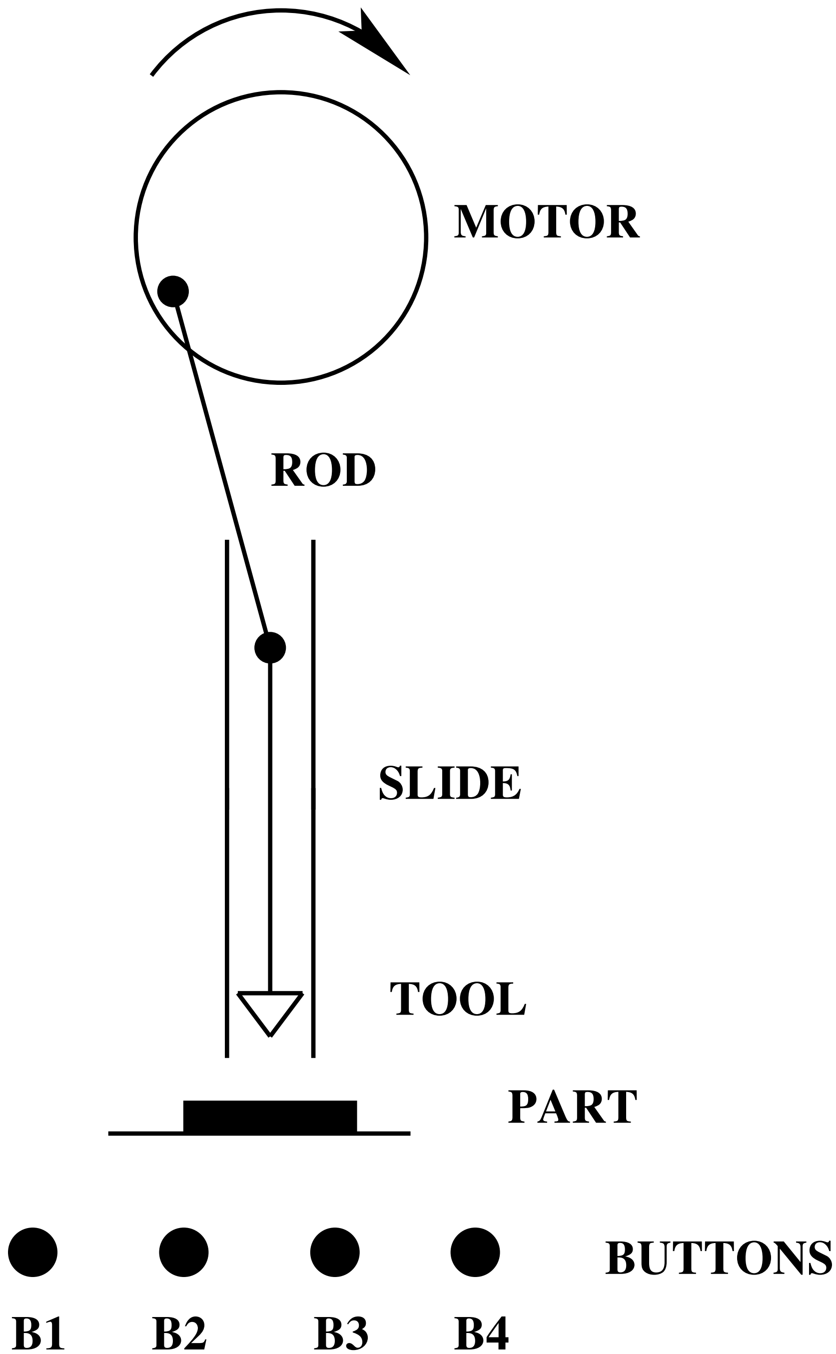
- A mechanical press controller

- Adapted from a real system

- The real system is coming from INRST:

Institut National de la Recherche sur la Sécurité du Travail

Mechanical Press Schema



Basic Equipment

- A Vertical Slide with a tool at its lower extremity

- An electrical Rotating Motor

- A Rod connecting the motor to the slide.

- A Clutch engaging or disengaging the motor on the rod

- When the clutch is disengaged, the slide stops “immediately”

Basic Commands

- Button B1: start motor

- Button B2: stop motor

- Button B3: engage clutch

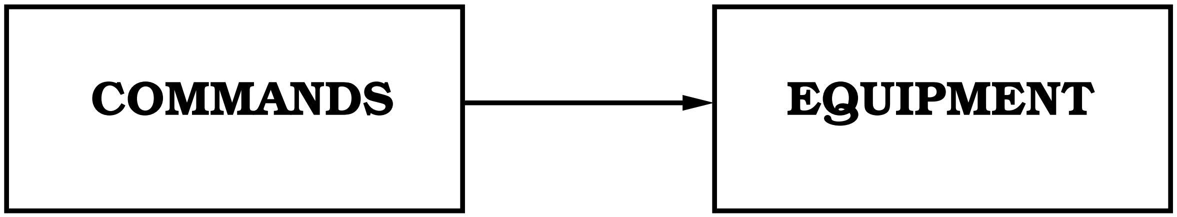
- Button B4: disengage clutch

Basic User Actions

- Action 1: Change the tool at the lower extremity of the slide

- Action 2: Put a part to be treated under the slide

- Action 3: Remove the part


COMMANDS
EQUIPMENT

A Typical User Session

1. start the motor (button B1)
2. change the tool (action 1)
3. put a part (action 2),
4. engage the clutch (button B3): the press now works,
5. disengage the clutch (button B4): the press does not work,
6. remove the part (action 3),
7. repeat zero or more times steps 3 to 6,
8. repeat zero or more times steps 2 to 7,
9. stop the motor (button B2).

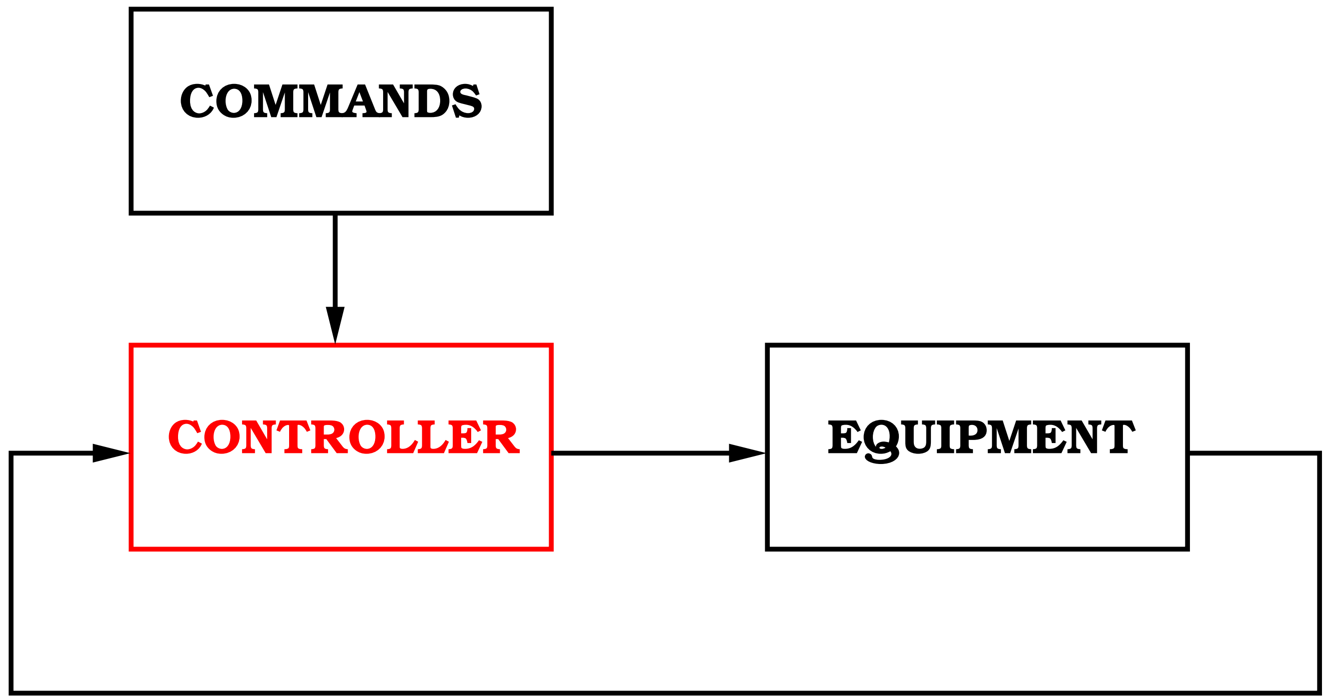
Danger: Necessity of a Controller

- step 2 (change the tool),

- step 3 (put a part),

- step 6 (remove the part) are all DANGEROUS

Second Schematic View

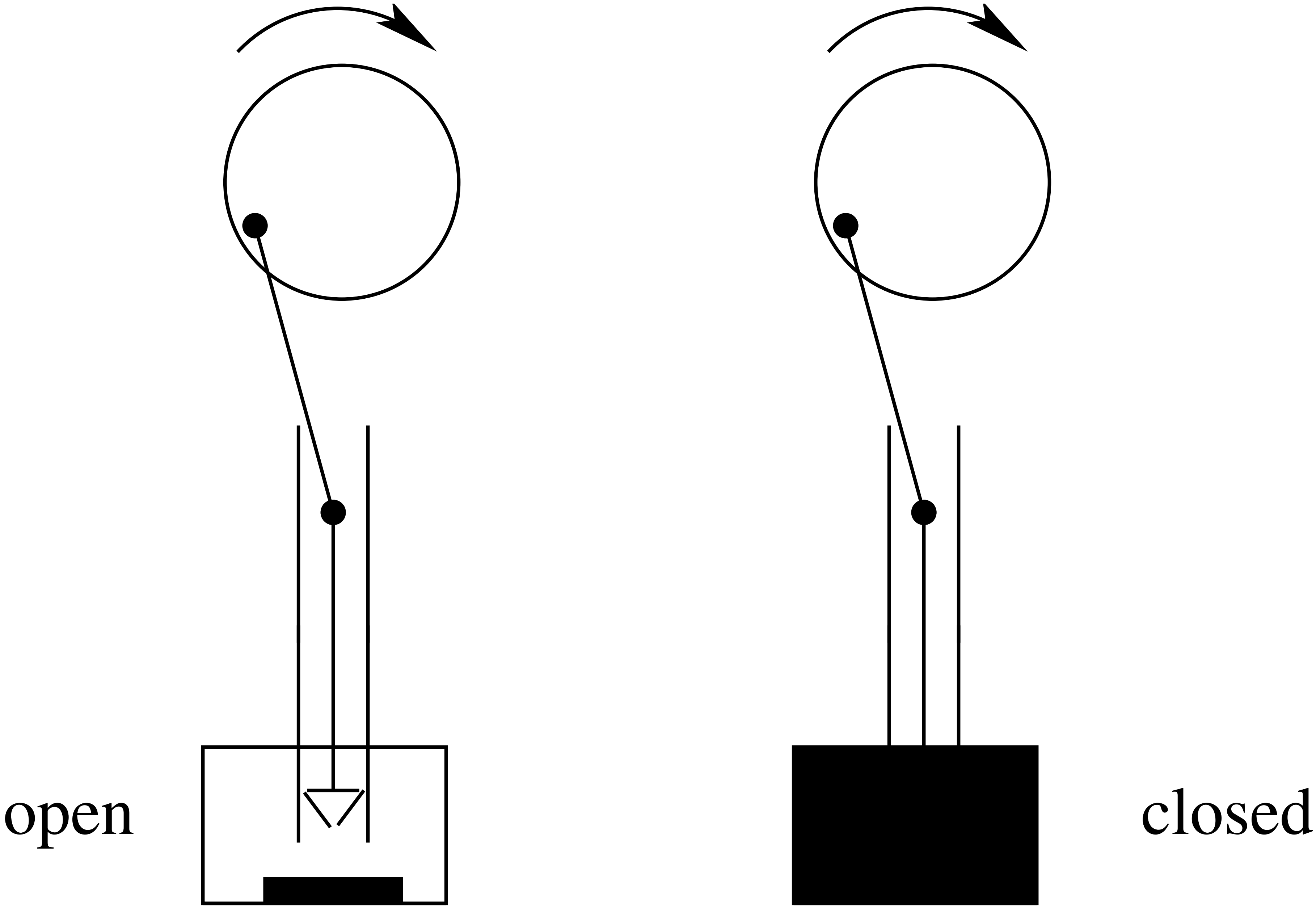


More Elaborate Commands for Protecting the User

- Controlling the way the clutch is engaged or disengaged

- Protection by means of a Front Door

The Front Door



The Front Door: Behavior

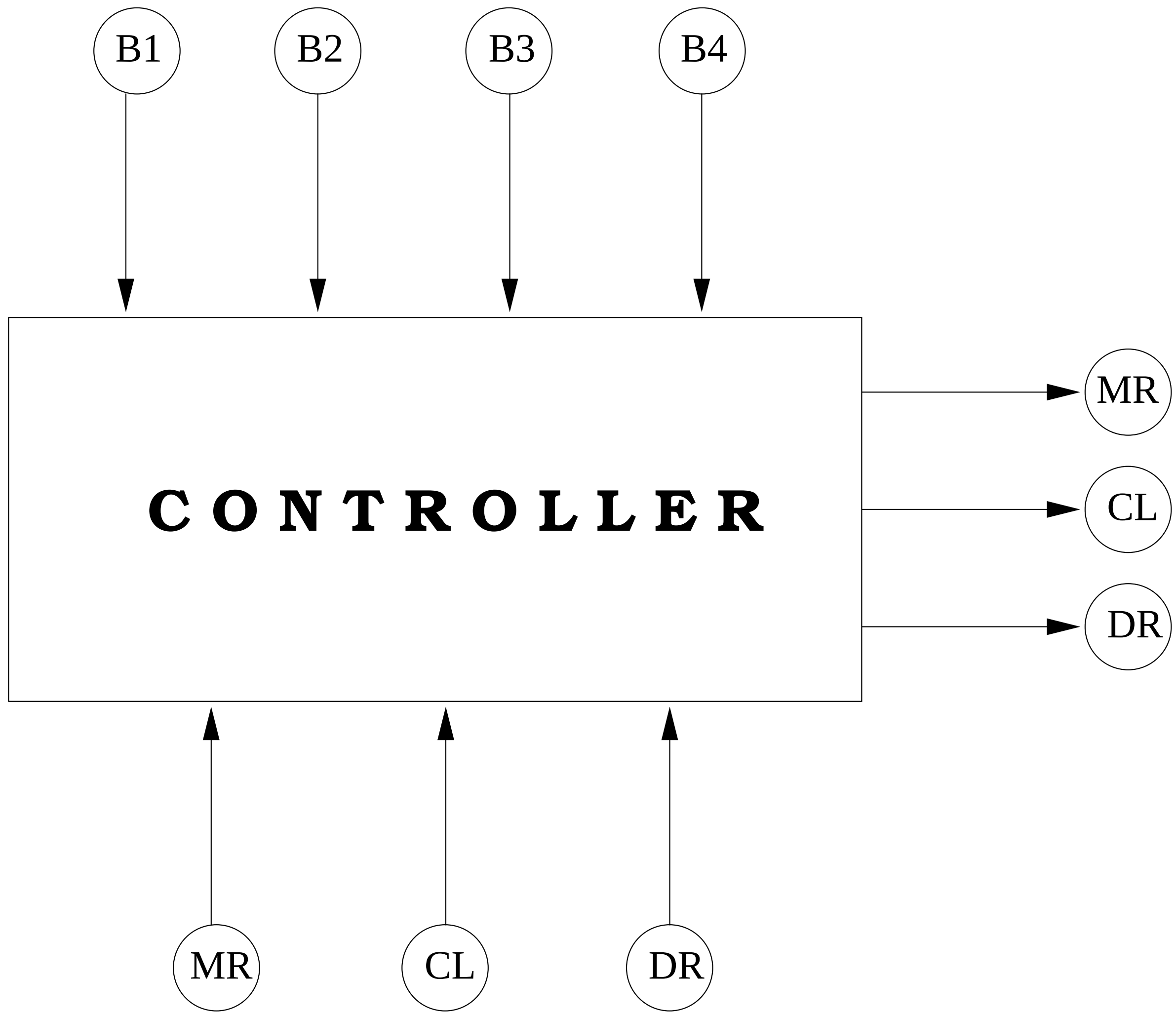
- Initially, the door is open

- When the user presses button B3 to engage the clutch,
  the door is first closed BEFORE engaging the clutch

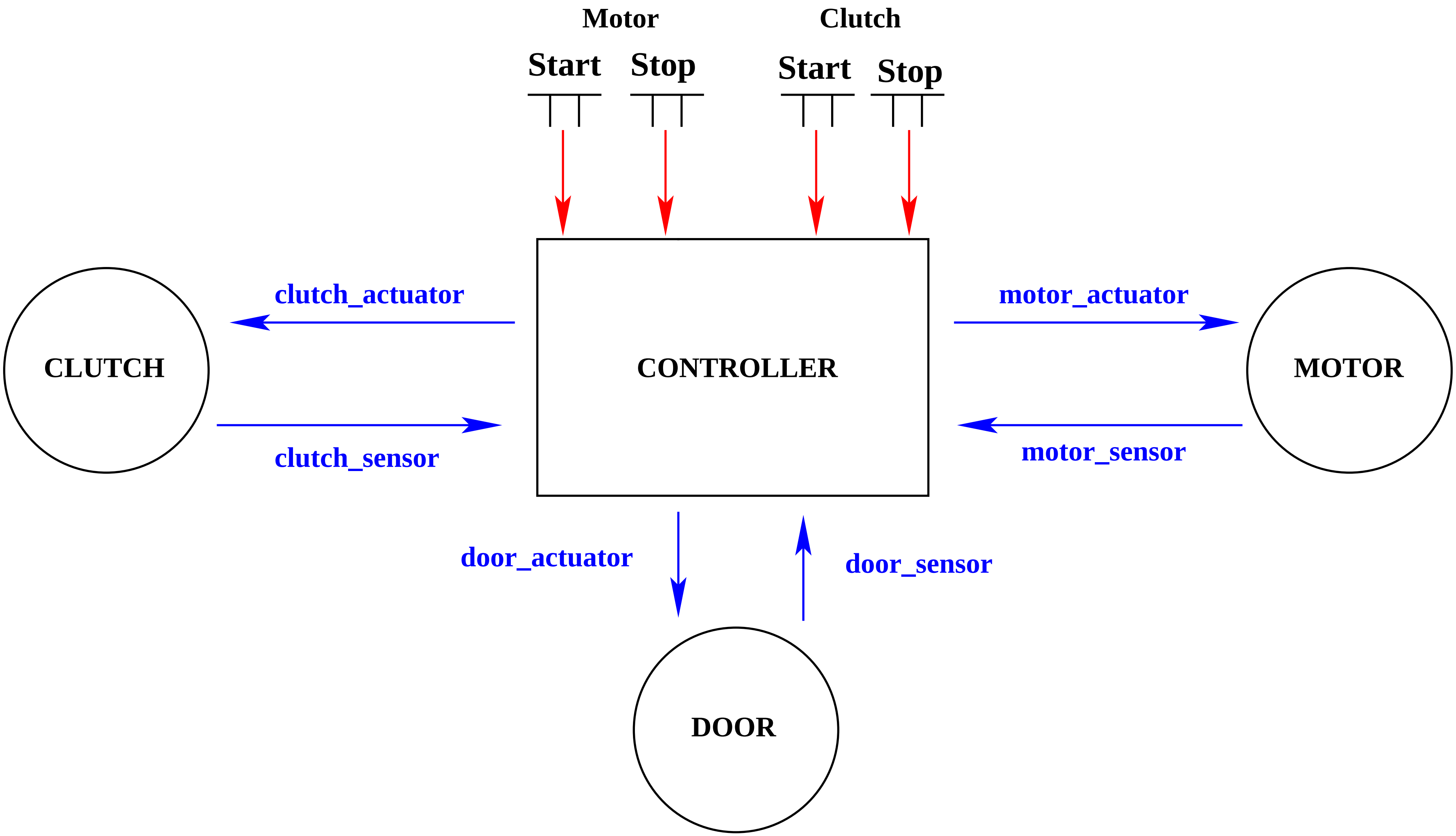
- When the user presses button B4 to disengage the clutch,
  the door is opened AFTER disengaging the clutch

- Notice: The door has no button.

Summary of Connections



Overview


Motor
Start
Stop
Clutch
Start
Stop
clutch_actuator
clutch_sensor
CLUTCH
CONTROLLER
motor_actuator
motor_sensor
MOTOR
door_actuator
door_sensor
DOOR

2. Presentation of some Design Patterns

Motivations

- A number of similar behaviors

- Some complex situations to handle

Similar Behavior (1)

- A specific action results eventually in having a specific reaction:

    - Pushing button B1 results eventually in starting the motor

    - Pushing button B4 results eventually in disengaging the clutch

    - . . .

Similar Behavior (2)

- Correlating two pieces of equipment:
    - When the clutch is engaged then the motor must work
    - When the clutch is engaged then the door must be closed

Similar Behavior (3)

- Making an action dependent of another one:

- Engaging the clutch implies closing the door first

- Disengaging the clutch means opening the door afterwards

Motivation: Example of Some Complex Situation

- Here is a sequence of events:

(1) User pushes button B1 (start motor)

(1') User does not remove his finger from button B1

(2) Controller sends the starting command to the motor

(3) Motor starts and sends feedback to the controller

(4) Controller is aware that the motor works

(5) User pushes button B2 (stop motor)

(6) Controller sends the stop command to the motor

(7) Motor stops and sends feedback to the controller

(8) Controller is aware that the motor does not work

(9) Controller must not send the starting command to the motor

Motivation: Example of Some Complex Situation

- Here is a sequence of events:

  (1) User pushes button B1 (start motor)

  (2) Controller sends the starting command to the motor

  (3.1) Motor starts and sends feedback to the controller

  (3.2) User pushes button B2 (stop motor)

- (3.1) and (3.2) may occur simultaneously

- If controller treats (3.1) before (3.2): motor is stopped

- If controller treats (3.2) before (3.1): motor is not stopped

- We want to build systems which are correct by construction

- We want to have more methods for doing so

- "Design pattern" is an Object Oriented concept

- We would like to borrow this concept for doing formal developments

- A preliminary tentative with reactive system developments

- Advantage: systematic developments and also refinement of proofs

A Personal View on Design Patterns

- This is an engineering concept

- It can be used outside OO

- The goal of each DP is to solve a certain category of problems

- But the design pattern has to be adapted to the problem at hand

- Is it compatible with formal developments?

- Let’s apply this approach to the design of reactive systems

A Wikipedia View on Design Patterns (1)

- A design pattern isn't a finished design that can be transformed into code

- It is a template for how to solve a problem that can be used in many different situations

- Patterns originated as an architectural concept by Christopher Alexander

- "Design Patterns: Elements of Reusable Object-Oriented Software" published in 1994 (Gamma et al)

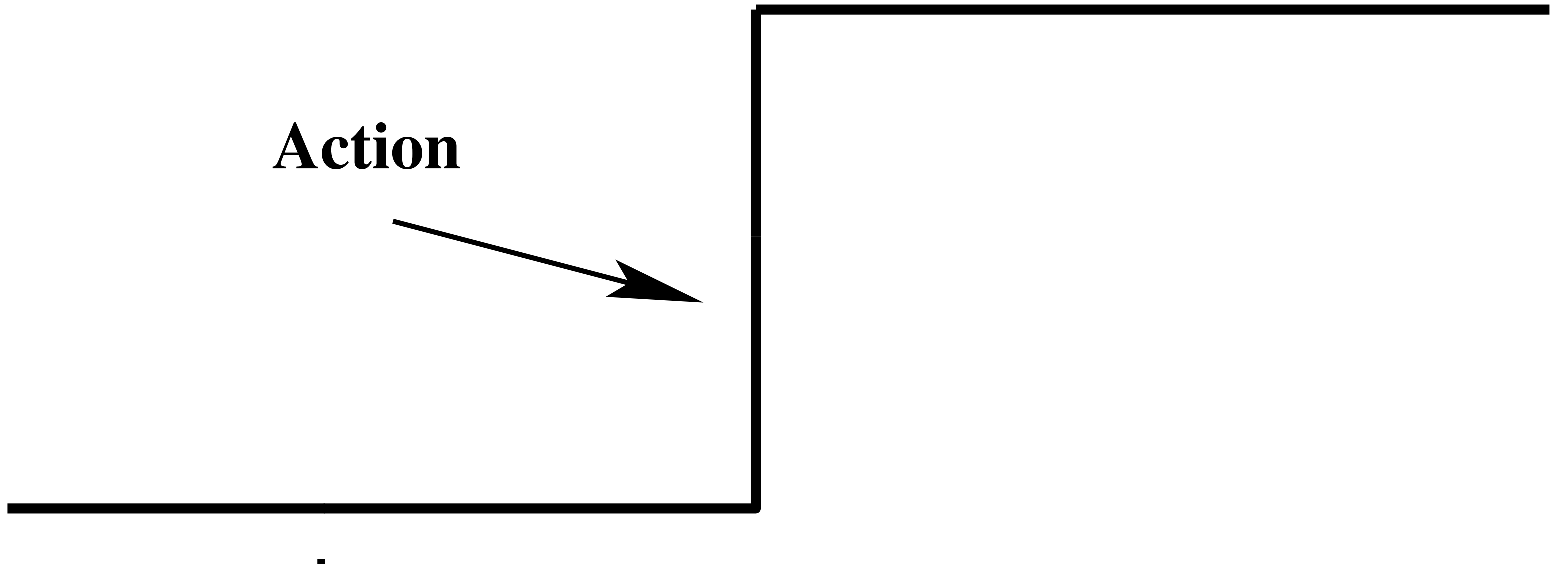
A Wikipedia View on Design Patterns (2)

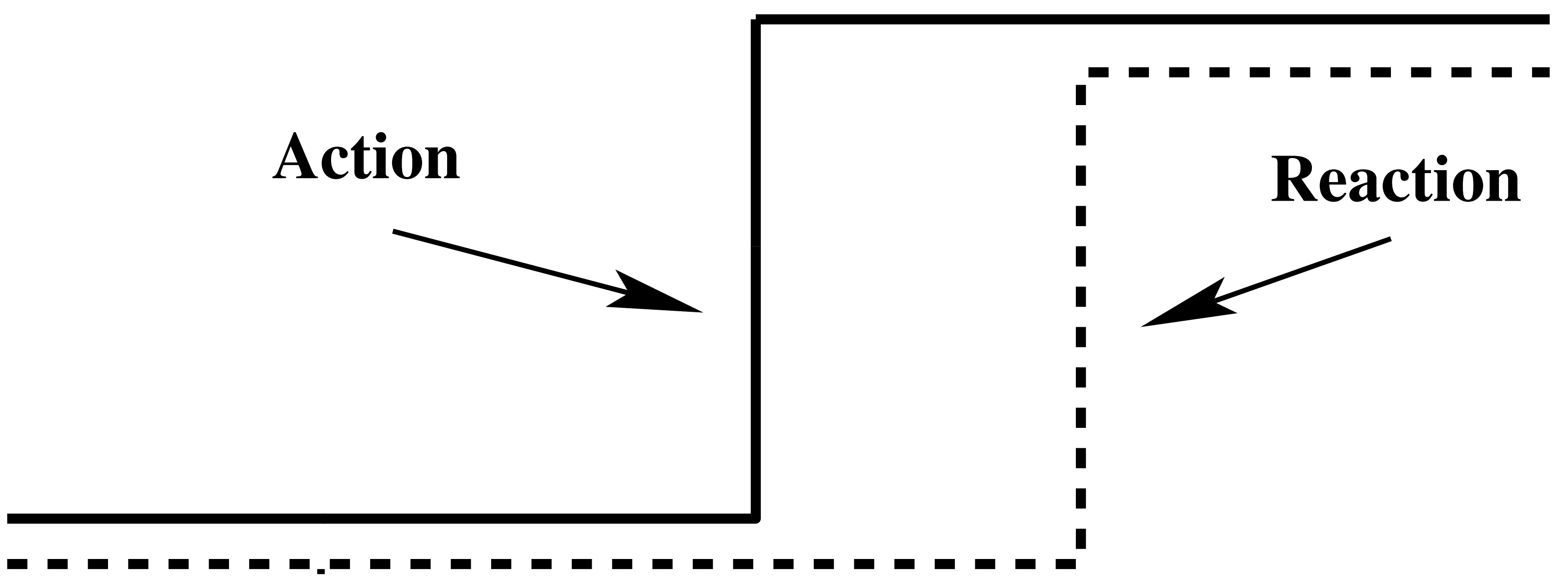
- Design pattern can speed up the development process by providing tested and proven development paradigms

- The documentation for a design pattern should contain enough information about the problem that the pattern addresses, the context in which it is used, and the suggested solution.

- Some feel that the need for patterns results from using computer languages or techniques with insufficient abstraction

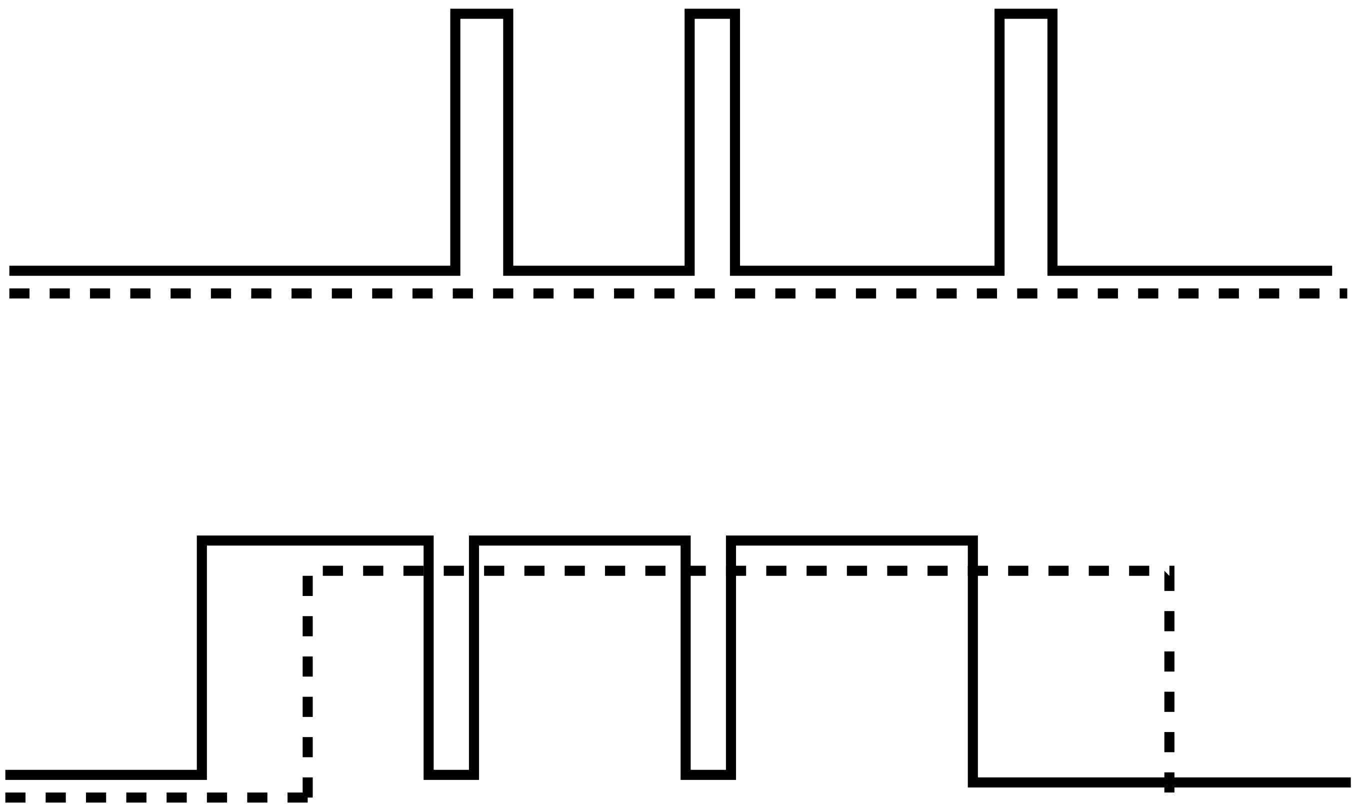
An Action Pattern




Action
Reaction

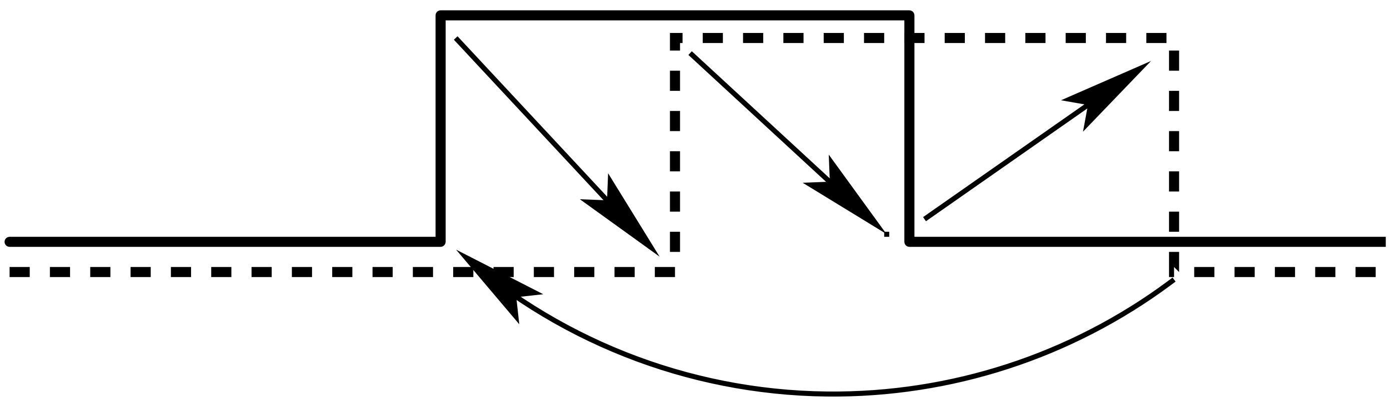
- Sometimes, the reaction has not enough time to react

- Because the action moves too quickly



- Sometimes, the reaction always follows the action

- They are both synchronized



- We built first a model of a weak reaction

- The strong reaction will be a refinement of the weak one

Model for weak action and reaction: the State

variables: a
r
ca
cr

pat0_1: $a \in \{0, 1\}$

pat0_2: $r \in \{0, 1\}$

pat0_3: $ca \in \mathbb{N}$

pat0_4: $cr \in \mathbb{N}$

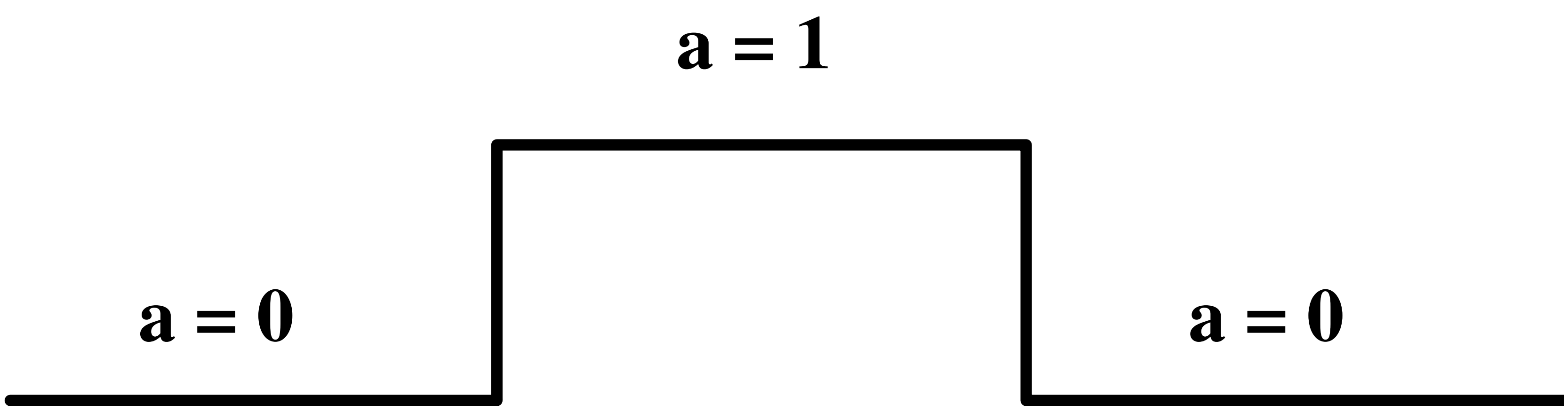
pat0_5: $cr \leq ca$

- a denotes the action
- r denotes the reaction
- ca and cr denote how many times a and r are set to 1
- **pat0_5** formalizes the weak reaction

Model for weak action and reaction: the Events (1)

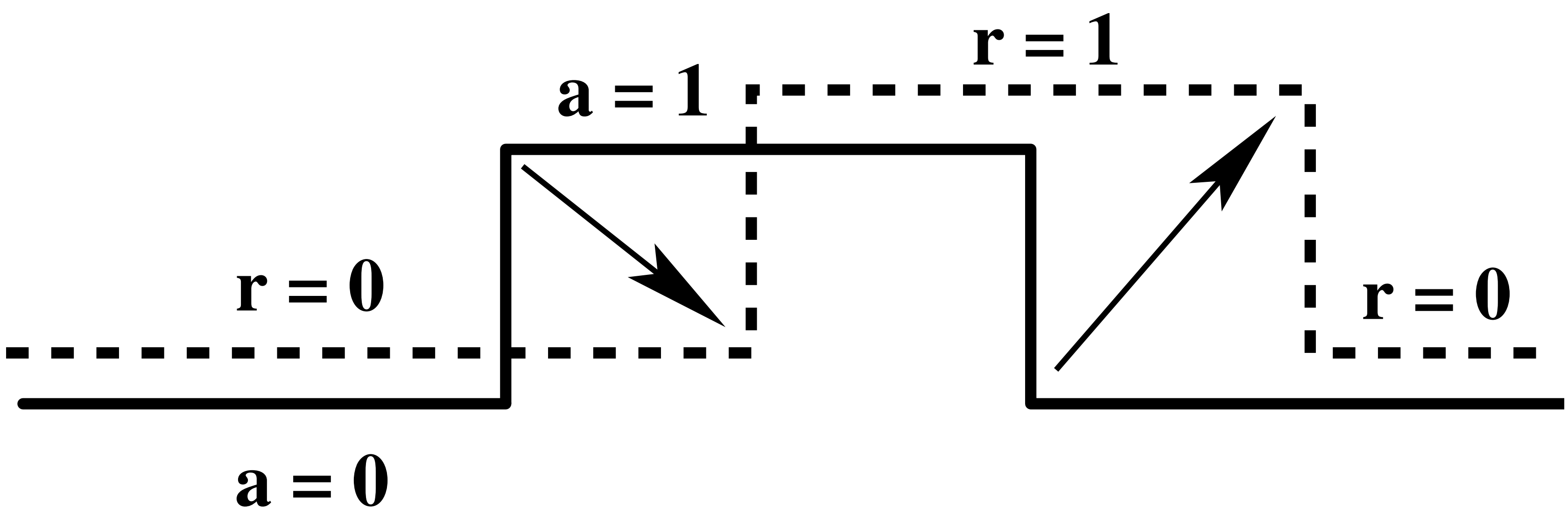
a_on
when
$a = 0$
then
$a := 1$
$ca := ca + 1$
end

a_off
when
$a = 1$
then
$a := 0$
end



r_on
when
$r = 0$
$a = 1$
then
$r := 1$
$cr := cr + 1$
end

r_off
when
$r = 1$
$a = 0$
then
$r := 0$
end



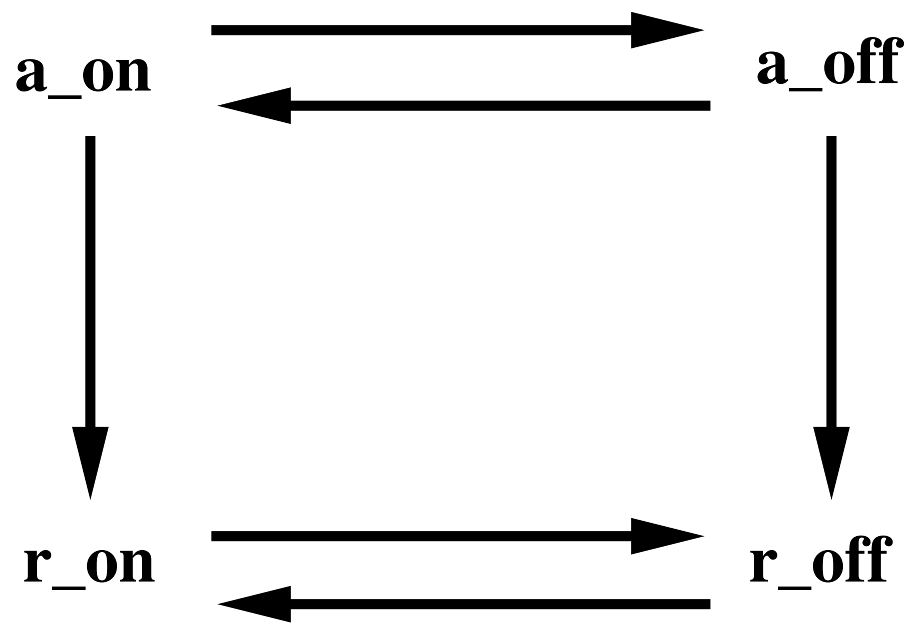
Summary of Events

a_on
when
$a = 0$
then
$a := 1$
$ca := ca + 1$
end

a_off
when
$a = 1$
then
$a := 0$
end

r_on
when
$r = 0$
$a = 1$
then
$r := 1$
$cr := cr + 1$
end

r_off
when
$r = 1$
$a = 0$
then
$r := 0$
end



Summary of Weak Synchronization

variables: $a,$
$r,$
$ca,$
cr

pat0_1: $a \in \{0,1\}$

pat0_2: $r \in \{0,1\}$

pat0_3: $ca \in \mathbb{N}$

pat0_4: $cr \in \mathbb{N}$

pat0_5: $cr \leq ca$

init
$a := 0$
$r := 0$
$ca := 0$
$cr := 0$

a_on
when
$a = 0$
then
$a := 1$
$ca := ca + 1$
end

a_off
when
$a = 1$
then
$a := 0$
end

r_on
when
$r = 0$
$a = 1$
then
$r := 1$
$cr := cr + 1$
end

r_off
when
$r = 1$
$a = 0$
then
$r := 0$
end

Nothing guarantees that the invariants are preserved

Invariant Preservation Statement by an Event

Invariants Guards of the event $\vdash$ Modified Invariant	INV

This is called a

Proof Obligation Rule

- The rule takes the form of a sequent

- A sequent is made of:
  - an antecedent containing zero or more assumptions
  - a consequent containing a predicate to prove

Naming the Proof Obligation

Invariants Guards of the event $\vdash$ Modified Invariant	INV

This is called a

Proof Obligation Rule

- We have 5 invariants: **pat0_1** to **pat0_5**

- We have 4 events: a_on, a_off, r_on, and r_off

- This makes 20 Proof Obligations

- Naming conventions: event-name / invariant-name / INV

Invariant Preservation Statement by an Event

Invariants Guards of the event $\vdash$ Modified Invariant	INV

POs are generated

by a tool: the POG

a_on
when
$a = 0$
then
$a := 1$
$ca := ca + 1$
end

$a \in \{0,1\}$ $r \in \{0,1\}$ $ca \in \mathbb{N}$ $cr \in \mathbb{N}$ $cr \leq ca$ $a = 0$ $\vdash$ $1 \in \{0,1\}$	a_on / **pat0_1** / INV

Preservation of invariant **pat0_1** by event a_on

One Proof Fails: r_on/pat0_5/INV

- The preservation proof of **pat0_5** ($cr \le ca$) by event r_on fails

```
r_on
  when
    r = 0
    a = 1
  then
    r := 1
    cr := cr + 1
  end
```

$\dots$	$\dots$
$cr \le ca$	**pat0_5**
$r = 0$	guard of r_on
$a = 1$	guard of r_on
$\vdash$	
$cr + 1 \le ca$	modified **pat0_5**

One Proof Fails: r_on/pat0_5/INV

- The preservation proof of **pat0_5** ($cr \leq ca$) by event r_on fails

```
r_on
  when
    r = 0
    a = 1
  then
    r := 1
    cr := cr + 1
  end
```

$\ldots$	$\ldots$
$cr \leq ca$	**pat0_5**
$r = 0$	guard of r_on
$a = 1$	guard of r_on
$\vdash$	
$cr + 1 \leq ca$	modified **pat0_5**

- We have to add the assumption $cr < ca$ in our sequent

One Proof Fails: r_on/pat0_5/INV

- The preservation proof of **pat0_5** ($cr \le ca$) by event r_on fails

```
r_on
  when
    r = 0
    a = 1
  then
    r := 1
    cr := cr + 1
  end
```

$\ldots$	$\ldots$
$cr \le ca$	**pat0_5**
$r = 0$	guard of r_on
$a = 1$	guard of r_on
$\vdash$	
$cr + 1 \le ca$	modified **pat0_5**

- We have to add the assumption $cr < ca$ in our sequent

- Two solutions: strengthening the guard or adding a new invariant

```
r_on
  when
    r = 0
    a = 1
    cr < ca
  then
    r := 1
    cr := cr + 1
  end
```

$\ldots$	$\ldots$
$cr \leq ca$	**pat0_5**
$r = 0$	guard of r_on
$a = 1$	guard of r_on
$cr < ca$	new guard of r_on
$\vdash$	
$cr + 1 \leq ca$	modified **pat0_5**

- Drawback: One has to keep variables cr and ca in the guard

- These variables were introduced for the modelling only

First Solution: Adding an Invariant

- We cannot introduce invariant $cr < ca$ directly (it does not hold)

- We introduce an implicative invariant

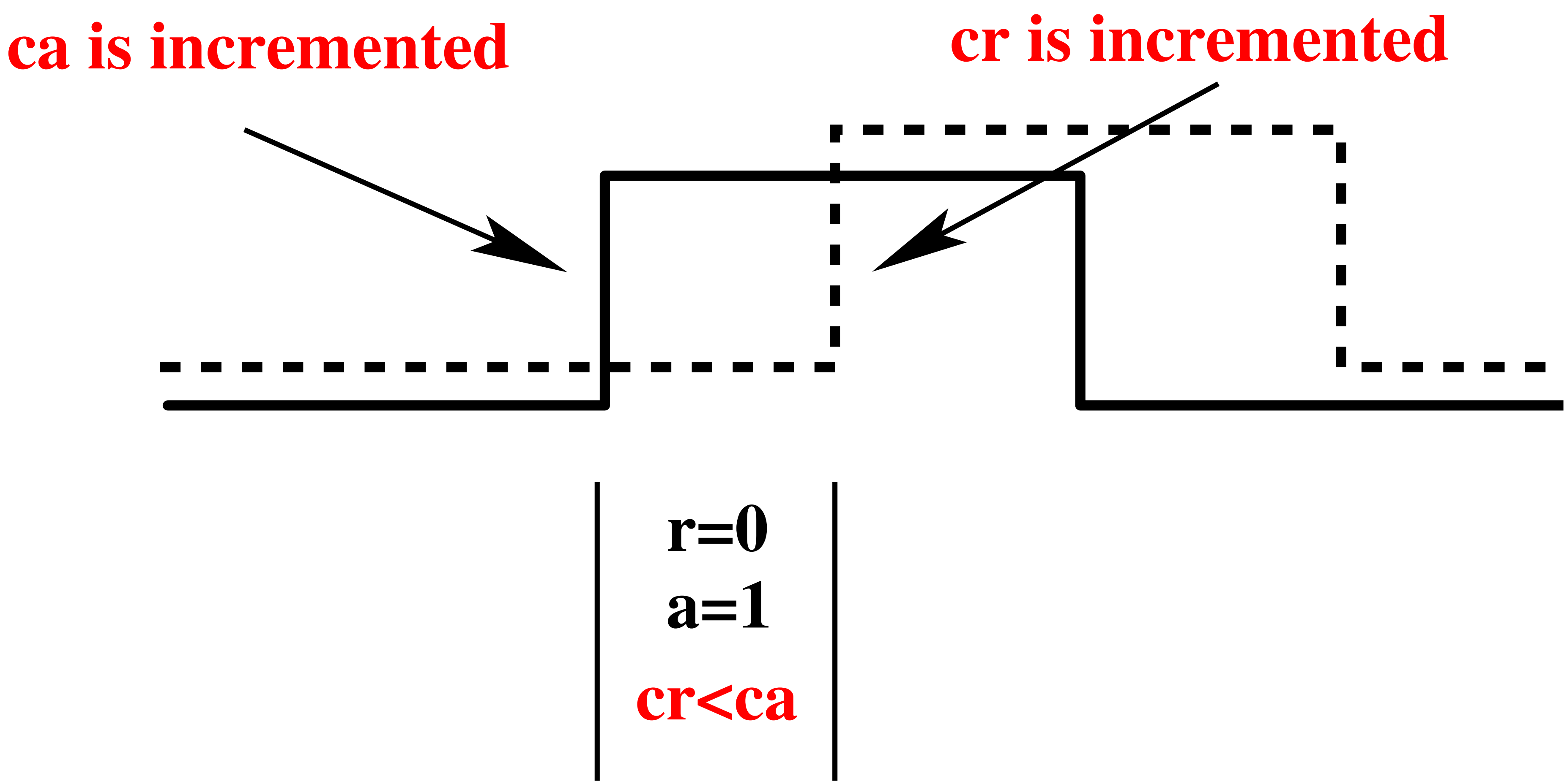
pat0_6: $r = 0 \;\wedge\; a = 1 \;\Rightarrow\; cr < ca$

```
r_on
  when
    r = 0
    a = 1
  then
    r := 1
    cr := cr + 1
  end
```

$\dots$	$\dots$
$cr \leq ca$	**pat0_5**
$r = 0 \;\wedge\; a = 1 \;\Rightarrow\; cr < ca$	**pat0_6**
$r = 0$	guard of r_on
$a = 1$	guard of r_on
$\vdash$	
$cr + 1 \leq ca$	modified **pat0_5**

- Drawback: One has to prove that this new invariant is maintained

pat0_6: $r = 0 \wedge a = 1 \Rightarrow cr < ca$



Problems with the New Invariant Preservation

pat0_6: $r = 0 \ \wedge \ a = 1 \ \Rightarrow \ cr < ca$

a_on
  when
    $a = 0$
  then
    $a := 1$
    $ca := ca + 1$
  end

a_off
  when
    $a = 1$
  then
    $a := 0$
  end

r_on
  when
    $r = 0$
    $a = 1$
  then
    $r := 1$
    $cr := cr + 1$
  end

r_off
  when
    $r = 1$
    $a = 0$
  then
    $r := 0$
  end

- No problem with a_on since ca is incremented

- No problem with a_off since a becomes 0

- No problem with r_on since r becomes 1

- No problem with r_off since $a = 0$ (guard)

pat0_6: $r = 0 \ \wedge \ a = 1 \ \Rightarrow \ cr < ca$

The preservation of this invariant by r_on leads to proving:

```
r_on
  when
    r = 0
    a = 1
  then
    r := 1
    cr := cr + 1
  end
```

$$
\begin{array}{l}
r = 0 \ \wedge \ a = 1 \ \Rightarrow \ cr < ca \\
r = 0 \\
a = 1 \\
\vdash \\
1 = 0 \ \wedge \ a = 1 \ \Rightarrow \ cr + 1 < ca
\end{array}
$$

which holds trivially

Summary of the State of the weak Reaction

pat0_1:	$a \in \{0, 1\}$
pat0_2:	$r \in \{0, 1\}$
pat0_3:	$ca \in \mathbb{N}$
pat0_4:	$cr \in \mathbb{N}$
pat0_5:	$cr \leq ca$
pat0_6:	$r = 0 \land a = 1 \Rightarrow cr < ca$

The counters have been removed

init
$a := 0$
$r := 0$

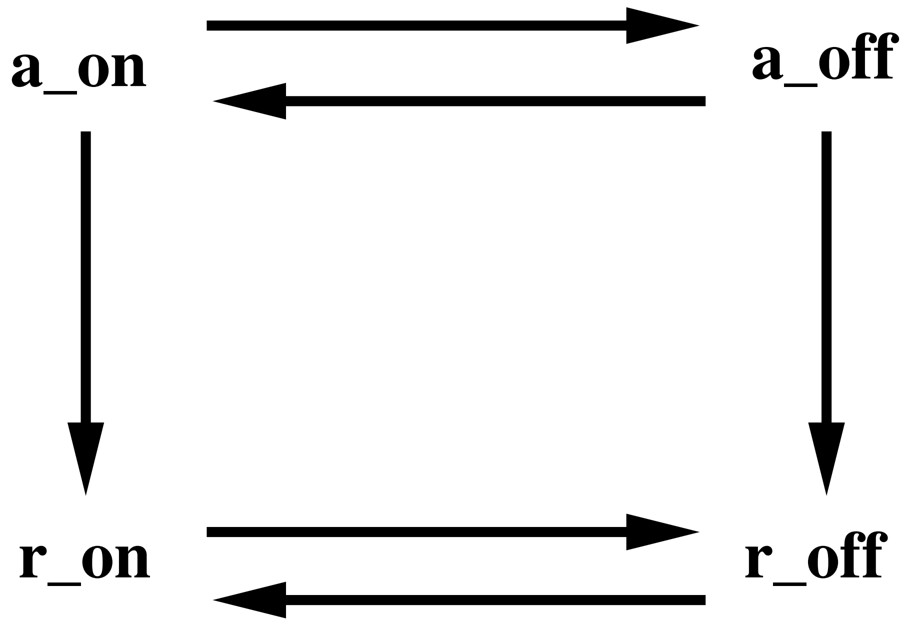
a_on
when
$a = 0$
then
$a := 1$
end

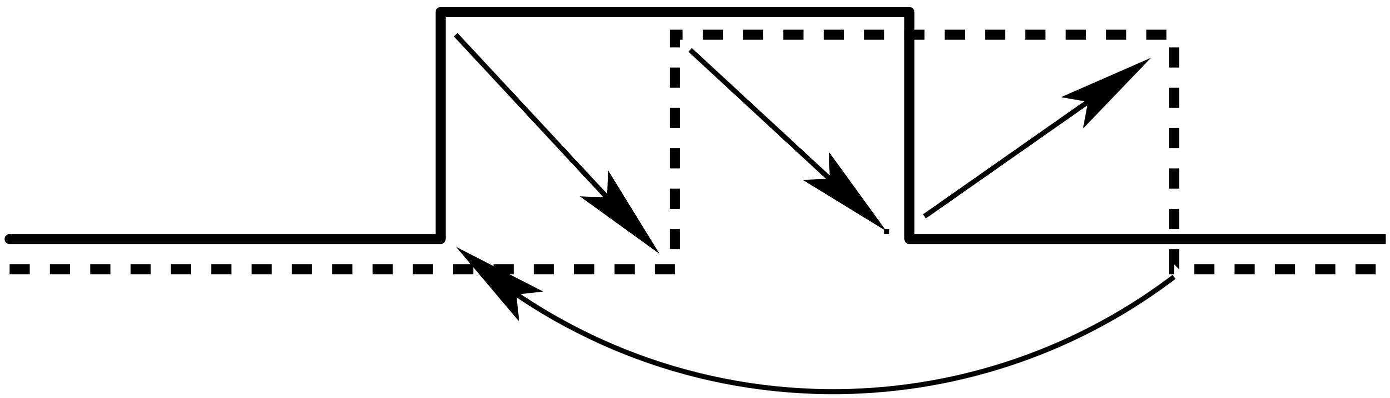
a_off
when
$a = 1$
then
$a := 0$
end

r_on
when
$r = 0$
$a = 1$
then
$r := 1$
end

r_off
when
$r = 1$
$a = 0$
then
$r := 0$
end

Weak Synchronization of Events





- We add the following invariant

pat1_1: $ca \leq cr + 1$

- Remember invariant **pat0_5**

pat0_5: $cr \leq ca$

We have thus: $cr \leq ca \leq cr + 1$

pat1_1: $ca \leq cr + 1$

a_on
when
$a = 0$
then
$a := 1$
$ca := ca + 1$
end

a_off
when
$a = 1$
then
$a := 0$
end

r_on
when
$r = 0$
$a = 1$
then
$r := 1$
$cr := cr + 1$
end

r_off
when
$r = 1$
$a = 0$
then
$r := 0$
end

- Problem with a_on since ca is incremented
- No problem with a_off since no changes
- No problem with r_on since cr is incremented
- No problem with r_off since no changes

One of the Invariant Preservation Proofs Fails

- Event a_on cannot maintain **pat1_1**

```
a_on
  when
    a = 0
  then
    a := 1
    ca := ca + 1
  end
```

$cr \leq ca$	**pat0_5**
$ca \leq cr + 1$	**pat1_1**
$a = 0$	guard of a_on
$\vdash$	
$ca + 1 \leq cr + 1$	modified **pat1_1**

One of the Invariant Preservation Proofs Fails

- Event a_on cannot maintain **pat1_1**

```
a_on
  when
    a = 0
  then
    a := 1
    ca := ca + 1
  end
```

$cr \leq ca$	**pat0_5**
$ca \leq cr + 1$	**pat1_1**
$a = 0$	guard of a_on
$\vdash$	
$ca + 1 \leq cr + 1$	modified **pat1_1**

- We need the assumption $ca \leq cr$

- Event a_on cannot maintain **pat1_1**

```
a_on
  when
    a = 0
  then
    a := 1
    ca := ca + 1
  end
```

$cr \leq ca$	**pat0_5**
$ca \leq cr + 1$	**pat1_1**
$a = 0$	guard of a_on
$\vdash$	
$ca + 1 \leq cr + 1$	modified **pat1_1**

- We need the assumption $ca \leq cr$

- But we already have assumption $cr \leq ca$ (this is **pat0_5**)

One of the Invariant Preservation Proofs Fails

- Event a_on cannot maintain **pat1_1**

a_on **when** $a = 0$ **then** $a := 1$ $ca := ca + 1$ **end**

$cr \leq ca$	**pat0_5**
$ca \leq cr + 1$	**pat1_1**
$a = 0$	guard of a_on
$\vdash$	
$ca + 1 \leq cr + 1$	modified **pat1_1**

- We need the assumption $ca \leq cr$

- But we already have assumption $cr \leq ca$ (this is **pat0_5**)

- Thus we need the assumption $cr = ca$

- Event a_on cannot maintain **pat1_1**

a_on
  when
    $a = 0$
  then
    $a := 1$
    $ca := ca + 1$
  end

$cr \leq ca$	**pat0_5**
$ca \leq cr + 1$	**pat1_1**
$a = 0$	guard of a_on
$\vdash$	
$ca + 1 \leq cr + 1$	modified **pat1_1**

- We need the assumption $ca \leq cr$

- But we already have assumption $cr \leq ca$ (this is **pat0_5**)

- Thus we need the assumption $cr = ca$

- This suggests the new invariant: $a = 0 \;\; \Rightarrow \;\; cr = ca$

Problems with the New Invariant Preservation

pat1_2: $a = 0 \Rightarrow cr = ca$

```
a_on
  when
    a = 0
  then
    a := 1
    ca := ca + 1
  end
```

```
a_off
  when
    a = 1
  then
    a := 0
  end
```

```
r_on
  when
    r = 0
    a = 1
  then
    r := 1
    cr := cr + 1
  end
```

```
r_off
  when
    r = 1
    a = 0
  then
    r := 0
  end
```

- No problem with a_on since a becomes 1

- Problem with a_off since a becomes 0

- No problem with r_on since $a = 1$ (guard)

- No problem with r_off since no changes

Proposing a new invariant

pat1_2: $a = 0 \Rightarrow cr = ca$

- The proof of maintenance of this invariant by a_off fails

```
a_off
  when
    a = 1
  then
    a := 0
  end
```

$a = 0 \Rightarrow cr = ca$	**pat1_2**
$a = 1$	guard of a_off
$\vdash$	
$0 = 0 \Rightarrow cr = ca$	modified **pat1_2**

- This suggests a new invariant:

$$a = 1 \Rightarrow cr = ca$$

We need: $a = 1 \Rightarrow cr = ca$

- But we already have the following:

pat0_6: $a = 1 \wedge r = 0 \Rightarrow cr < ca$

This suggests the following:

pat1_3: $a = 1 \wedge r = 1 \Rightarrow cr = ca$

- In order for a_off to prove **pat1_2** ($a = 0 \;\Rightarrow\; cr = ca$)

a_off **when** $a = 1$ **then** $a := 0$ **end**	$a = 0 \;\Rightarrow\; cr = ca$ — **pat1_2** $a = 1 \,\wedge\, r = 1 \;\Rightarrow\; cr = ca$ — **pat1_3** $a = 1$ — guard $\vdash$ $0 = 0 \;\Rightarrow\; cr = ca$ — modified **pat1_2**

- We need to strengthen its guard because of **pat1_3**:

- In order for a_off to prove **pat1_2** ($a = 0 \;\Rightarrow\; cr = ca$)

```
a_off
  when
    r = 1
    a = 1
  then
    a := 0
  end
```

$a = 0 \;\Rightarrow\; cr = ca$	**pat1_2**
$a = 1 \;\wedge\; r = 1 \;\Rightarrow\; cr = ca$	**pat1_3**
$r = 1$	new guard
$a = 1$	guard
$\vdash$	
$0 = 0 \;\Rightarrow\; cr = ca$	modified **pat1_2**

- We need to strengthen its guard because of **pat1_3**:

Problems with the New Invariant Preservation

pat1_3: $a = 1 \wedge r = 1 \Rightarrow cr = ca$

a_on
when
$a = 0$
then
$a := 1$
$ca := ca + 1$
end

a_off
when
$r = 1$
$a = 1$
then
$a := 0$
end

r_on
when
$r = 0$
$a = 1$
then
$r := 1$
$cr := cr + 1$
end

r_off
when
$r = 1$
$a = 0$
then
$r := 0$
end

- Problem with a_on since a becomes 1 and ca is incremented
- No problem with a_off since a becomes 0
- Problem with r_on since $a = 1$ (guard), r becomes 1, and cr is incremented
- No problem with r_off since r becomes 0

One of the Proofs Fails

- Event a_on cannot maintain invariant **pat1_3**

$$a = 1 \;\land\; r = 1 \;\;\Rightarrow\;\; cr = ca$$

a_on
  when
    $a = 0$

  then
    $a := 1$
    $ca := ca + 1$
  end

$$a = 1 \;\land\; r = 1 \;\;\Rightarrow\;\; cr = ca$$
$$a = 0$$

$$\vdash$$

$$1 = 1 \;\land\; r = 1 \;\;\Rightarrow\;\; cr = ca + 1$$

- This suggest strengthening the guard of a_on: $r = 0$

One of the Proofs Fails

- Event a_on cannot maintain invariant **pat1_3**

$$a = 1 \ \land \ r = 1 \ \Rightarrow \ cr = ca$$

```
a_on
  when
    a = 0
    r = 0
  then
    a := 1
    ca := ca + 1
  end
```

$a = 1 \ \land \ r = 1 \ \Rightarrow \ cr = ca$
$a = 0$
$r = 0$ new guard
$\vdash$
$1 = 1 \ \land \ r = 1 \ \Rightarrow \ cr = ca + 1$

- This suggest strengthening the guard of a_on: $r = 0$

Problems with the New Invariant Preservation

pat1_3: $a = 1 \land r = 1 \Rightarrow cr = ca$

a_on	a_off	r_on	r_off
when $r = 0$, $a = 0$ **then** $a := 1$, $ca := ca + 1$ **end**	**when** $r = 1$, $a = 1$ **then** $a := 0$ **end**	**when** $r = 0$, $a = 1$ **then** $r := 1$, $cr := cr + 1$ **end**	**when** $r = 1$, $a = 0$ **then** $r := 0$ **end**

- No problem with a_on since $r = 0$ (guard)

- No problem with a_off since a becomes 0

- Problem with r_on since $a = 1$ (guard), r becomes 1, and cr is incremented

- No problem with r_off since r becomes 0

- Invariant **pat1_3**

$$a = 1 \;\wedge\; r = 1 \;\Rightarrow\; cr = ca$$

```
r_on
  when
    r = 0
    a = 1
  then
    r := 1
    cr := cr + 1
  end
```

$$a = 1 \;\wedge\; r = 1 \;\Rightarrow\; cr = ca$$
$$r = 0$$
$$a = 1$$
$$\vdash$$
$$a = 1 \;\wedge\; 1 = 1 \;\Rightarrow\; cr + 1 = ca$$

We have forgotten invariants **pat1_1** and **pat0_6**

- Invariant **pat1_3**

$$a = 1 \ \wedge \ r = 1 \ \Rightarrow \ cr = ca$$

```
r_on
  when
    r = 0
    a = 1
  then
    r := 1
    cr := cr + 1
  end
```

$$r = 0 \ \wedge \ a = 1 \ \Rightarrow \ cr < ca$$
$$ca \leq cr + 1$$
$$a = 1 \ \wedge \ r = 1 \ \Rightarrow \ cr = ca$$
$$r = 0$$
$$a = 1$$
$$\vdash$$
$$a = 1 \ \wedge \ 1 = 1 \ \Rightarrow \ cr + 1 = ca$$

- Everything is proved now

- We do not need to add more invariants

Summary of the Events for the Strong Reaction

The counters have

been removed

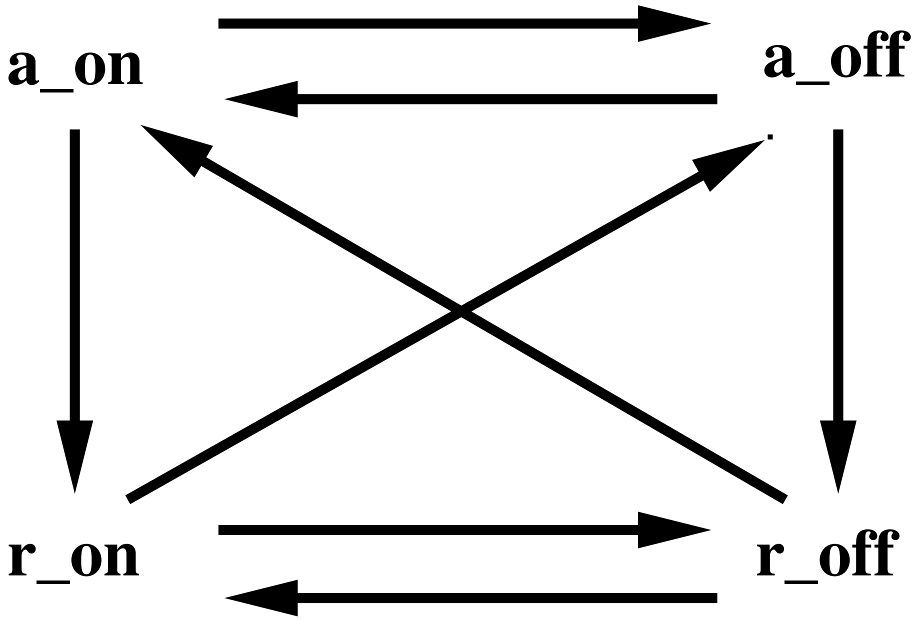
a_on
when
$a = 0$
$r = 0$
then
$a := 1$
end

a_off
when
$a = 1$
$r = 1$
then
$a := 0$
end

init
$a := 0$
$r := 0$

r_on
when
$r = 0$
$a = 1$
then
$r := 1$
end

r_off
when
$r = 1$
$a = 0$
then
$r := 0$
end


a_on
a_off
r_on
r_off

- Putting together these two invariants

pat1_2:	$a = 0 \;\Rightarrow\; ca = cr$
pat1_3:	$a = 1 \;\wedge\; r = 1 \;\Rightarrow\; cr = ca$

- leads to the following

pat1_4:	$a = 0 \;\vee\; r = 1 \;\Rightarrow\; ca = cr$

pat0_5:	$cr \leq ca$
pat0_6:	$a = 1 \ \wedge \ r = 0 \ \Rightarrow \ cr < ca$
pat1_1:	$ca \leq cr + 1$
pat1_4:	$a = 0 \ \vee \ r = 1 \ \Rightarrow \ ca = cr$

This can be simplified to

pat2_1:	$a = 1 \ \wedge \ r = 0 \ \Rightarrow \ ca = cr + 1$
pat2_2:	$a = 0 \ \vee \ r = 1 \ \Rightarrow \ ca = cr$

Summary of the State for the Strong Reaction

pat0_1: $a \in \{0, 1\}$

pat0_2: $r \in \{0, 1\}$

pat0_3: $ca \in \mathbb{N}$

pat0_4: $cr \in \mathbb{N}$

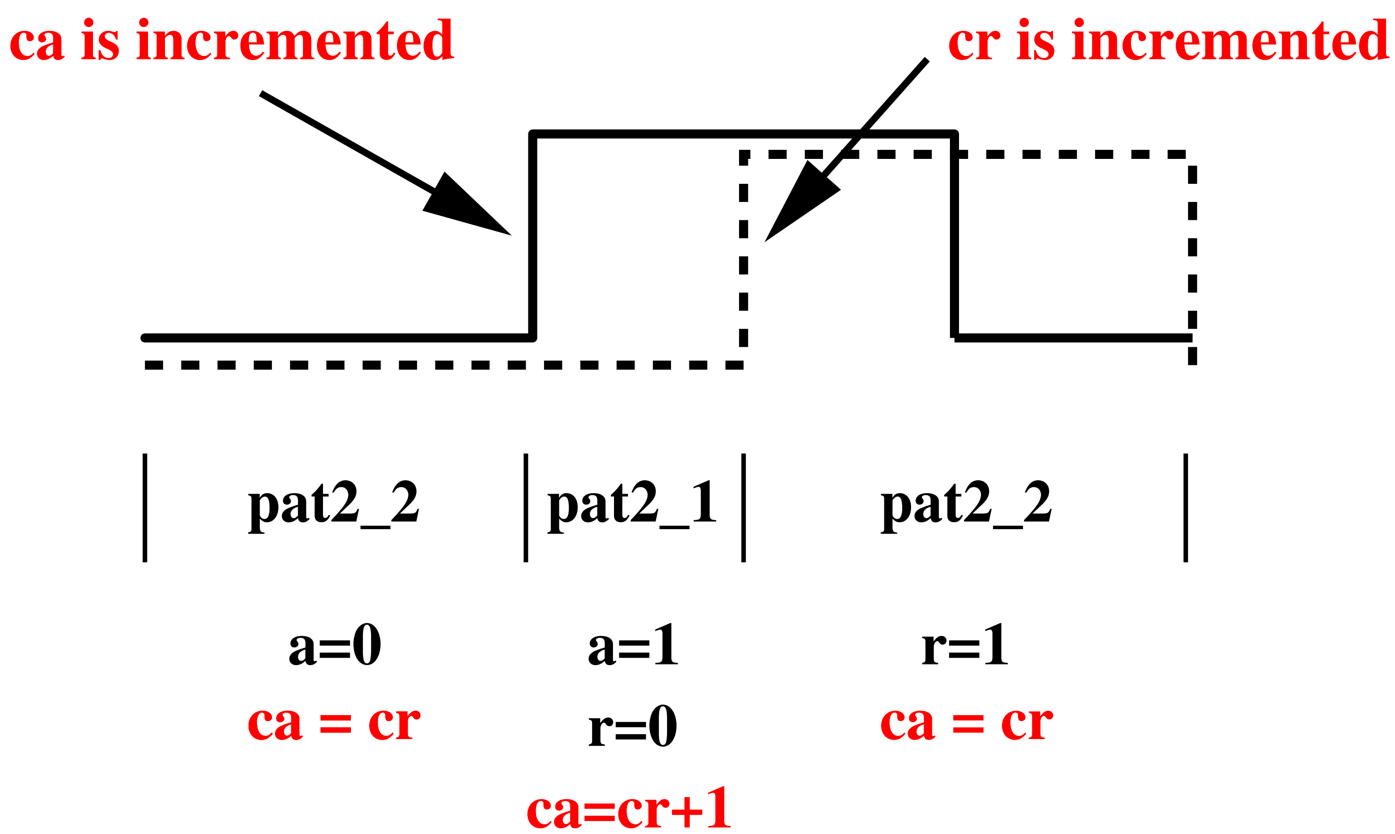
pat2_1: $a = 1 \ \land \ r = 0 \ \Rightarrow \ ca = cr + 1$

pat2_2: $a = 0 \ \lor \ r = 1 \ \Rightarrow \ ca = cr$

Comparing the two Invariants

pat2_1: $a = 1 \wedge r = 0 \Rightarrow ca = cr + 1$

pat2_2: $a = 0 \vee r = 1 \Rightarrow ca = cr$



What we Have Learned

- Proof failures helped us improving our models

- When an invariant preservation proof fails on an event,
  there are two solutions:
    - adding a new invariant
    - strengthening the guard

- Modelling considerations helped us choosing one or the other

- At the end, we reached a stable situation (fixpoint)

3. Writing the Requirement Document

Requirements: Describing Equipment

The system has got the following pieces of equipment: a Motor, a Clutch, and a Door	EQP_1
Four Buttons are used to start and stop the motor, and engage and disengage the clutch	EQP_2
A Controller is supposed to manage this equipment	EQP_3

Requirements: Connection Constraints

Buttons and Controller are weakly synchronized	FUN_1

Controller are Equipment are strongly synchronized	FUN_2

Requirements: Relationship Between Motor and Clutch

When the clutch is engaged, the motor must work	SAF_1

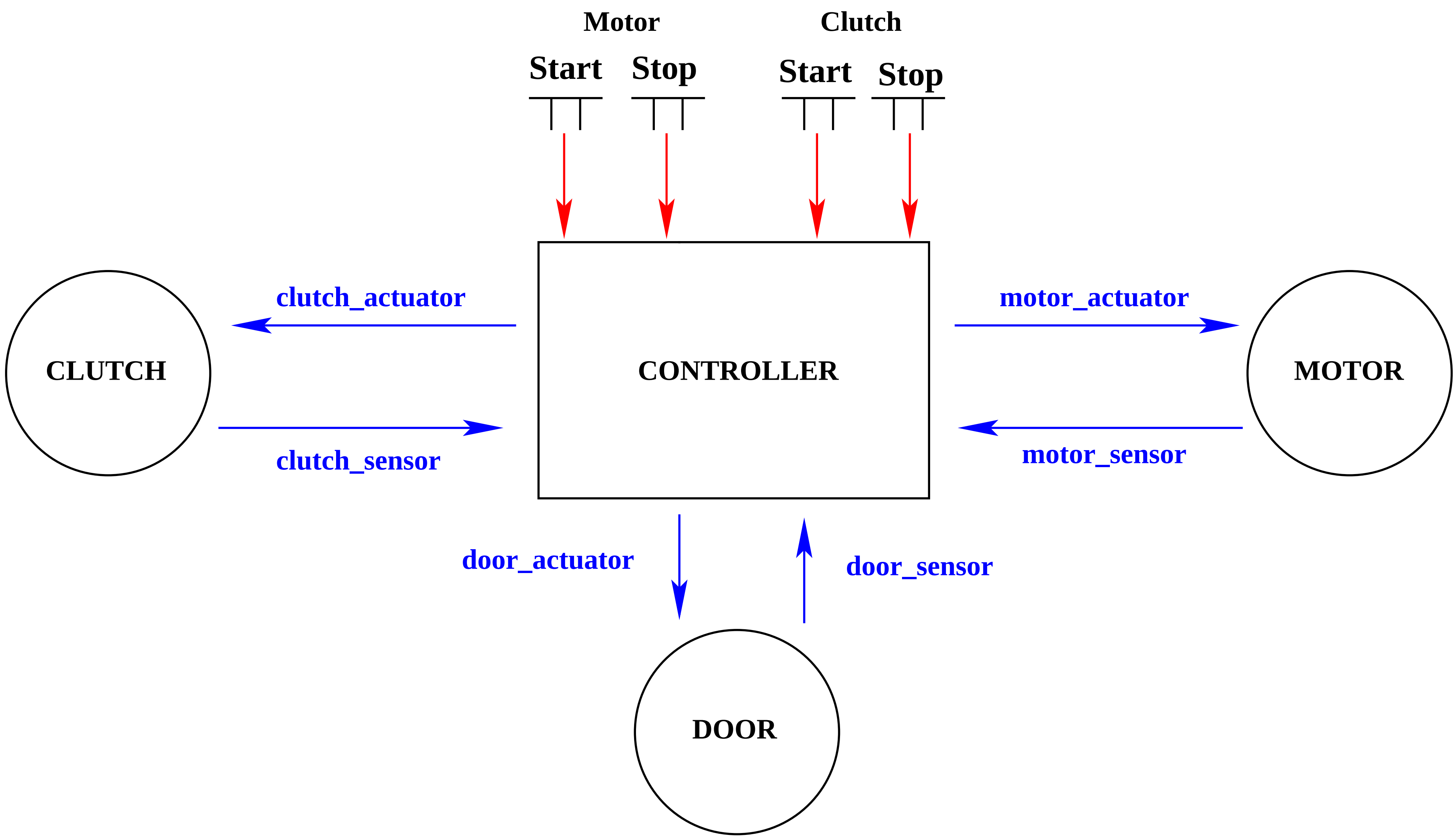
When the clutch is engaged, the door must be closed	SAF_2

Requirements: Relationship Between Door and Clutch

When the clutch is disengaged, the door cannot be closed several times, ONLY ONCE	FUN_3

When the door is closed, the clutch cannot be disengaged several times, ONLY ONCE	FUN_4

Opening and closing the door are not independent. It must be synchronized with disengaging and engaging the clutch	FUN_5


Motor
Start
Stop
Clutch
Start
Stop
clutch_actuator
CLUTCH
clutch_sensor
CONTROLLER
motor_actuator
MOTOR
motor_sensor
door_actuator
door_sensor
DOOR

4. Proposing a Refinement Strategy

Refinement Strategy

- Initial model: Connecting the controller to the motor

- 1st refinement: Connecting the motor buttons to the controller

- 2nd refinement: Connecting the controller to the clutch

- 3rd refinement: Constraining the clutch and the motor

Refinement Strategy (cont’d)

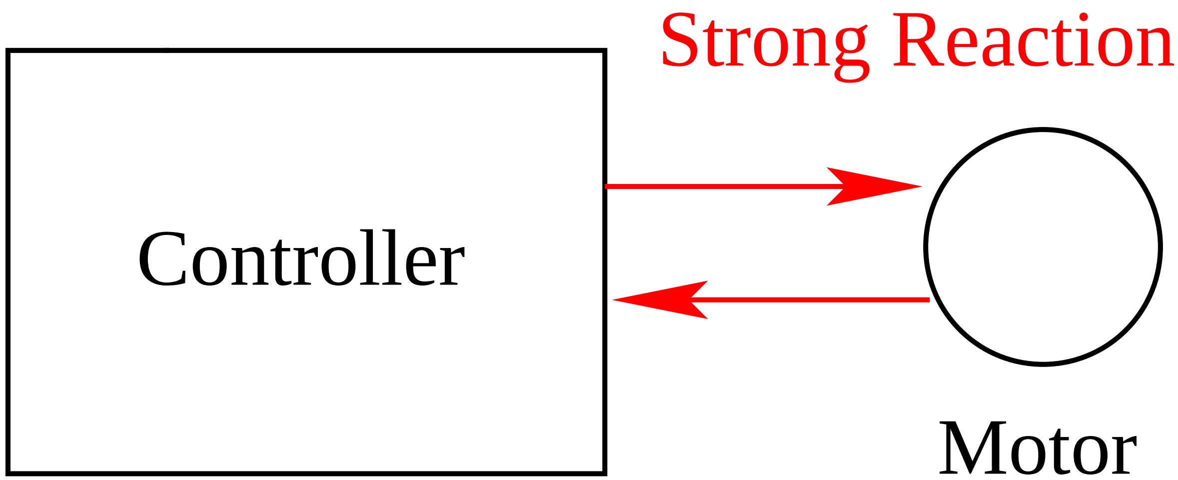
- 4th refinement: Connecting the controller to the door

- 5th refinement: Constraining the clutch and the door

- 6th refinement: More constraints between clutch and door

- 7th refinement: Connecting the clutch buttons to the controller

5. Development of the Model using Refinements and Design Patterns



Controller are Equipment are strongly synchronized	FUN_2

The counters have

been removed

a_on
when
$a = 0$
$r = 0$
then
$a := 1$
end

a_off
when
$a = 1$
$r = 1$
then
$a := 0$
end

init
$a := 0$
$r := 0$

r_on
when
$r = 0$
$a = 1$
then
$r := 1$
end

r_off
when
$r = 1$
$a = 0$
then
$r := 0$
end

Initial Model: the Context

set: $STATUS$

constants: $stopped$
$working$

axm0_1: $STATUS = \{stopped, working\}$

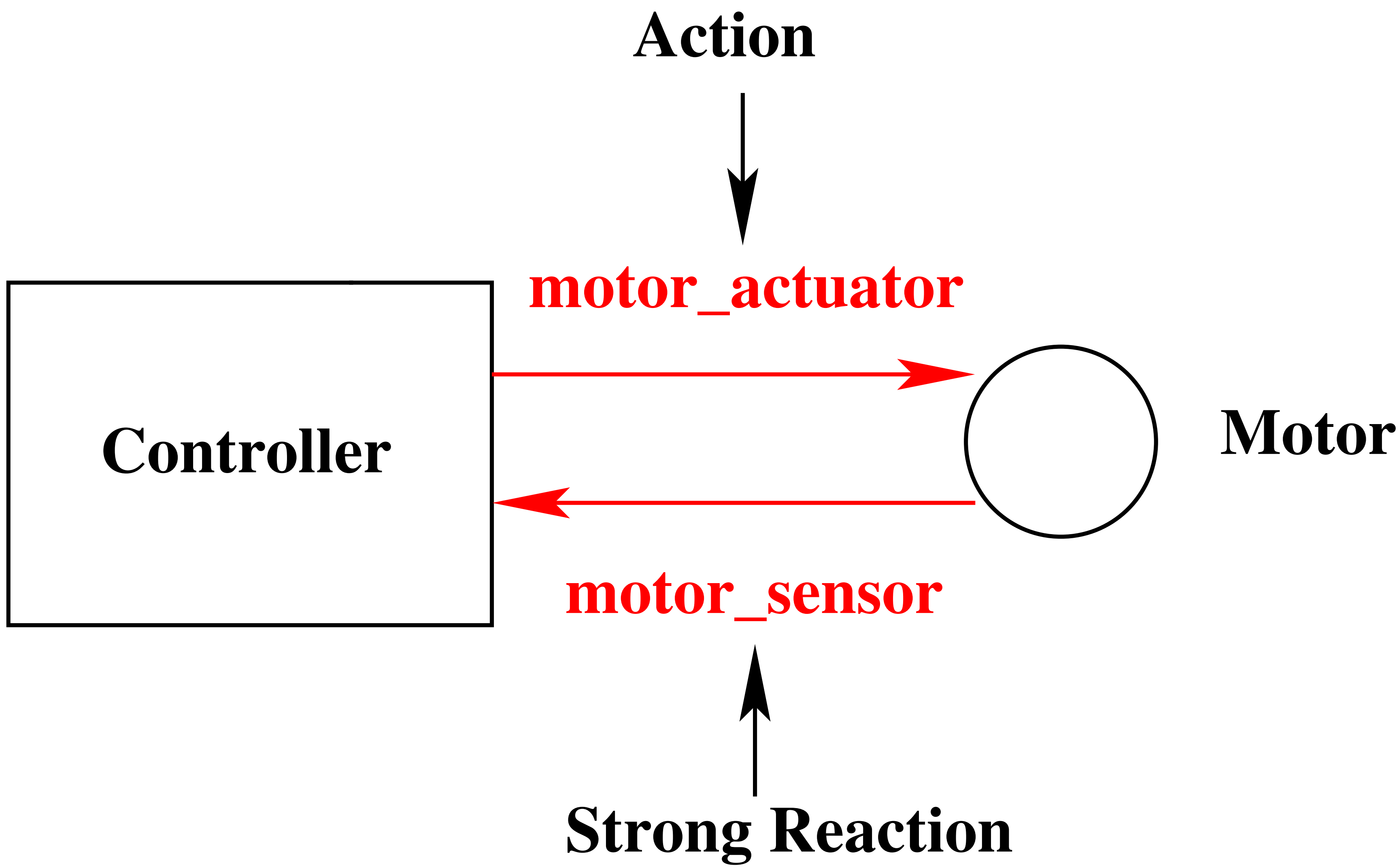
axm0_2: $stopped \neq working$

Initial Model: the State

variables: $motor\_actuator$
$motor\_sensor$

inv0_1: $motor\_sensor \in STATUS$

inv0_2: $motor\_actuator \in STATUS$


Action
motor_actuator
Controller
Motor
motor_sensor
Strong Reaction

Pattern Instantiation

- We instantiate the weak pattern as follows:

$$\begin{array}{lcl} a & \leadsto & motor\_actuator \\ r & \leadsto & motor\_sensor \\ 0 & \leadsto & stopped \\ 1 & \leadsto & working \end{array}$$

a_on	$\leadsto$	treat_start_motor
a_off	$\leadsto$	treat_stop_motor
r_on	$\leadsto$	Motor_start
r_off	$\leadsto$	Motor_stop

- Convention: Controller events start with "treat_"

Initial Model: Initialization

init
$a := 0$
$r := 0$

init
$motor\_actuator := stopped$
$motor\_sensor := stopped$

Initial Model: Controller Events (1)

a_on
when
$a = 0$
$r = 0$
then
$a := 1$
end

treat_start_motor
when
$motor\_actuator = stopped$
$motor\_sensor = stopped$
then
$motor\_actuator := working$
end

Initial Model: Environment Event (1)

r_on
when
$r = 0$
$a = 1$
then
$r := 1$
end

Motor_start
when
$motor\_sensor = stopped$
$motor\_actuator = working$
then
$motor\_sensor := working$
end

Initial Model: Controller Events (2)

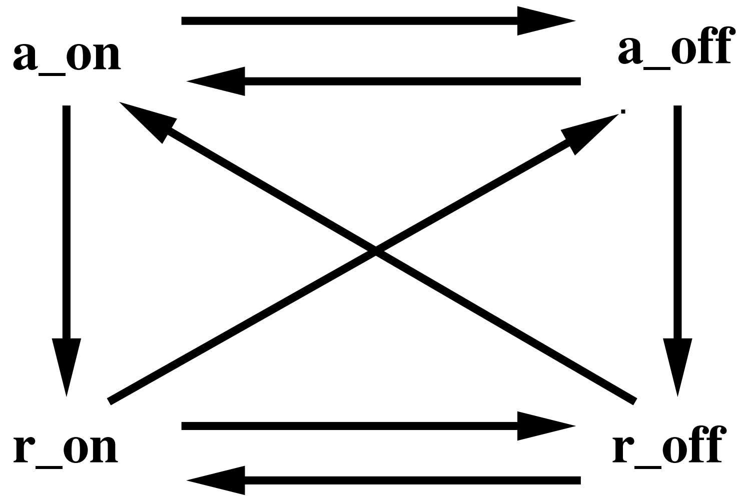
a_off
when
$a = 1$
$r = 1$
then
$a := 0$
end

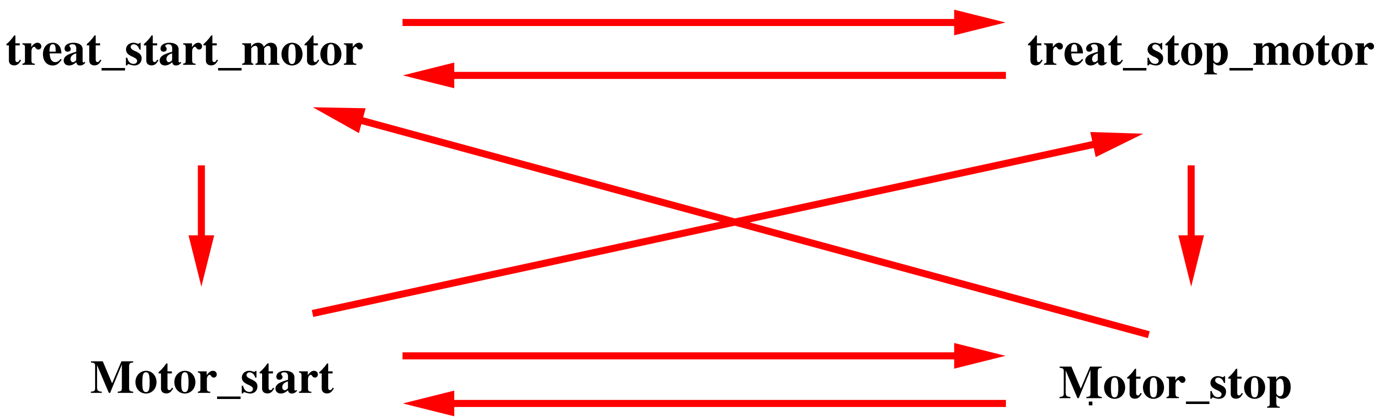
treat_stop_motor
when
$motor\_actuator = working$
$motor\_sensor = working$
then
$motor\_actuator := stopped$
end

Initial Model: Environment Event (2)

r_off
when
$r = 1$
$a = 0$
then
$r := 0$
end

Motor_stop
when
$motor\_sensor = working$
$motor\_actuator = stopped$
then
$motor\_sensor := stopped$
end


a_on
a_off
r_on
r_off

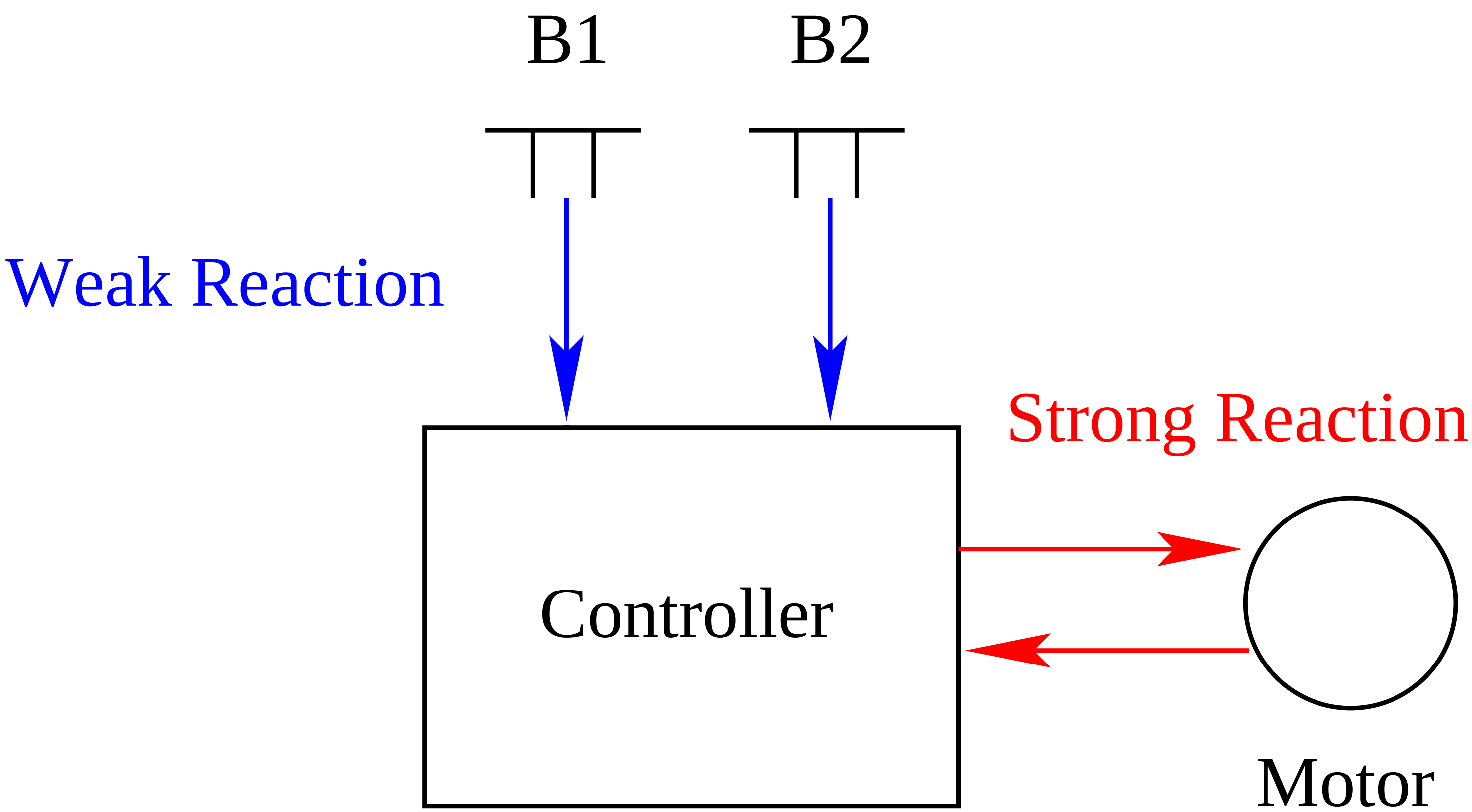

treat_start_motor
treat_stop_motor
Motor_start
Motor_stop

Initial Model: Summary of the Events

- Environment
  - motor_start
  - motor_stop

- Controller
  - treat_start_motor
  - treat_stop_motor

1st Reft.: Connecting the Motor Buttons to the Controller



Buttons and Controller are weakly synchronized	FUN_1

The counters have

been removed

init
$a := 0$
$r := 0$

a_on
when
$a = 0$
then
$a := 1$
end

a_off
when
$a = 1$
then
$a := 0$
end

r_on
when
$r = 0$
$a = 1$
then
$r := 1$
end

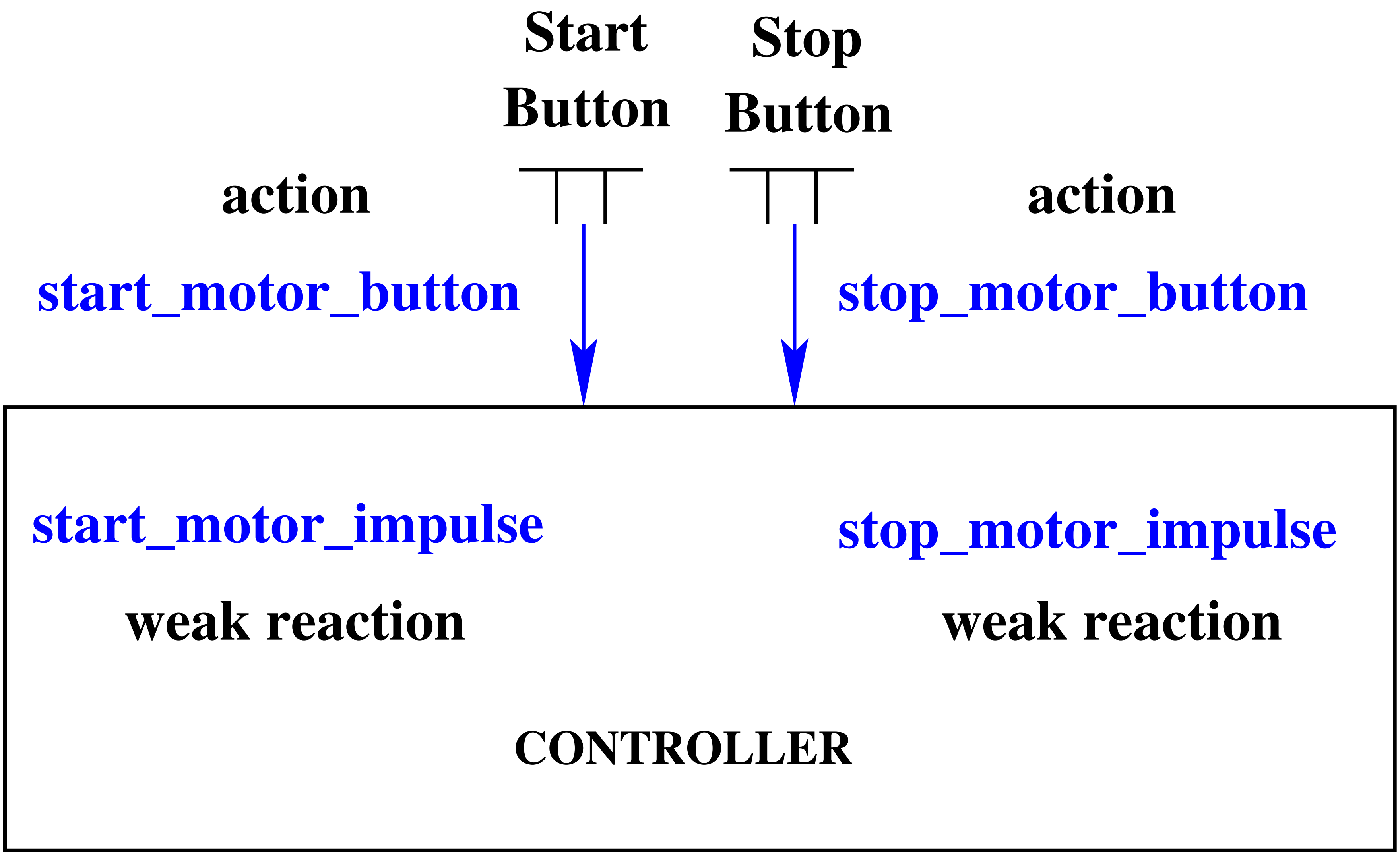
r_off
when
$r = 1$
$a = 0$
then
$r := 0$
end

First Refinement: the State

variables:

$\ldots$
$start\_motor\_button$
$stop\_motor\_button$
$start\_motor\_impulse$
$stop\_motor\_impulse$

inv1_1: $stop\_motor\_button \in \text{BOOL}$
inv1_2: $start\_motor\_button \in \text{BOOL}$
inv1_3: $stop\_motor\_impulse \in \text{BOOL}$
inv1_4: $start\_motor\_impulse \in \text{BOOL}$


Start
Button
Stop
Button
action
action
start_motor_button
stop_motor_button
start_motor_impulse
stop_motor_impulse
weak reaction
weak reaction
CONTROLLER

- We instantiate the pattern as follows:

$$\begin{array}{lcl} a & \leadsto & start\_motor\_button \\ r & \leadsto & start\_motor\_impulse \\ \mathbf{0} & \leadsto & \mathbf{FALSE} \\ \mathbf{1} & \leadsto & \mathbf{TRUE} \end{array}$$

a_on	$\leadsto$	push_start_motor_button
a_off	$\leadsto$	release_stop_motor_button
r_on	$\leadsto$	treat_push_start_motor_button
r_off	$\leadsto$	treat_release_start_motor_button

- We rename treat_start_motor as treat_push_start_motor_button init

$a := 0$

$r := 0$

init

$motor\_actuator := stopped$

$motor\_sensor := stopped$

$start\_motor\_button := \mathrm{FALSE}$

$start\_motor\_impulse := \mathrm{FALSE}$

First Refinement: New Environment Events (1)

a_on
when
$a = 0$
then
$a := 1$
end

push_start_motor_button
when
$start\_motor\_button = \mathrm{FALSE}$
then
$start\_motor\_button := \mathrm{TRUE}$
end

a_off
when
$a = 1$
then
$a := 0$
end

release_start_motor_button
when
$start\_motor\_button = \mathrm{TRUE}$
then
$start\_motor\_button := \mathrm{FALSE}$
end

First Refinement: Refining Controller Events (1)

r_on	treat_push_start_motor_button
	refines treat_start_motor
when $r = 0$, $a = 1$	**when** $start\_motor\_impulse = \mathrm{FALSE}$, $start\_motor\_button = \mathrm{TRUE}$, $motor\_actuator = stopped$, $motor\_sensor = stopped$
then $r := 1$	**then** $start\_motor\_impulse := \mathrm{TRUE}$, $motor\_actuator := working$
end	**end**

- This is the most important slide of the talk

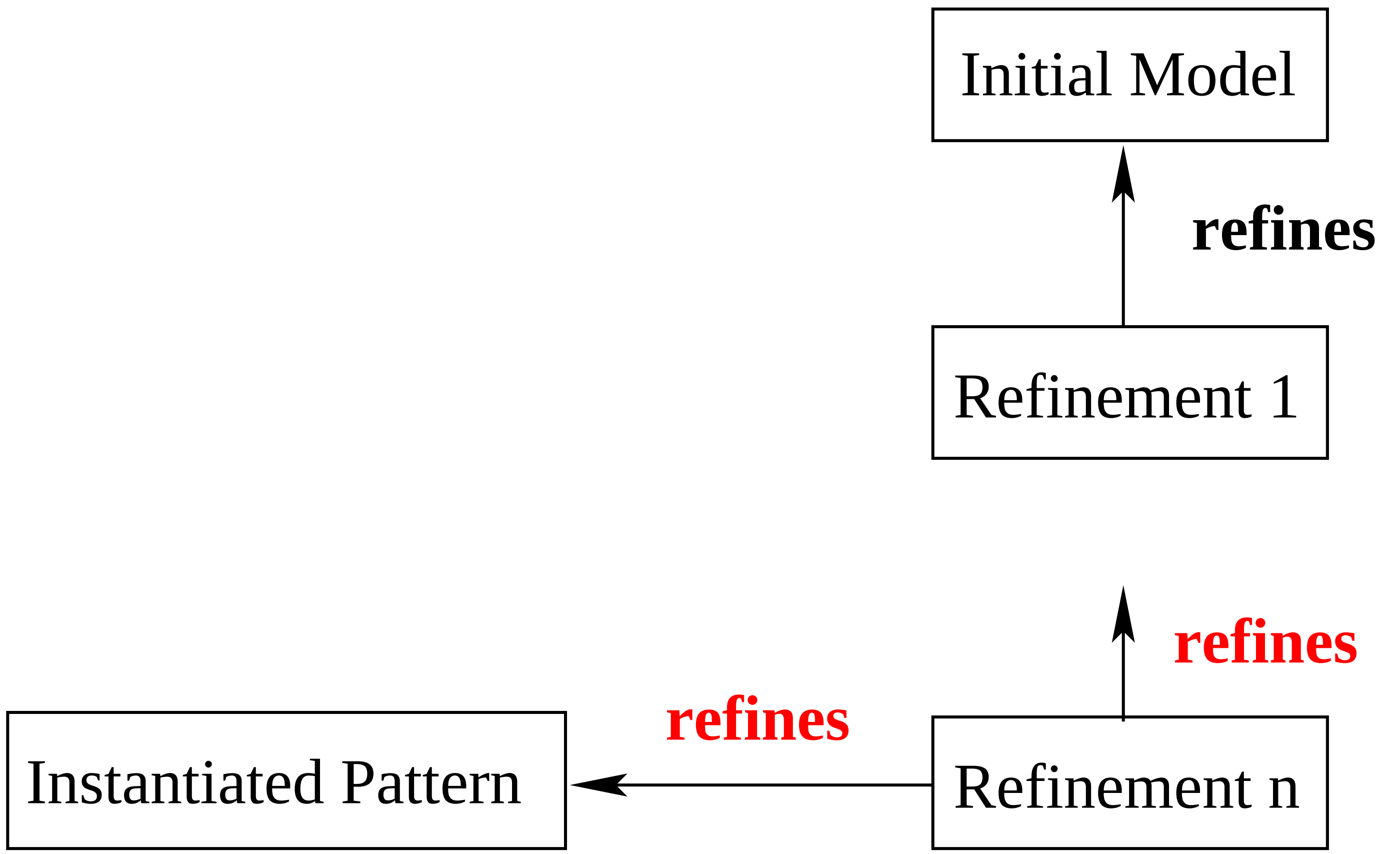
- We can see how patterns can be superposed

a_on
when
$a = 0$
$r = 0$
then
$a := 1$
end

treat_start_motor
when
$motor\_actuator = stopped$
$motor\_sensor = stopped$
then
$motor\_actuator := working$
end

r_on
when
r = 0
a = 1
then
r := 1
end

treat_push_start_motor_button
when
$start\_motor\_impulse = \mathrm{FALSE}$
$start\_motor\_button = \mathrm{TRUE}$
$motor\_actuator = stopped$
$motor\_sensor = stopped$
then
$start\_motor\_impulse := \mathrm{TRUE}$
$motor\_actuator := working$
end


Initial Model
refines
Refinement 1
refines
refines
Instantiated Pattern
Refinement n

First Refinement: New Controller Events (1)

r_off
when
$r = 1$
$a = 0$
then
$r := 0$
end

treat_release_start_motor_button
when
$start\_motor\_impulse = \mathrm{TRUE}$
$start\_motor\_button = \mathrm{FALSE}$
then
$start\_motor\_impulse := \mathrm{FALSE}$
end

- We instantiate the pattern as follows:

$$\begin{array}{lcl} a & \rightsquigarrow & stop\_motor\_button \\ r & \rightsquigarrow & stop\_motor\_impulse \\ 0 & \rightsquigarrow & \mathbf{FALSE} \\ 1 & \rightsquigarrow & \mathbf{TRUE} \end{array}$$

a_on	$\rightsquigarrow$	push_stop_motor_button
a_off	$\rightsquigarrow$	release_stop_motor_button
r_on	$\rightsquigarrow$	treat_push_stop_motor_button
r_off	$\rightsquigarrow$	treat_release_stop_motor_button

1st Refinement: More Refinement of Initialization

init

$$a := 0$$
$$r := 0$$

init

$$motor\_actuator := stopped$$
$$motor\_sensor := stopped$$
$$start\_motor\_button := \mathbf{FALSE}$$
$$start\_motor\_impulse := \mathbf{FALSE}$$
$$stop\_motor\_button := \mathbf{FALSE}$$
$$stop\_motor\_impulse := \mathbf{FALSE}$$

First Refinement: New Environment Events

a_on
when
$a = 0$
then
$a := 1$
end

push_stop_motor_button
when
$stop\_motor\_button = \mathrm{FALSE}$
then
$stop\_motor\_button := \mathrm{TRUE}$
end

a_off
when
$a = 1$
then
$a := 0$
end

release_stop_motor_button
when
$stop\_motor\_button = \mathrm{TRUE}$
then
$stop\_motor\_button := \mathrm{FALSE}$
end

r_on

when

$r = 0$

$a = 1$

then

$r := 1$

end

treat_push_stop_motor_button

refines

treat_stop_motor

when

$stop\_motor\_impulse = \mathrm{FALSE}$

$stop\_motor\_button = \mathrm{TRUE}$

$motor\_sensor = working$

$motor\_actuator = working$

then

$stop\_motor\_impulse := \mathrm{TRUE}$

$motor\_actuator := stopped$

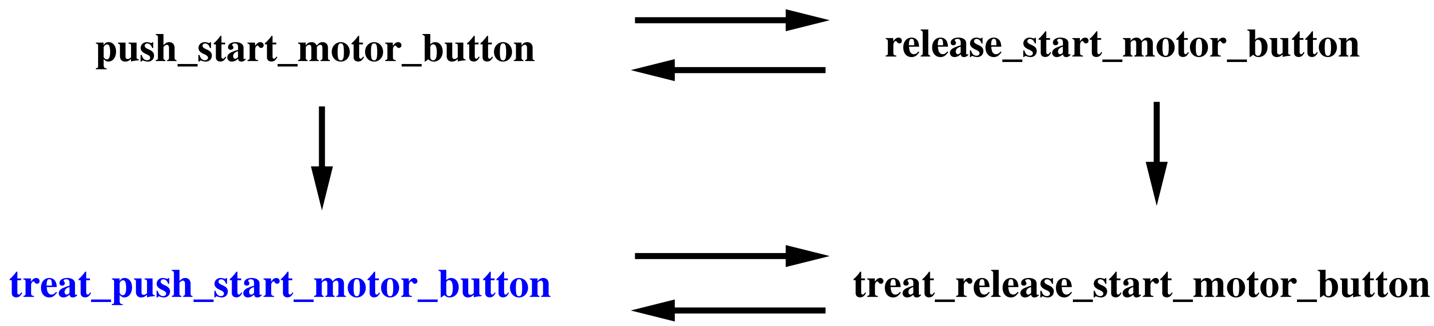
end

First Refinement: New Controller Events (2)

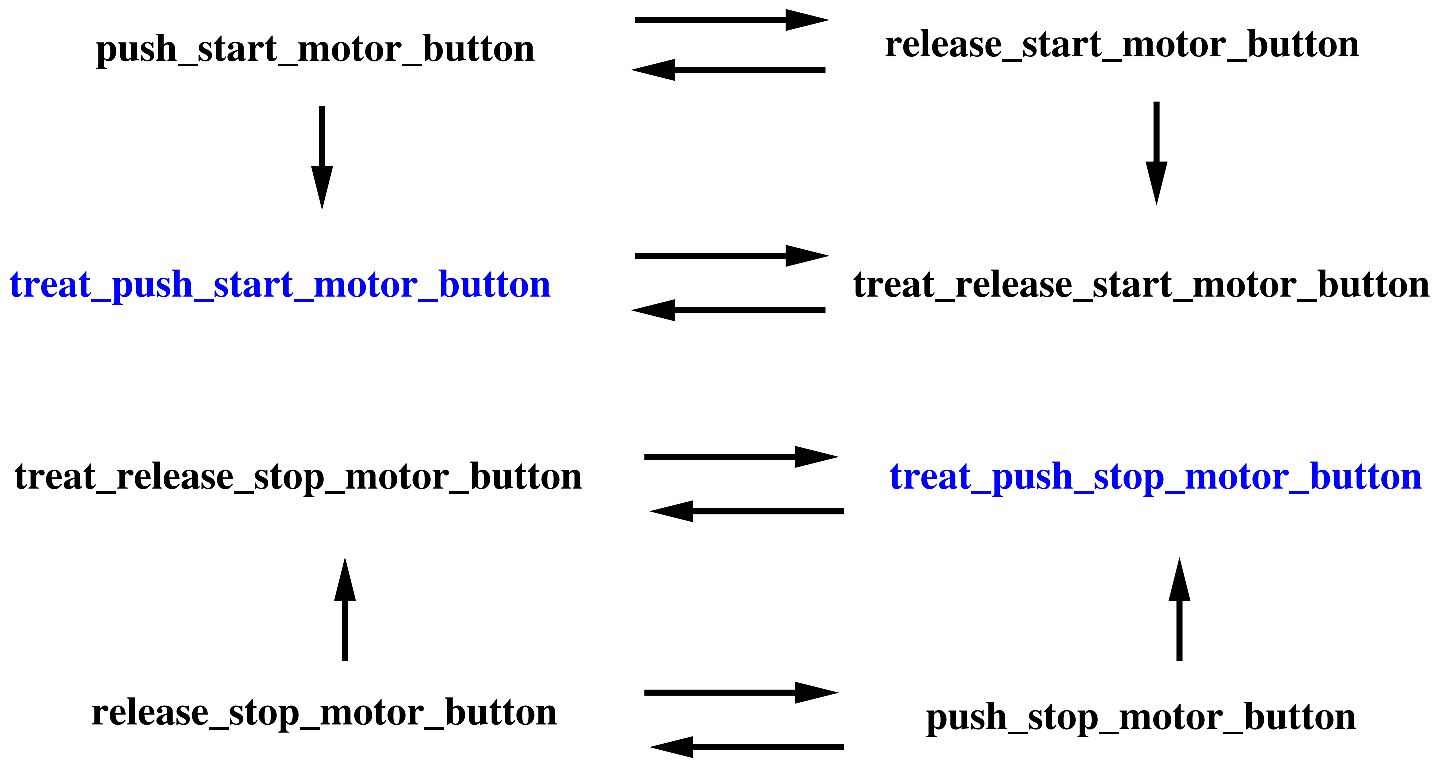
r_off
  when
    $r = 1$
    $a = 0$
  then
    $r := 0$
  end

treat_release_stop_motor_button
  when
    $stop\_motor\_impulse = \mathrm{TRUE}$
    $stop\_motor\_button = \mathrm{FALSE}$
  then
    $stop\_motor\_impulse := \mathrm{FALSE}$
  end

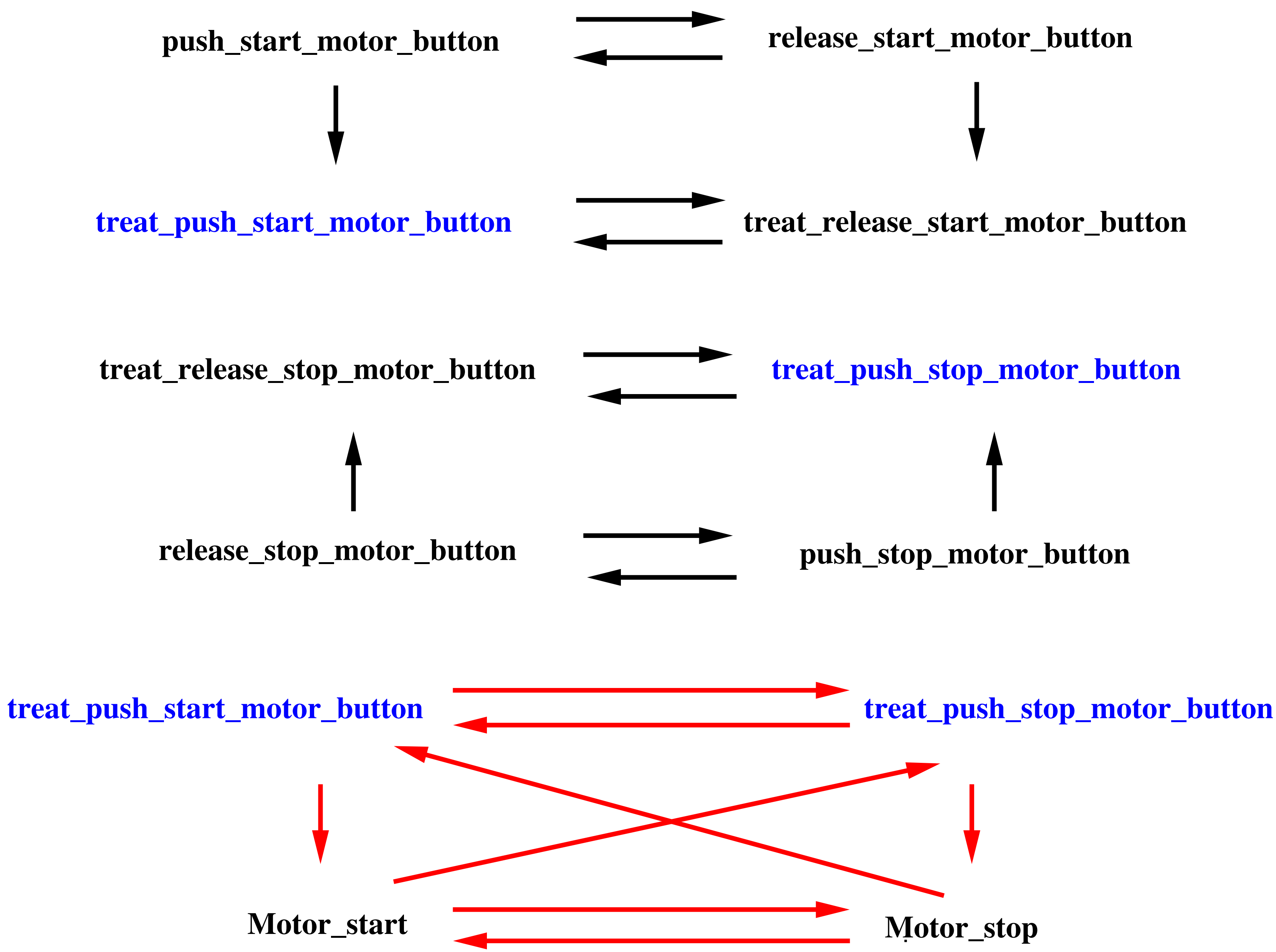
Independent Synchronizations



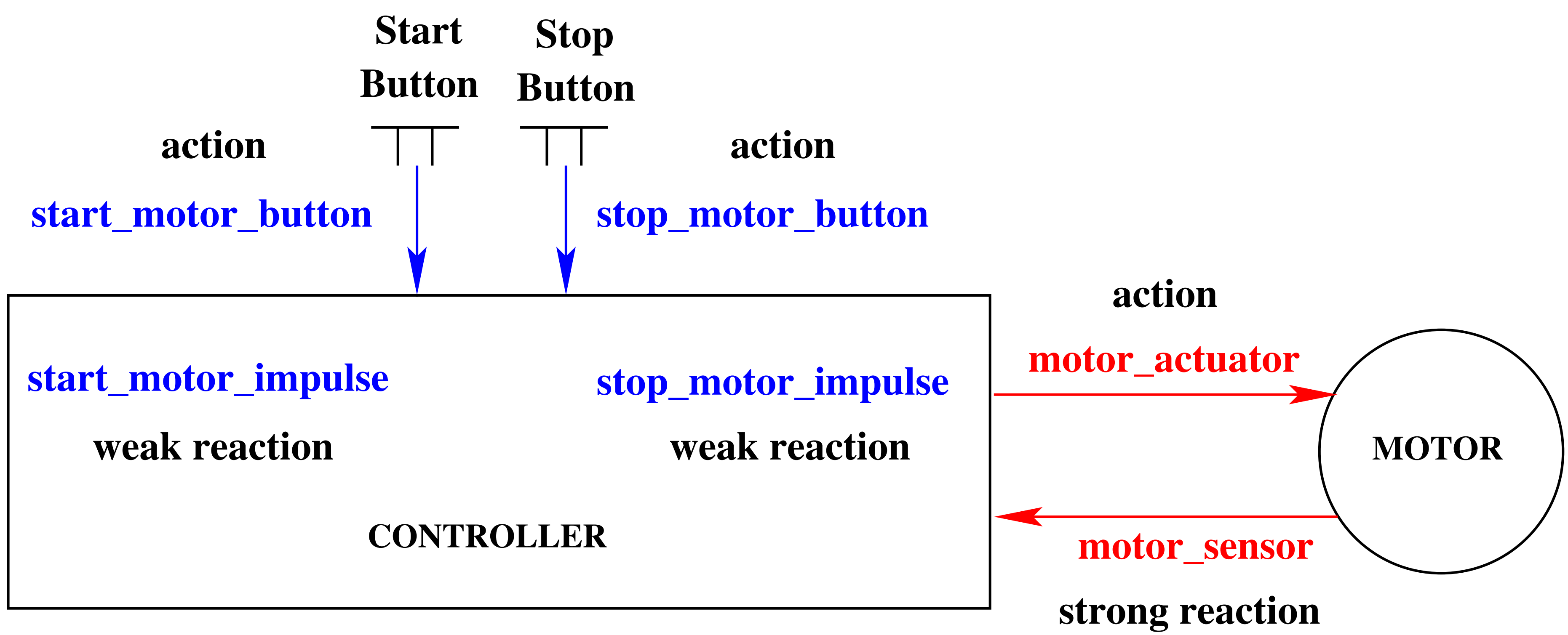
Independent Synchronizations



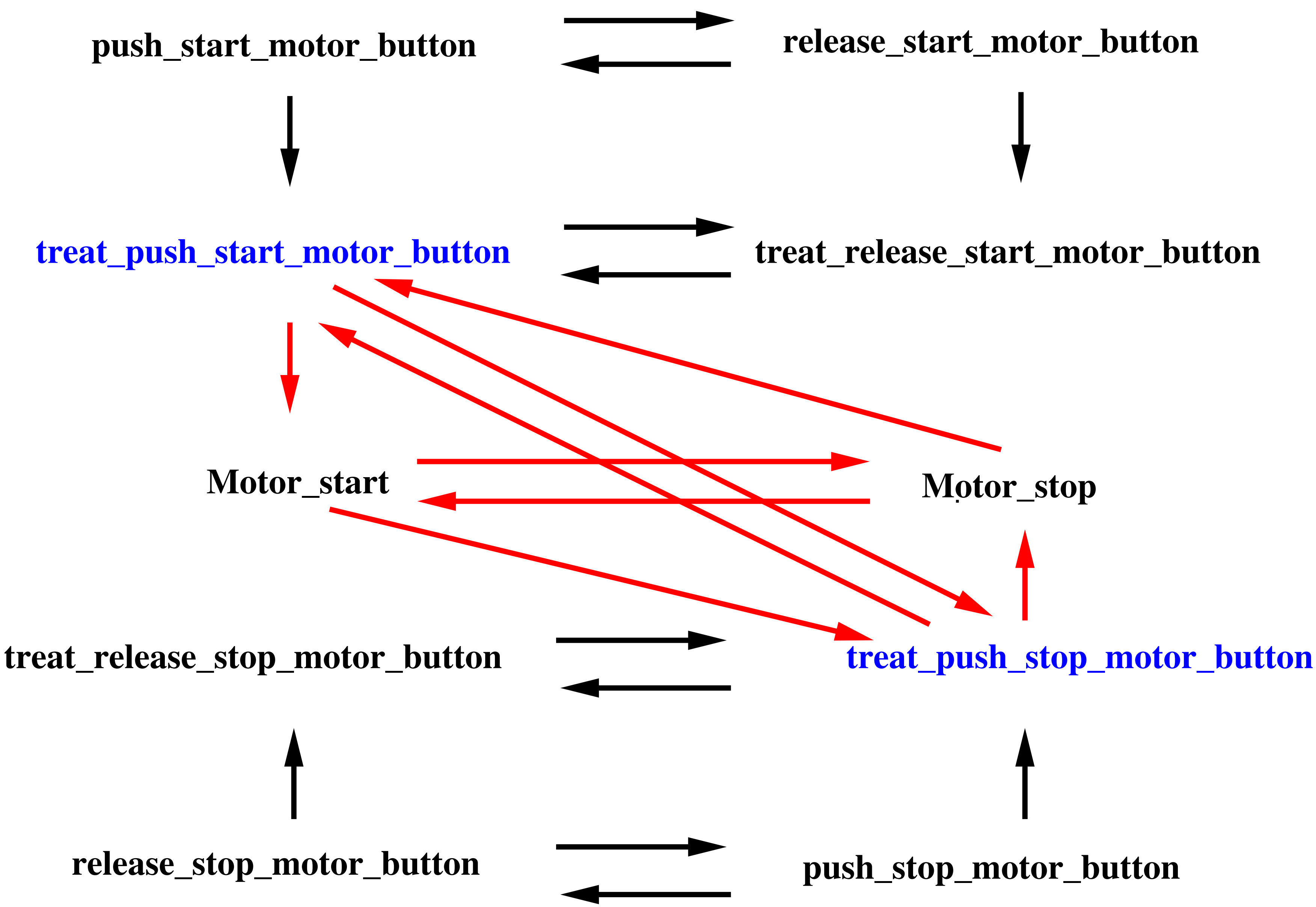
Independent Synchronizations



Weak and Strong Reactions Together



Combined Synchronizations



treat_push_start_motor_button
refines
treat_start_motor
when
$start\_motor\_impulse = \mathrm{FALSE}$
$start\_motor\_button = \mathrm{TRUE}$
$motor\_actuator = stopped$
$motor\_sensor = stopped$
then
$start\_motor\_impulse := \mathrm{TRUE}$
$motor\_actuator := working$
end

- What happens when the following hold

$$\neg\,(motor\_actuator = stopped \;\;\wedge\;\; motor\_sensor = stopped)$$

- We need another event

Problems with treat_push_start_motor_button

treat_push_start_motor_button
refines
treat_start_motor
when
$start\_motor\_impulse = \text{FALSE}$
$start\_motor\_button = \text{TRUE}$
$motor\_actuator = stopped$
$motor\_sensor = stopped$
then
$start\_motor\_impulse := \text{TRUE}$
$motor\_actuator := working$
end

treat_push_start_motor_button_false
when
$start\_motor\_impulse = \text{FALSE}$
$start\_motor\_button = \text{TRUE}$
$\neg\,(motor\_actuator = stopped \;\wedge$
$motor\_sensor = stopped)$
then
$start\_motor\_impulse := \text{TRUE}$
end

- In the second case, the button has been pushed but the internal conditions are not met

- However, we need to record that the button has been pushed:

$$start\_motor\_impulse := \text{TRUE}$$

Problems with treat_push_stop_motor_button

treat_push_stop_motor_button	treat_push_stop_motor_button_false
refines treat_stop_motor	
when	**when**
$stop\_motor\_impulse = \mathrm{FALSE}$	$stop\_motor\_impulse = \mathrm{FALSE}$
$stop\_motor\_button = \mathrm{TRUE}$	$stop\_motor\_button = \mathrm{TRUE}$
$motor\_sensor = working$	$\neg\,(motor\_sensor = working\ \wedge$
$motor\_actuator = working$	$motor\_actuator = working)$
then	**then**
$stop\_motor\_impulse := \mathrm{TRUE}$	$stop\_motor\_impulse := \mathrm{TRUE}$
$motor\_actuator := stopped$	
end	**end**

- In the second case, the button has been pushed but the internal conditions are not met

- However, we need to record that the button has been pushed:

$$stop\_motor\_impulse := \mathrm{TRUE}$$

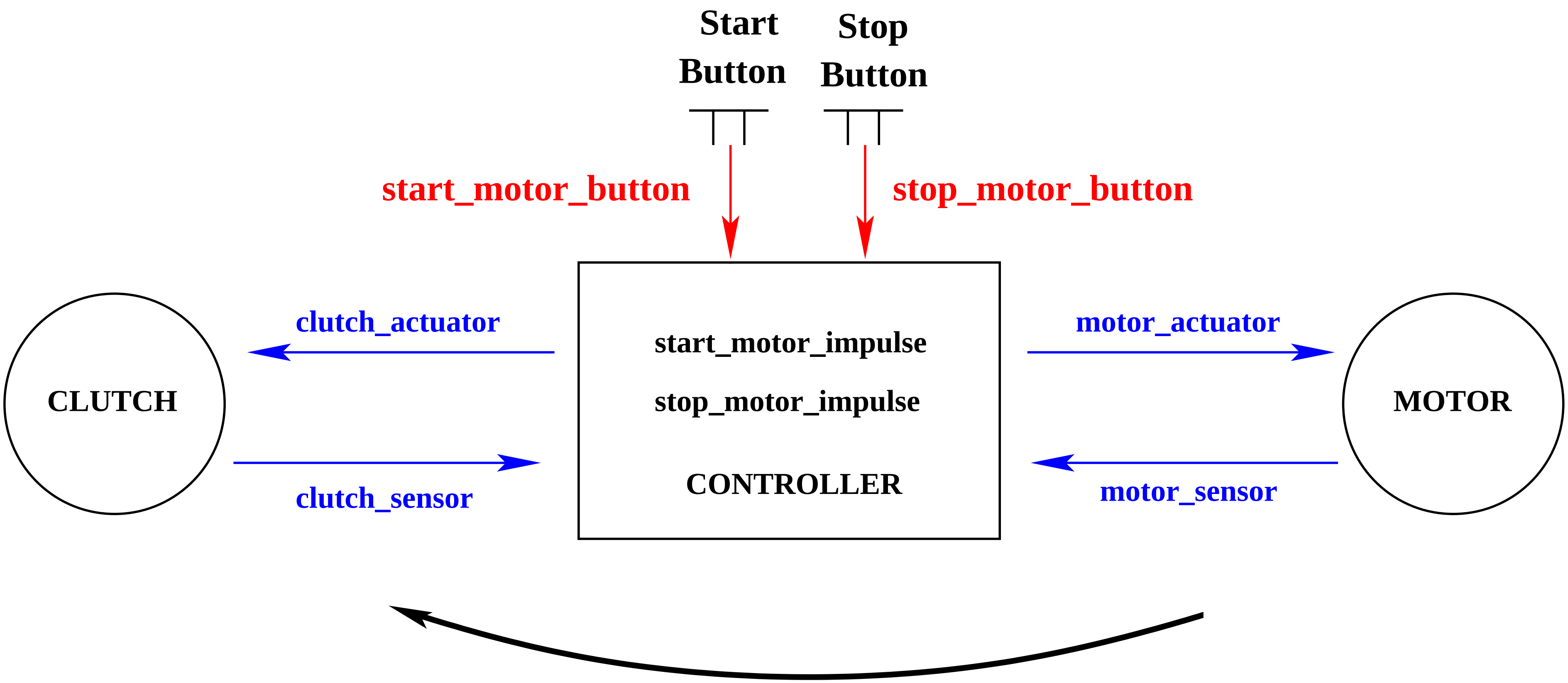
First Refinement: Summary of the Events (1)

- Environment
  - motor_start
  - motor_stop
  - push_start_motor_button
  - release_start_motor_button
  - push_stop_motor_button
  - release_stop_motor_button

First Refinement: Summary of the Events (2)

- Controller
    - treat_push_start_motor_button
    - treat_push_start_motor_button_false
    - treat_push_stop_motor_button
    - treat_push_stop_motor_button_false
    - treat_release_start_motor_button
    - treat_release_stop_motor_button

2nd Refinement: Connecting the Controller to the Clutch



- We introduce the set in a new context:

$$CLUTCH = \{engaged, disengaged\}$$

- We copy the initial model where we instantiate:

$$\begin{array}{lcl} motor & \leadsto & clutch \\ STATUS & \leadsto & CLUTCH \\ working & \leadsto & engaged \\ stopped & \leadsto & disengaged \end{array}$$

Second Refinement: Summary of the Events (1)

- Environment
  - motor_start
  - motor_stop
  - clutch_start
  - clutch_stop
  - push_start_motor_button
  - release_start_motor_button
  - push_stop_motor_button
  - release_stop_motor_button

Second Refinement: Summary of the Events (2)

- Controller
    - treat_push_start_motor_button
    - treat_push_start_motor_button_false
    - treat_push_stop_motor_button
    - treat_push_stop_motor_button_false
    - treat_release_start_motor_button
    - treat_release_stop_motor_button
    - treat_start_clutch
    - treat_stop_clutch

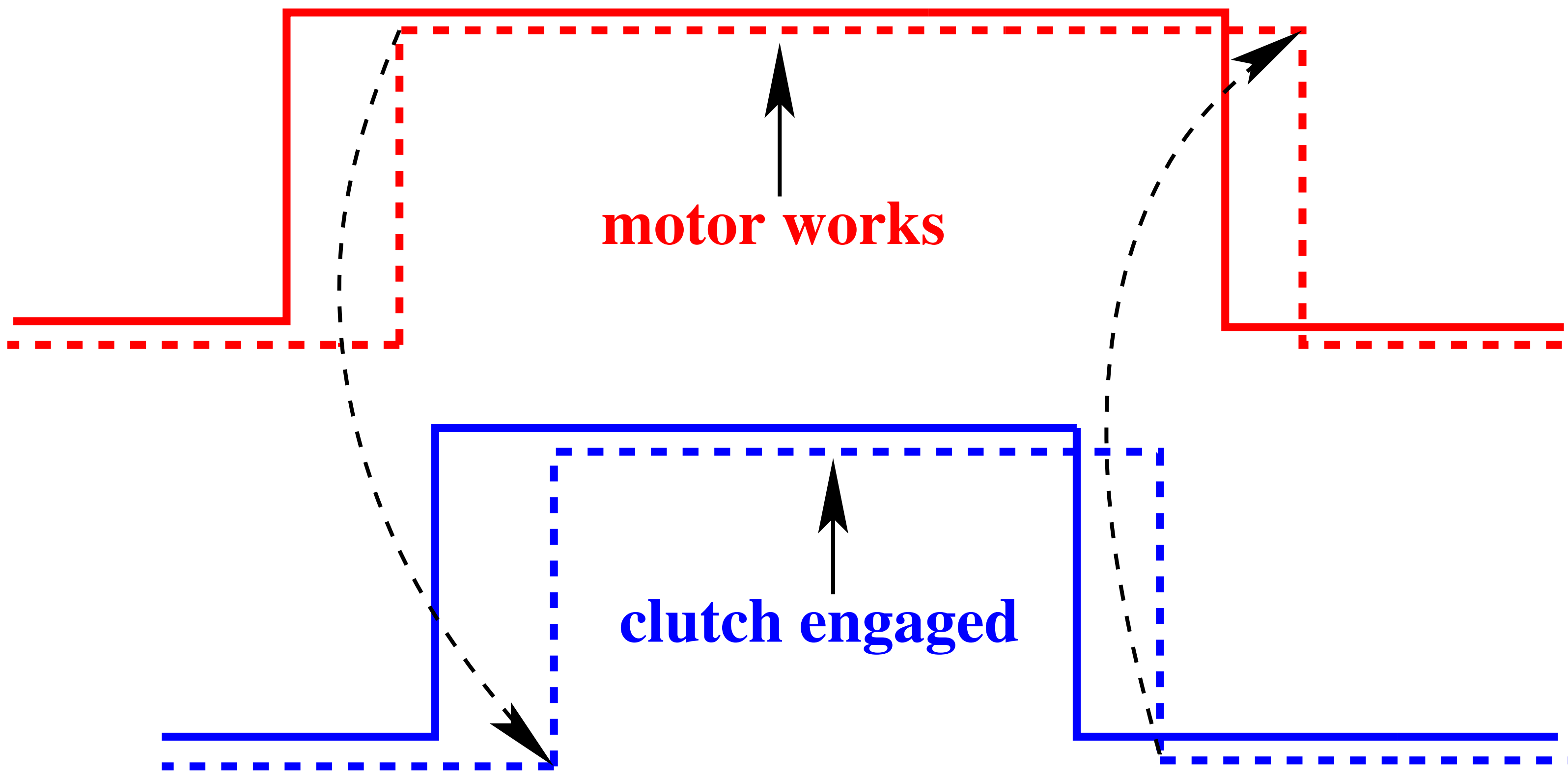
- An additional safety constraint

When the clutch is engaged, the motor must work	SAF_1

- For this we develop ANOTHER DESIGN PATTERN

- It is called: Weak synchronization of two Strong Reactions

Pattern: Weak Synchronization of Strong Reactions

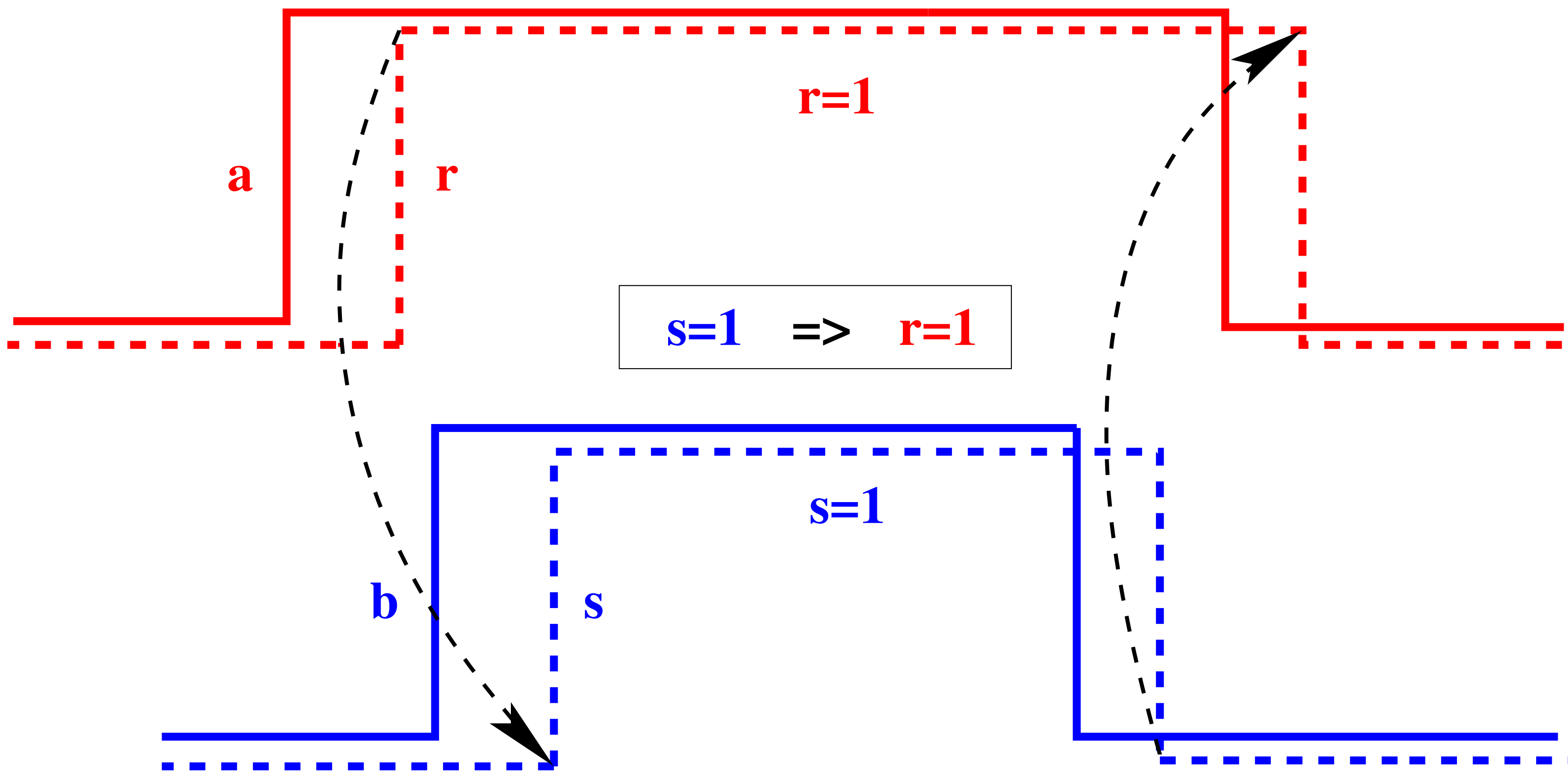


When the clutch is engaged

then

the motor must work

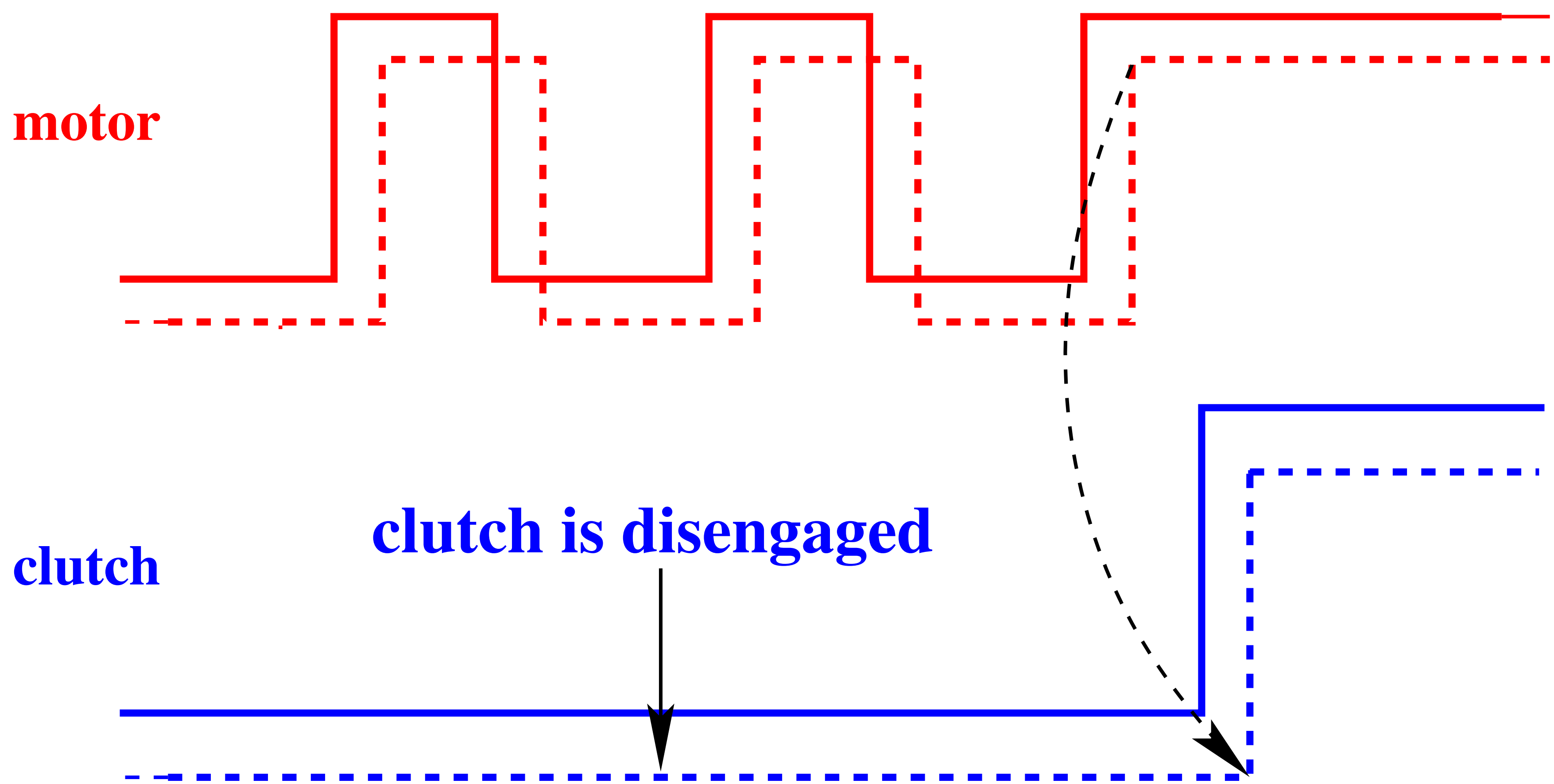
Pattern: Weak Synchronization of Strong Reactions



When the clutch is engaged

then

the motor must work

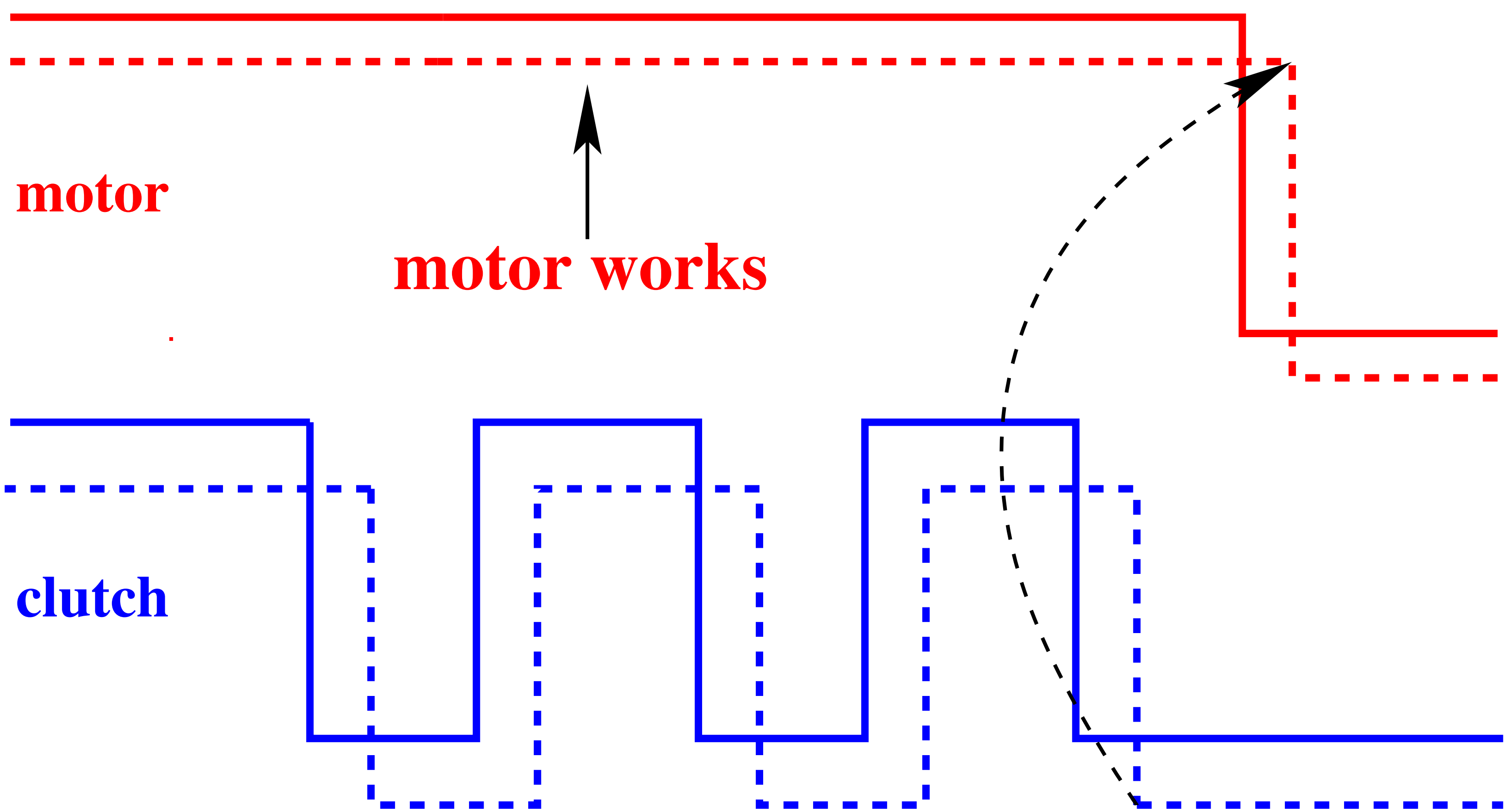


When the clutch is disengaged,

then

the motor can be started and stopped several times

The Synchronization is Weak (2)

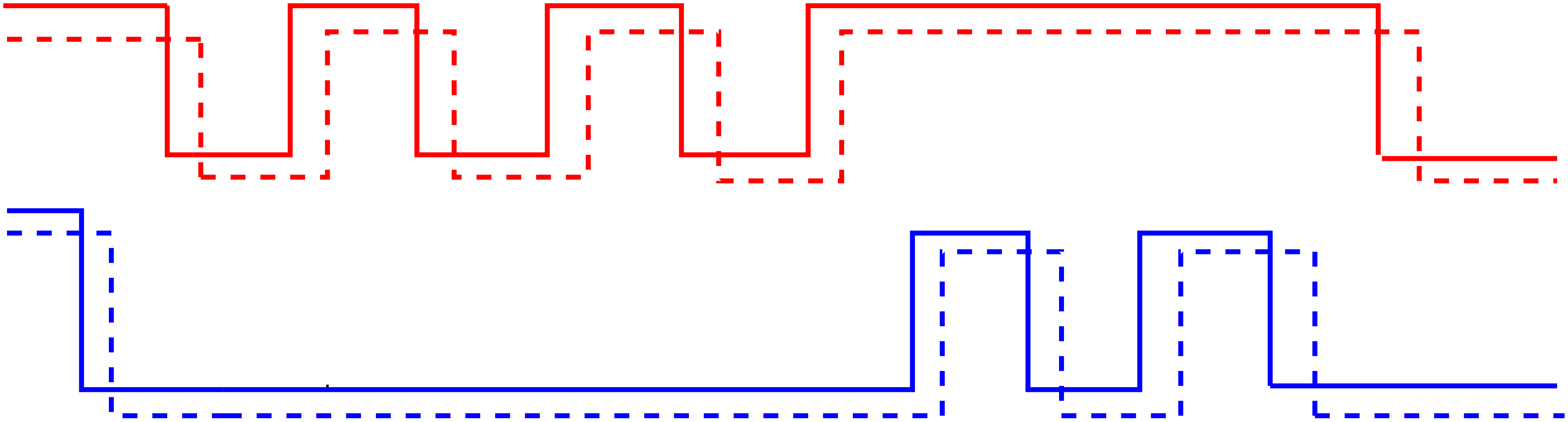


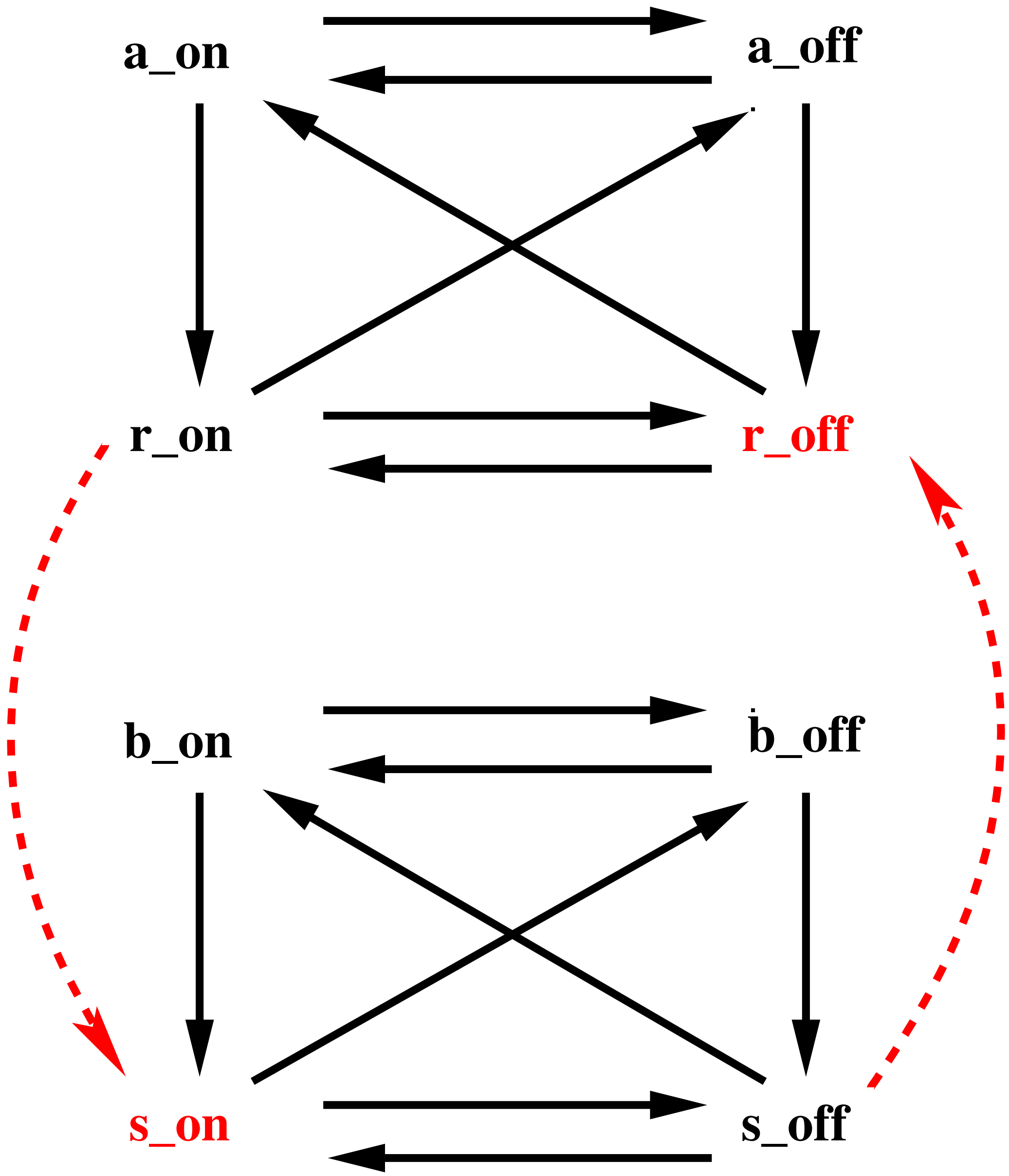
When the motor works,

then

the clutch can be engaged and disengaged several times

Putting the Two Together




a_on
a_off
r_on
r_off
b_on
b_off
s_on
s_off

The Initial State Situation

dbl0_1: $a \in \{0,1\}$
dbl0_2: $r \in \{0,1\}$
dbl0_3: $ca \in \mathbb{N}$
dbl0_4: $cr \in \mathbb{N}$
dbl0_5: $a = 1 \;\wedge\; r = 0 \;\Rightarrow\; ca = cr + 1$
dbl0_6: $a = 0 \;\vee\; r = 1 \;\Rightarrow\; ca = cr$

dbl0_7: $b \in \{0,1\}$
dbl0_8: $s \in \{0,1\}$
dbl0_9: $cb \in \mathbb{N}$
dbl0_10: $cs \in \mathbb{N}$
dbl0_11: $b = 1 \;\wedge\; s = 0 \;\Rightarrow\; cb = cs + 1$
dbl0_12: $b = 0 \;\vee\; s = 1 \;\Rightarrow\; cb = cs$

The Initial Event Situation (1)

a_on
when
$a = 0$
$r = 0$
then
$a := 1$
$ca := ca + 1$
end

r_on
when
$r = 0$
$a = 1$
then
$r := 1$
$cr := cr + 1$
end

a_off
when
$a = 1$
$r = 1$
then
$a := 0$
end

r_off
when
$r = 1$
$a = 0$
then
$r := 0$
end

The Initial Event Situation (2)

b_on
when
$b = 0$
$s = 0$
then
$b := 1$
$cb := cb + 1$
end

s_on
when
$s = 0$
$b = 1$
then
$s := 1$
$cs := cs + 1$
end

b_off
when
$b = 1$
$s = 1$
then
$b := 0$
end

s_off
when
$s = 1$
$b = 0$
then
$s := 0$
end

dbl1_1: $s = 1 \;\Rightarrow\; r = 1$

- It seems sufficient to add the following guards

```
s_on
  when
    s = 0
    b = 1
    r = 1
  then
    s := 1
    cs := cs + 1
  end
```

```
r_off
  when
    r = 1
    a = 0
    s = 0
  then
    r := 0
  end
```

- But we do not want to touch these events

s_on
when
$s = 0$
$b = 1$
$r = 1$
then
$s := 1$
$cs := cs + 1$
end

r_off
when
$r = 1$
$a = 0$
$s = 0$
then
$r := 0$
end

- We introduce the following additional invariants

dbl1_2: $b = 1 \Rightarrow r = 1$

dbl1_3: $a = 0 \Rightarrow s = 0$

Maintaining Invariant dbl1_2 (1)

dbl1_2: $b = 1 \Rightarrow r = 1$

In order to maintain this invariant, we have to refine b_on

```
b_on
when
   b = 0
   s = 0
then
   b := 1
   cb := cb + 1
end
```

$\rightsquigarrow$

```
b_on
when
   b = 0
   s = 0
   r = 1
then
   b := 1
   cb := cb + 1
end
```

Maintaining (Contraposition of) Invariant dbl1_2 (2)

dbl1_2: $b = 1 \;\Rightarrow\; r = 1 \qquad (r = 0 \;\Rightarrow\; b = 0)$

In order to maintain this invariant, we have to refine r_off

r_off **when** $r = 1$ $a = 0$ **then** $r := 0$ **end**	$\rightsquigarrow$	r_off **when** $r = 1$ $a = 0$ $b = 0$ **then** $r := 0$ **end**

- But, again, we do not want to touch this event

r_off
  when
    $r = 1$
    $a = 0$
    $b = 0$
  then
    $r := 0$
  end

- We introduce the following invariant

dbl1_4: $a = 0 \;\Rightarrow\; b = 0$

Maintaining Invariant dbl1_3 (1)

dbl1_3: $a = 0 \quad \Rightarrow \quad s = 0$

In order to maintain this invariant, we have to refine a_off

a_off **when** $a = 1$ $r = 1$ **then** $a := 0$ **end**	$\leadsto$	a_off **when** $a = 1$ $r = 1$ $s = 0$ **then** $a := 0$ **end**

Maintaining (Contraposition of) Invariant dbl1_3 (2)

dbl1_3: $a = 0 \Rightarrow s = 0 \qquad (s = 1 \Rightarrow a = 1)$

In order to maintain this invariant, we have to refine s_on

s_on **when** $s = 0$ $b = 1$ **then** $s := 1$ $cs := cs + 1$ **end**	$\leadsto$	s_on **when** $s = 0$ $b = 1$ $a = 1$ **then** $s := 1$ $cs := cs + 1$ **end**

- But, again, we do not want to touch this event

```
s_on
  when
    s = 0
    b = 1
    a = 1
  then
    s := 1
    cs := cs + 1
  end
```

- We have to introduce the following invariant

$$b = 1 \;\Rightarrow\; a = 1$$

- Fortunately, this is **dbl1_4** ($a = 0 \;\Rightarrow\; b = 0$) contraposed

Maintaining Invariant dbl1_4 (1)

dbl1_4: $a = 0 \;\Rightarrow\; b = 0$

In order to maintain this invariant, we have to refine a_off again

```
a_off                      a_off
when                       when
   a = 1                      a = 1
   r = 1                      r = 1
   s = 0           ~>         s = 0
then                          b = 0
   a := 0                  then
end                           a := 0
                           end
```

dbl1_4: $a = 0 \Rightarrow b = 0 \qquad (b = 1 \Rightarrow a = 1)$

In order to maintain this invariant, we have to refine b_on again

```
b_on
when
   b = 0
   s = 0
   r = 1
then
   b, cb := 1, cb + 1
end
```

$\rightsquigarrow$

```
b_on
when
   b = 0
   s = 0
   r = 1
   a = 1
then
   b, cb := 1, cb + 1
end
```

dbl1_1:	$s = 1$	$\Rightarrow$	$r = 1$
dbl1_2:	$b = 1$	$\Rightarrow$	$r = 1$
dbl1_3:	$a = 0$	$\Rightarrow$	$s = 0$
dbl1_4:	$a = 0$	$\Rightarrow$	$b = 0$

b_on
when
  $b = 0$
  $s = 0$
  $r = 1$
  $a = 1$
then
  $b, cb := 1, cb + 1$
end

a_off
when
  $a = 1$
  $r = 1$
  $s = 0$
  $b = 0$
then
  $a := 0$
end

dbl1_1: $s = 1 \Rightarrow r = 1$
dbl1_2: $b = 1 \Rightarrow r = 1$
dbl1_3: $a = 0 \Rightarrow s = 0$ $\quad (s = 1 \Rightarrow a = 1)$
dbl1_4: $a = 0 \Rightarrow b = 0$ $\quad (b = 1 \Rightarrow a = 1)$

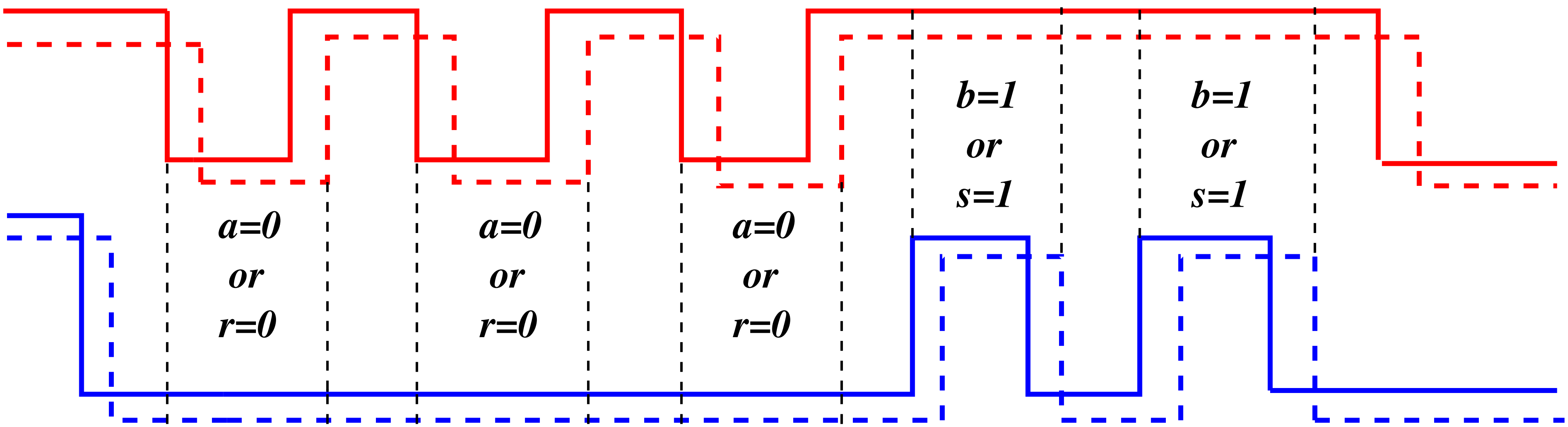
This can be put into a single invariant

dbl1_5: $b = 1 \lor s = 1 \Rightarrow a = 1 \land r = 1$

with the following contraposed form

dbl1_6: $a = 0 \lor r = 0 \Rightarrow b = 0 \land s = 0$

Intuition about the Invariants

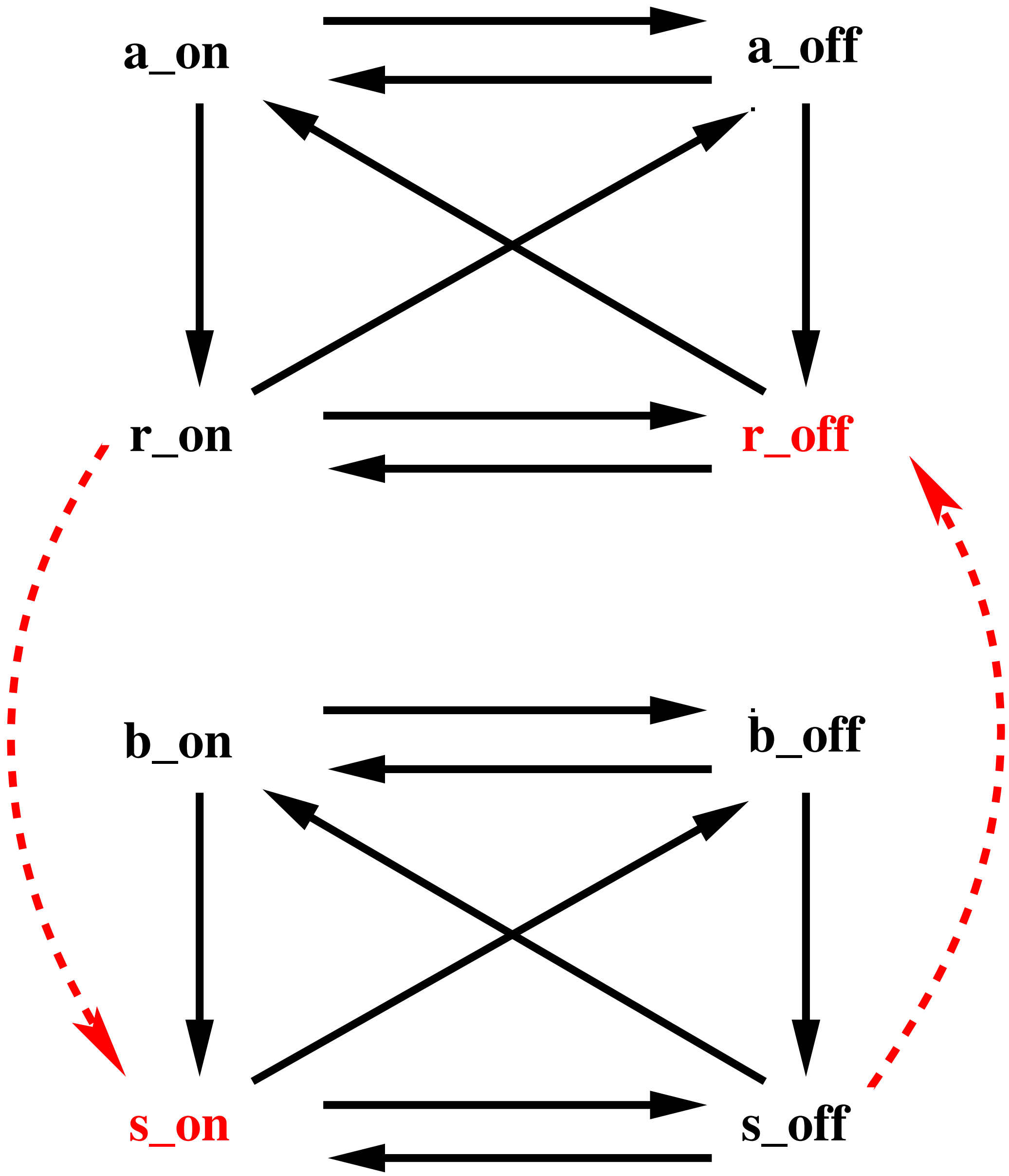


Reminder: - - - is the motor and - - - is the clutch

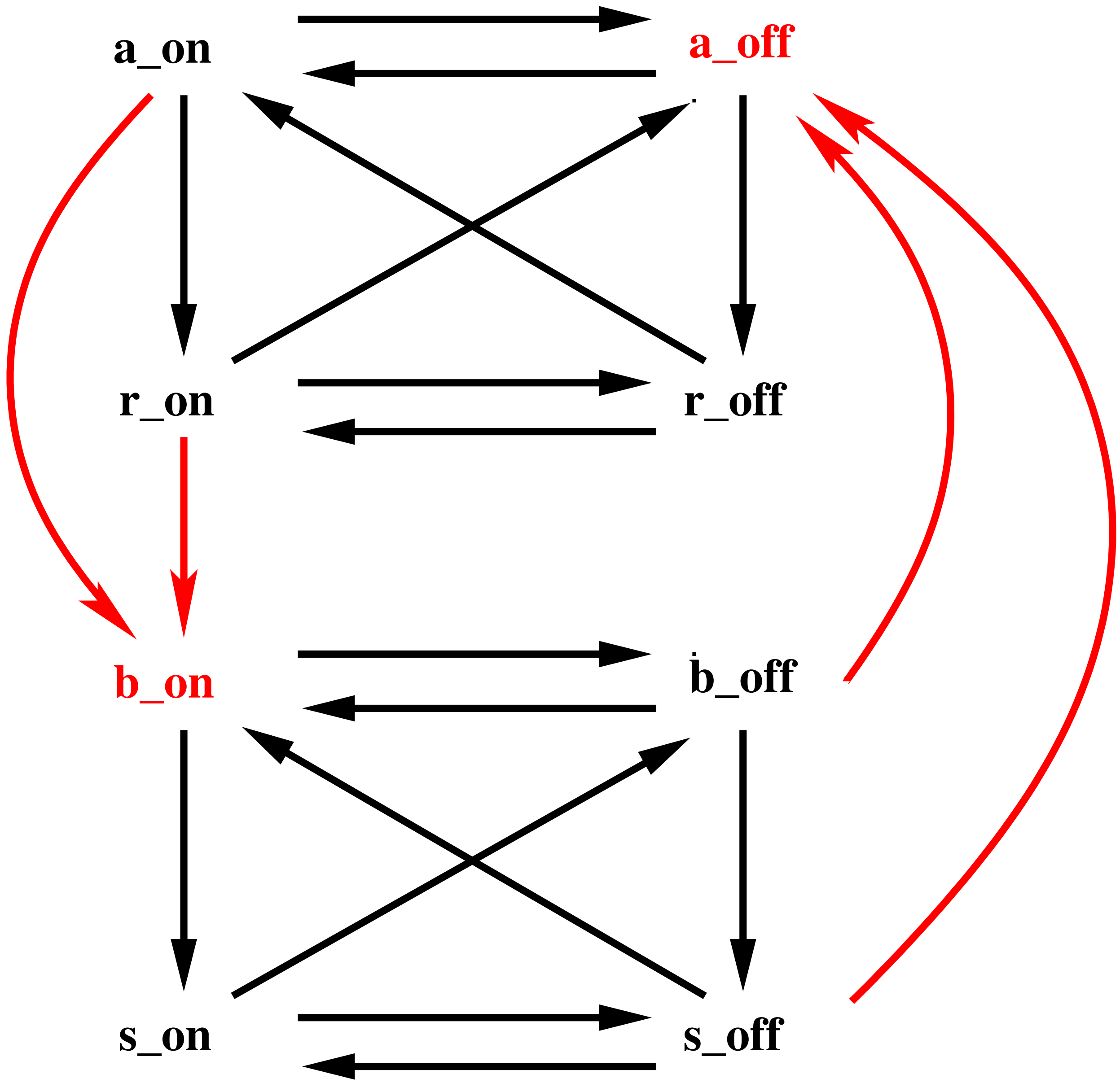
dbl1_5: $b = 1 \vee s = 1 \Rightarrow a = 1 \wedge r = 1$

dbl1_6: $a = 0 \vee r = 0 \Rightarrow b = 0 \wedge s = 0$

Weak Synchronization of Strong Reaction: the Problem



Weak Synchronization of Strong Reaction: the Solution



When the clutch is engaged, the motor must work	SAF_1

inv3_1: $clutch\_sensor = engaged$
$\Rightarrow$
$motor\_sensor = working$

- This is an instance of the previous design pattern

- We instantiate the pattern as follows:

a	$\leadsto$	$motor\_actuator$	a_on	$\leadsto$	treat_push_start_motor_button
r	$\leadsto$	$motor\_sensor$	a_off	$\leadsto$	treat_push_stop_motor_button
0	$\leadsto$	$stopped$	r_on	$\leadsto$	Motor_start
1	$\leadsto$	$working$	r_off	$\leadsto$	Motor_stop

b	$\leadsto$	$clutch\_actuator$	b_on	$\leadsto$	treat_start_clutch
s	$\leadsto$	$clutch\_sensor$	b_off	$\leadsto$	treat_stop_clutch
0	$\leadsto$	$disengaged$	s_on	$\leadsto$	Clutch_start
1	$\leadsto$	$engaged$	s_off	$\leadsto$	Clutch_stop

Translating the pattern invariants (1)

dbl1_1: $s = 1 \;\Rightarrow\; r = 1$
dbl1_2: $b = 1 \;\Rightarrow\; r = 1$

inv3_1:	$clutch\_sensor = engaged$ $\Rightarrow$ $motor\_sensor = working$
inv3_2:	$clutch\_actuator = engaged$ $\Rightarrow$ $motor\_sensor = working$

Translating the pattern invariants (2)

dbl1_3: $a = 0 \Rightarrow s = 0$
dbl1_4: $a = 0 \Rightarrow b = 0$

inv3_3: $motor\_actuator = stopped \Rightarrow clutch\_sensor = disengaged$
inv3_4: $motor\_actuator = stopped \Rightarrow clutch\_actuator = disengaged$

Adapting the Events of the Pattern (1)

b_on
when
$b = 0$
$s = 0$
$r = 1$
$a = 1$
then
$b := 1$
end

treat_start_clutch
when
$clutch\_actuator = disengaged$
$clutch\_sensor = disengaged$
$motor\_sensor = working$
$motor\_actuator = working$
then
$clutch\_actuator := engaged$
end

Adapting the events of the pattern (2)

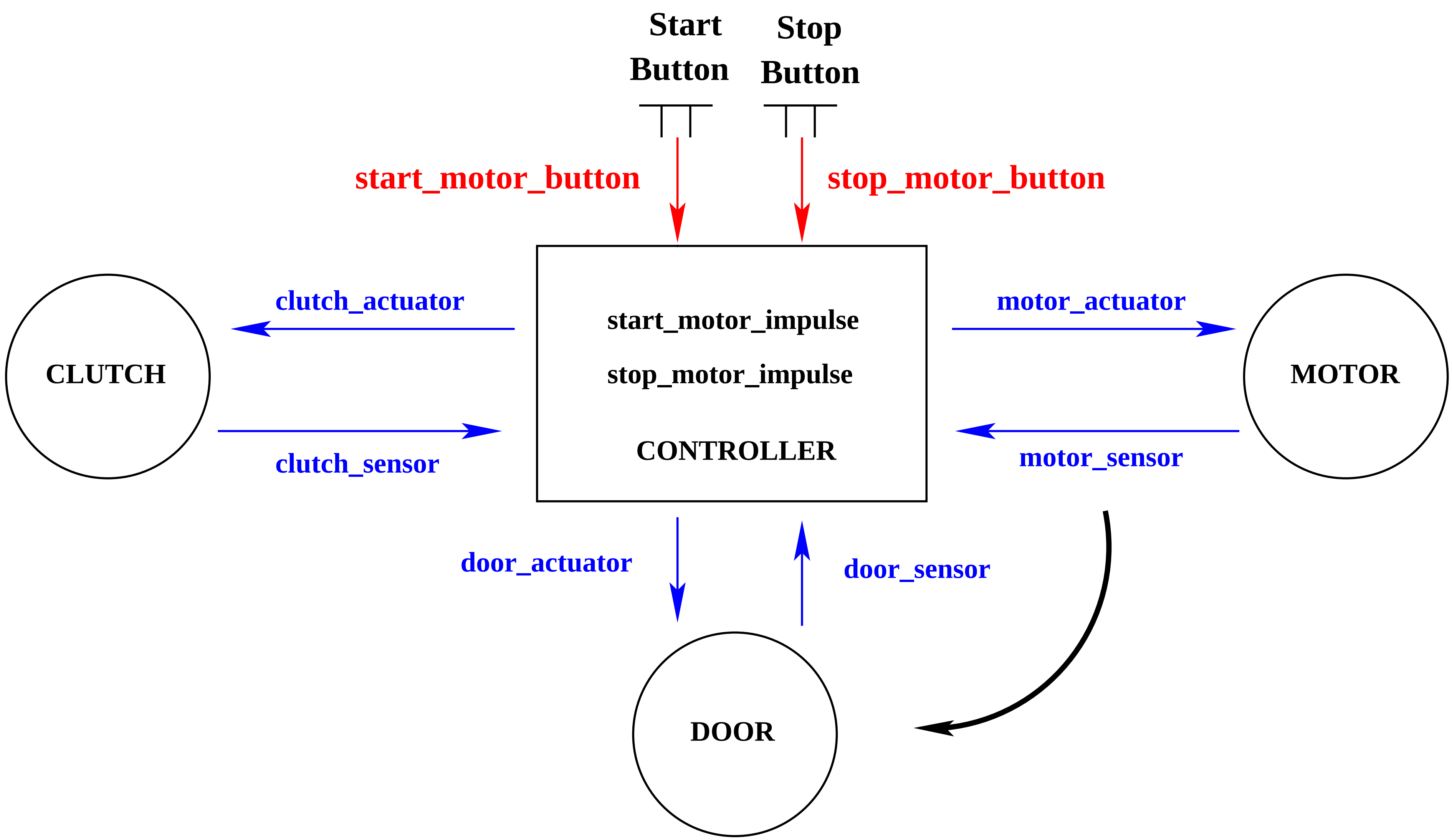
a_off **when** $a = 1$ $r = 1$ $s = 0$ $b = 0$ **then** $a := 0$ **end**	treat_push_stop_motor_button **when** $stop\_motor\_impulse = \mathrm{FALSE}$ $stop\_motor\_button = \mathrm{TRUE}$ $motor\_actuator = working$ $motor\_sensor = working$ $clutch\_sensor = disengaged$ $clutch\_actuator = disengaged$ **then** $motor\_actuator := stopped$ $stop\_motor\_impulse := \mathrm{TRUE}$ **end**

Third Refinement: Summary of the Events (1)

- Environment (no new events)
  - motor_start
  - motor_stop
  - clutch_start
  - clutch_stop
  - push_start_motor_button
  - release_start_motor_button
  - push_stop_motor_button
  - release_stop_motor_button

- Controller (no new events)
    - treat_push_start_motor_button
    - treat_push_start_motor_button_false
    - treat_push_stop_motor_button
    - treat_push_stop_motor_button_false
    - treat_release_start_motor_button
    - treat_release_stop_motor_button
    - treat_start_clutch
    - treat_stop_clutch

Fourth Refinement: Connecting the Controller to the Door



- We copy (after renaming "motor" to "door") what has been done in the initial model

- We introduce the set in a new context:

$$DOOR = \{open, closed\}$$

- We copy the initial model where we instantiate:

$$\begin{array}{lcl} motor & \leadsto & door \\ STATUS & \leadsto & DOOR \\ working & \leadsto & closed \\ stopped & \leadsto & open \end{array}$$

Fourth Refinement: Summary of the Events (1)

- Environment
  - motor_start
  - motor_stop
  - clutch_start
  - clutch_stop
  - door_close
  - door_open
  - push_start_motor_button
  - release_start_motor_button
  - push_stop_motor_button
  - release_stop_motor_button

Fourth Refinement: Summary of the Events (2)

- Controller
    - treat_push_start_motor_button
    - treat_push_start_motor_button_false
    - treat_push_stop_motor_button
    - treat_push_stop_motor_button_false
    - treat_release_start_motor_button
    - treat_release_stop_motor_button
    - treat_start_clutch
    - treat_stop_clutch
    - treat_close_door
    - treat_open_door

- An additional safety constraint

When the clutch is engaged, the door must be closed	SAF_2

- We copy (after renaming "motor" to "door") what has been done in the third model:

When the clutch is engaged, the motor must work	SAF_1

- Can you guess it?

Fifth Reft.: Something was forgotten Concerning the Door

- Can you guess it?

- When the motor is not working, we must allow users:
  - to change the tool
  - to replace the part to be treated

Fifth Reft.: Something was forgotten Concerning the Door

- Can you guess it?

- When the motor is not working, we must allow users:
  - to change the tool
  - to replace the part to be treated

- Hence the following additional requirement (which was forgotten)

When the motor is stopped, the door must be open	SAF_3

Fifth Reft.: Something was forgotten Concerning the Door

- Can you guess it?

- When the motor is not working, we must allow users:
  - to change the tool
  - to replace the part to be treated

- Hence the following additional requirement (which was forgotten)

When the door is closed, the motor must work	SAF_3'

- SAF_3' is the contraposed form of SAF_3

- Additional safety constraint

When the door is closed, the motor must work	SAF_3'

- We copy (after renaming "clutch" to "door") what has been done in the third model:

When the clutch is engaged, the motor must work	SAF_1

Fifth Reft.: Summary of the Safety Requirements

When the clutch is engaged, the motor must work	SAF_1

When the clutch is engaged, the door must be closed	SAF_2

When the door is closed, the motor must work	SAF_3'

- Requirement SAF_1 is now redundant: $SAF\_2 \wedge SAF\_3' \Rightarrow SAF\_1$

Possible New Refinement Strategy

- Initial model: Connecting the controller to the motor

- 1st refinement: Connecting the motor button to the controller

- 2nd refinement: Connecting the controller to the clutch

- 3rd (4th) refinement: Connecting the controller to the door

Possible New Refinement Strategy (cont'd)

- 4th (5th) refinement: Constraining the clutch and the door
  Constraining the motor and the door

- 5th (6th) refinement: More constraints between clutch and door

- 6th (7th) refinement: Connecting the clutch button to the controller

Fifth Refinement: Summary of the Events (1)

- Environment (no new events)
  - motor_start
  - motor_stop
  - clutch_start
  - clutch_stop
  - door_close
  - door_open
  - push_start_motor_button
  - release_start_motor_button
  - push_stop_motor_button
  - release_stop_motor_button

Fifth Refinement: Summary of the Events (2)

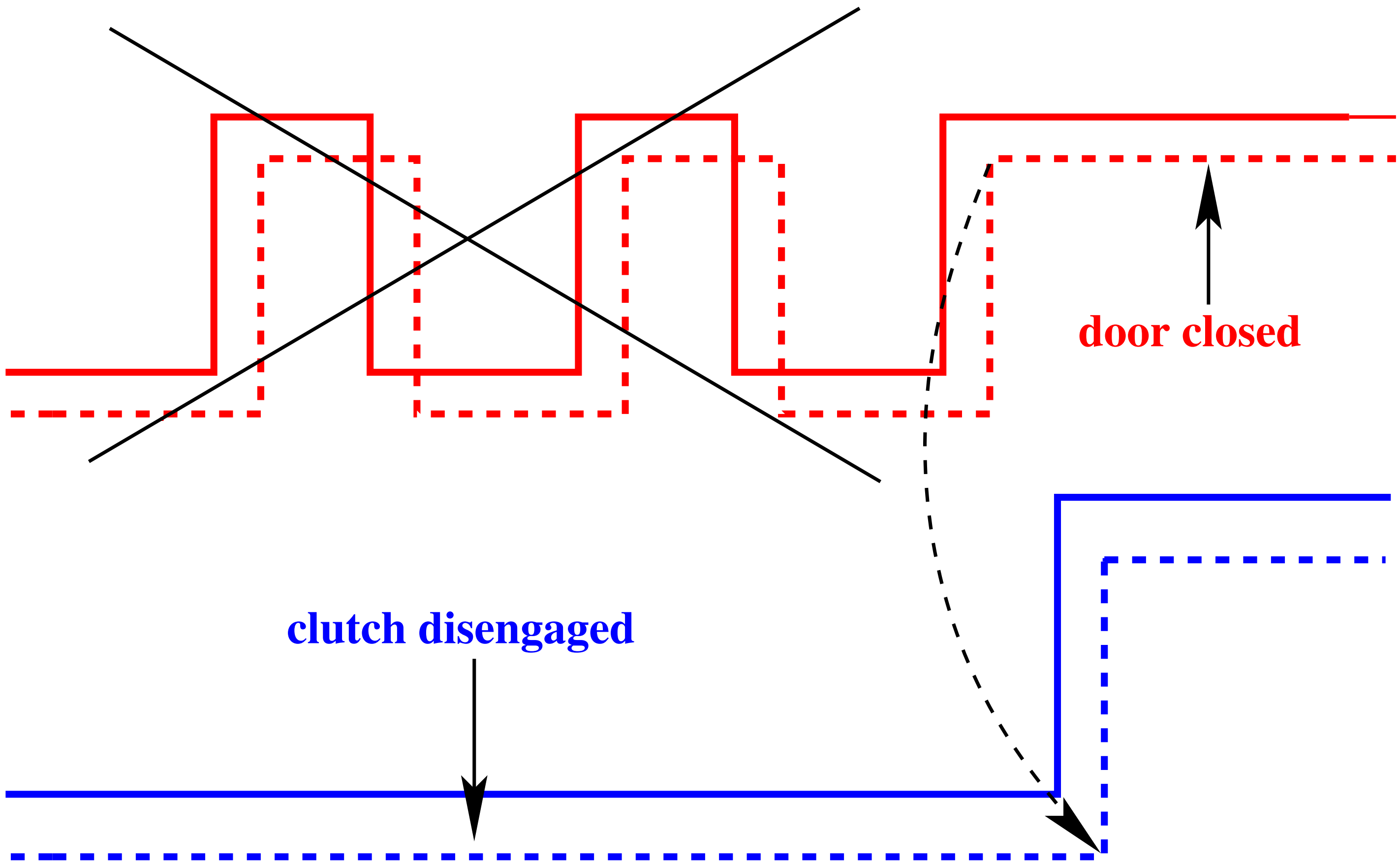
- Controller (no new events)
  - treat_push_start_motor_button
  - treat_push_start_motor_button_false
  - treat_push_stop_motor_button
  - treat_push_stop_motor_button_false
  - treat_release_start_motor_button
  - treat_release_stop_motor_button
  - treat_start_clutch
  - treat_stop_clutch
  - treat_close_door
  - treat_open_door

- Adding two functional constraints

When the clutch is disengaged, the door cannot be closed several times, ONLY ONCE	FUN_3

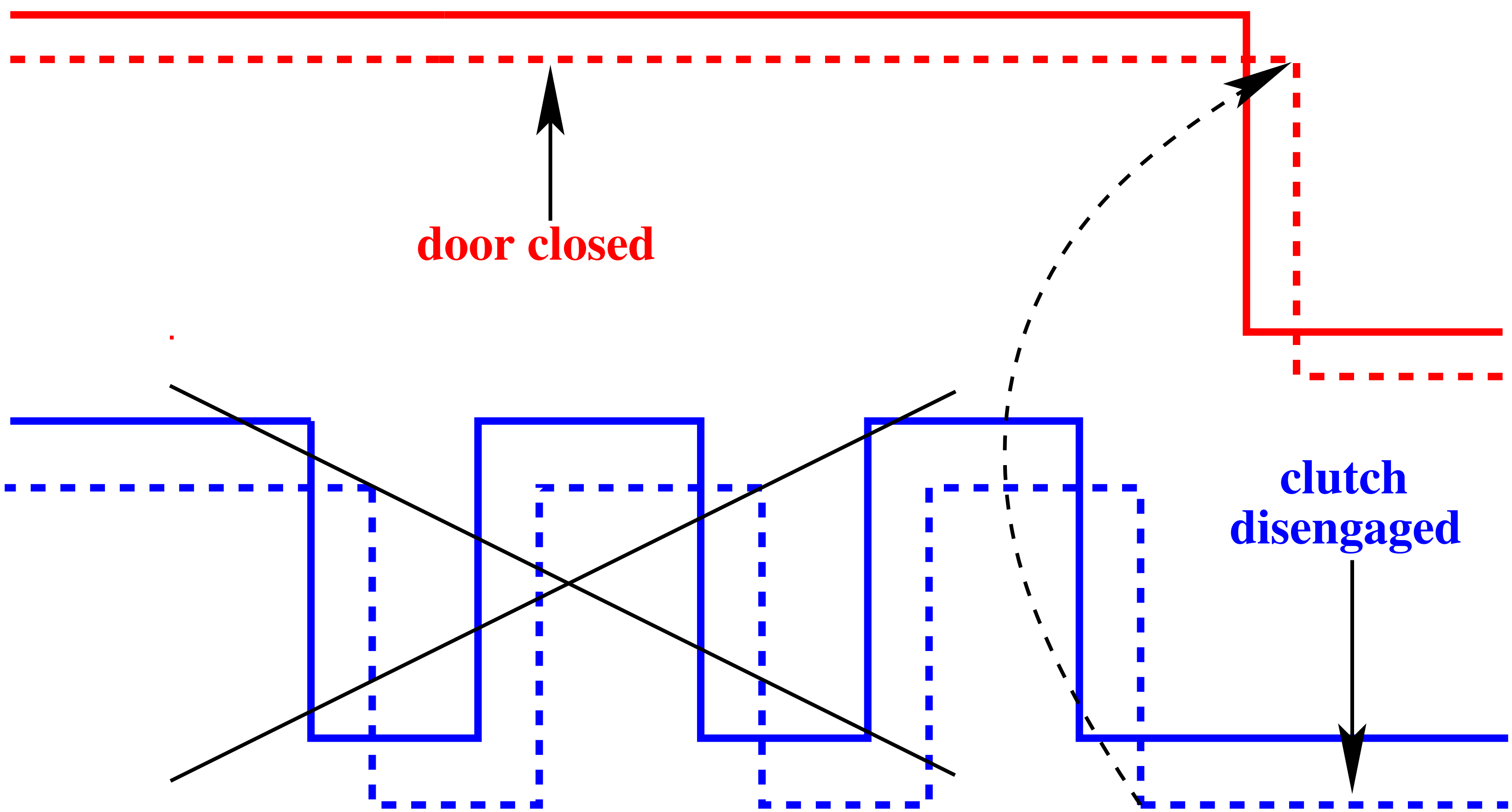
When the door is closed, the clutch cannot be disengaged several times, ONLY ONCE	FUN_4

Problem with the Weak Synchronization of Strong Reactions



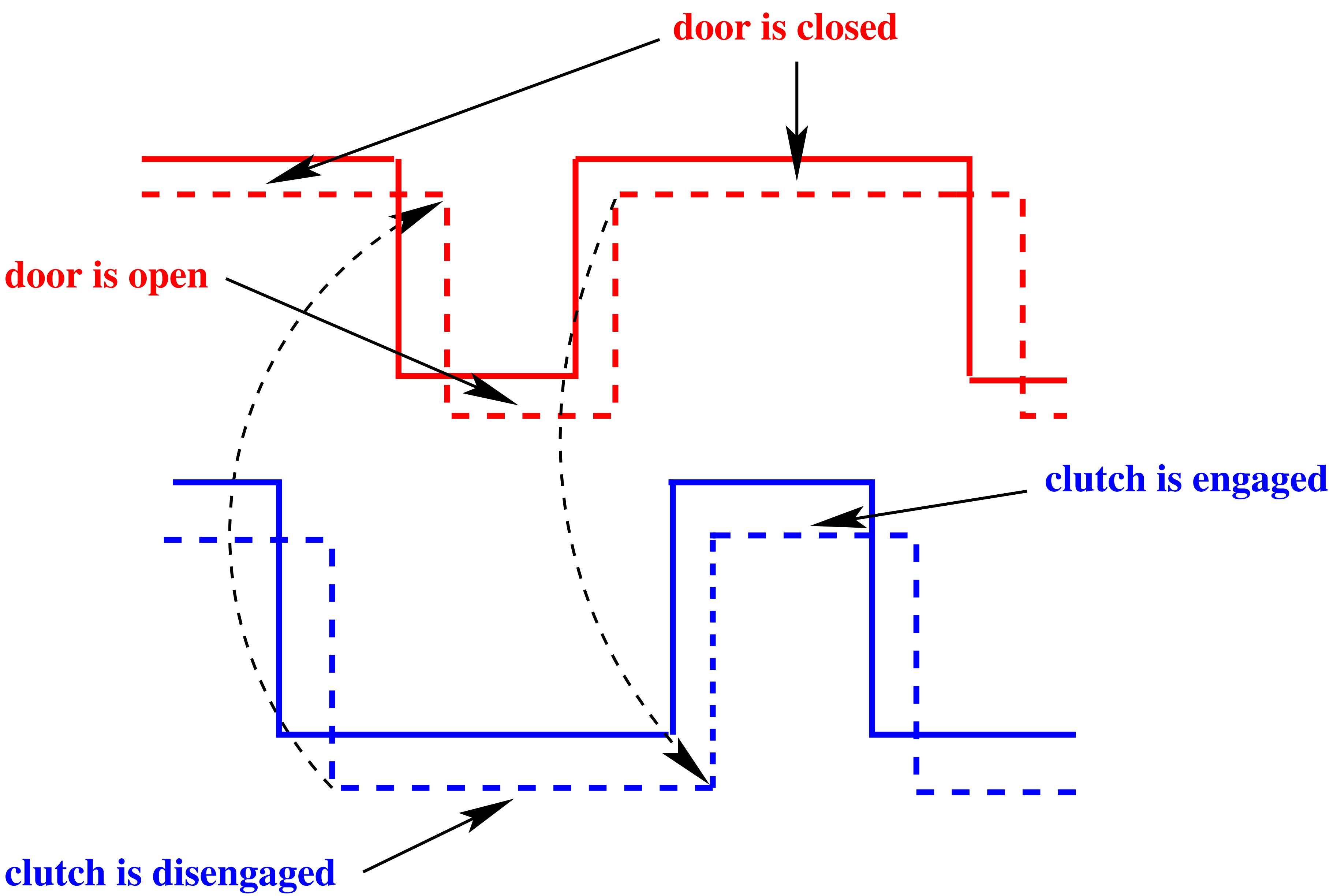
- When the clutch is disengaged, the door cannot be closed several times

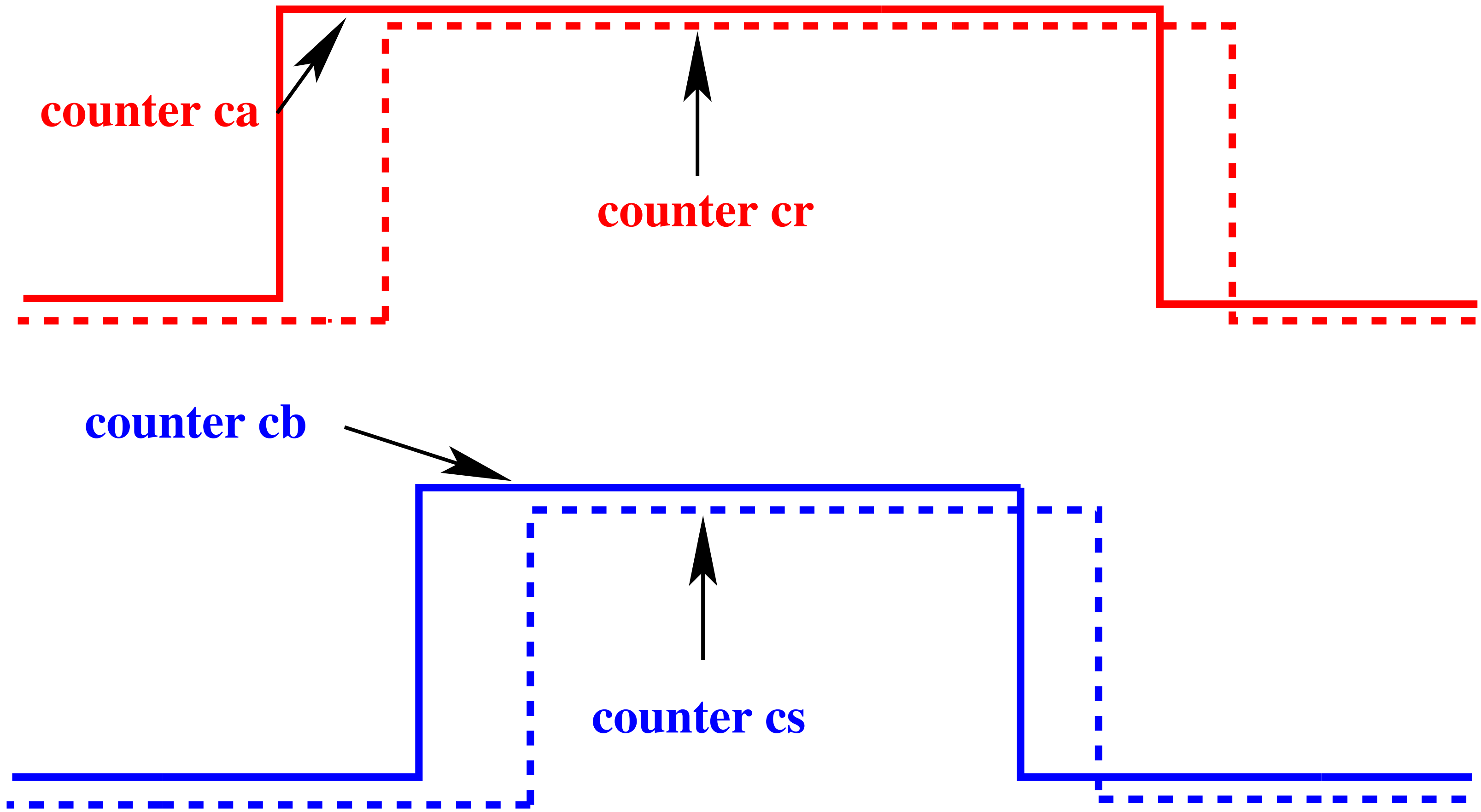
Problem with the Weak Synchronization of Strong Reactions



- When the door is closed, the clutch cannot be disengaged several times

The Full Picture



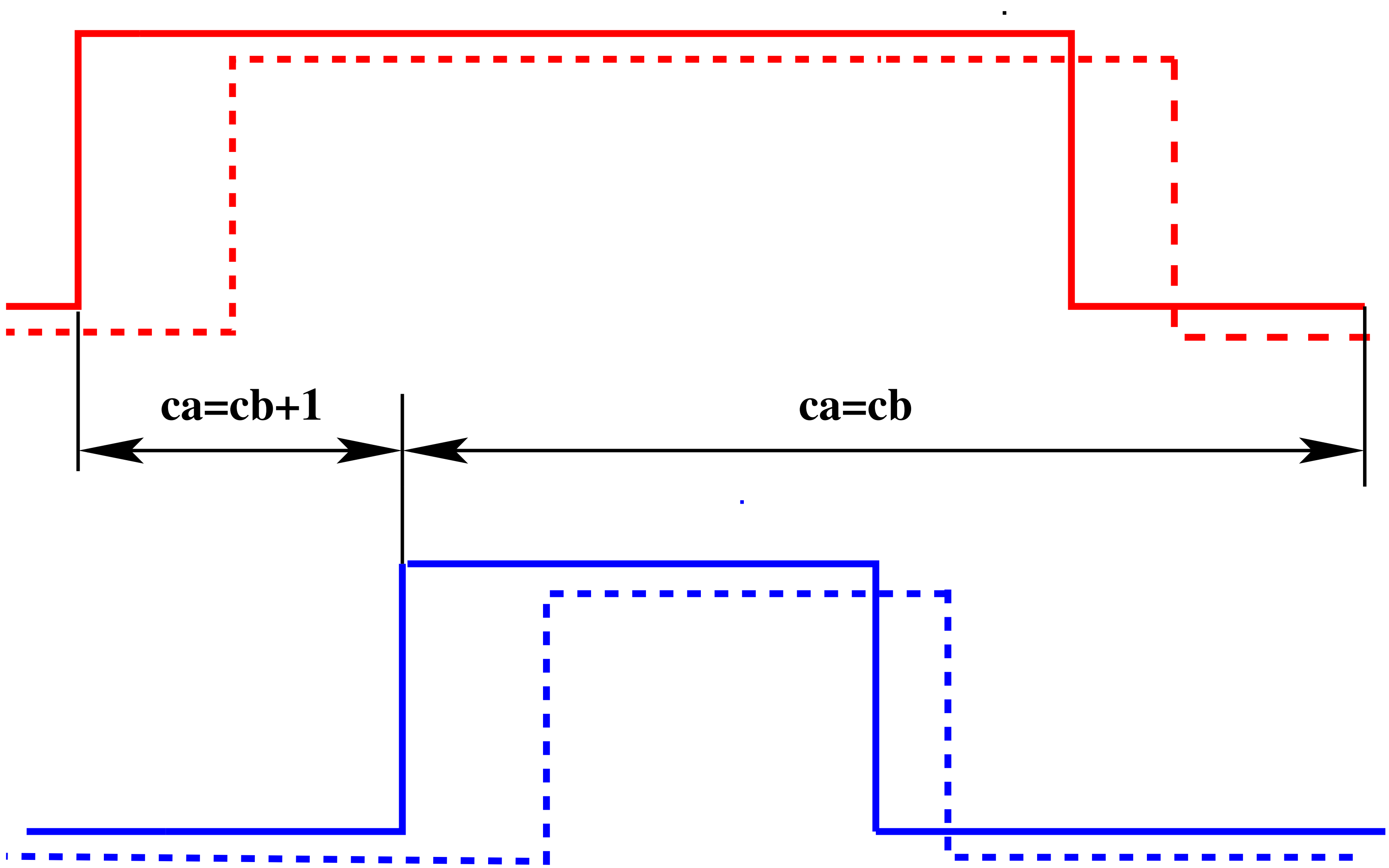


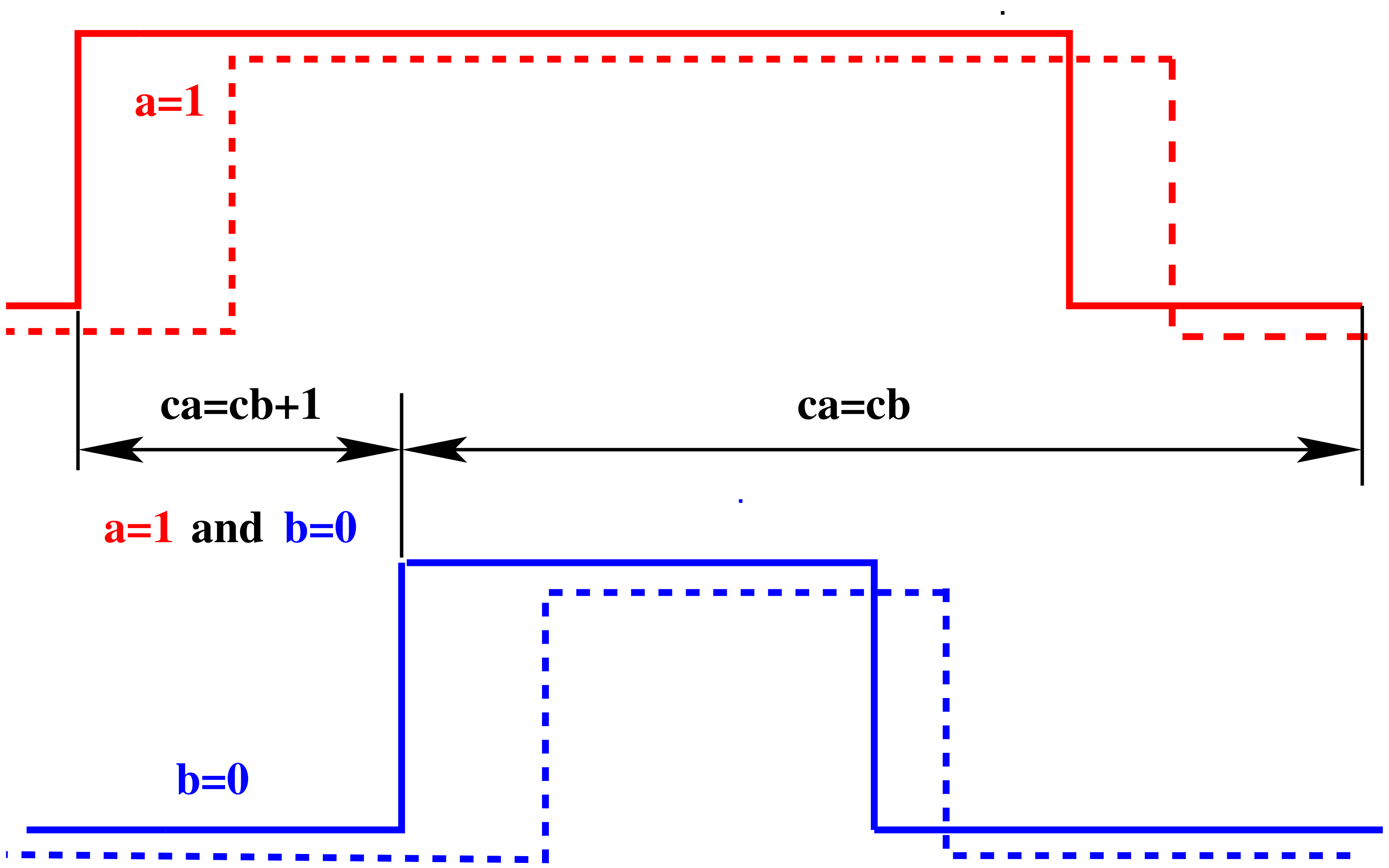
What we want:

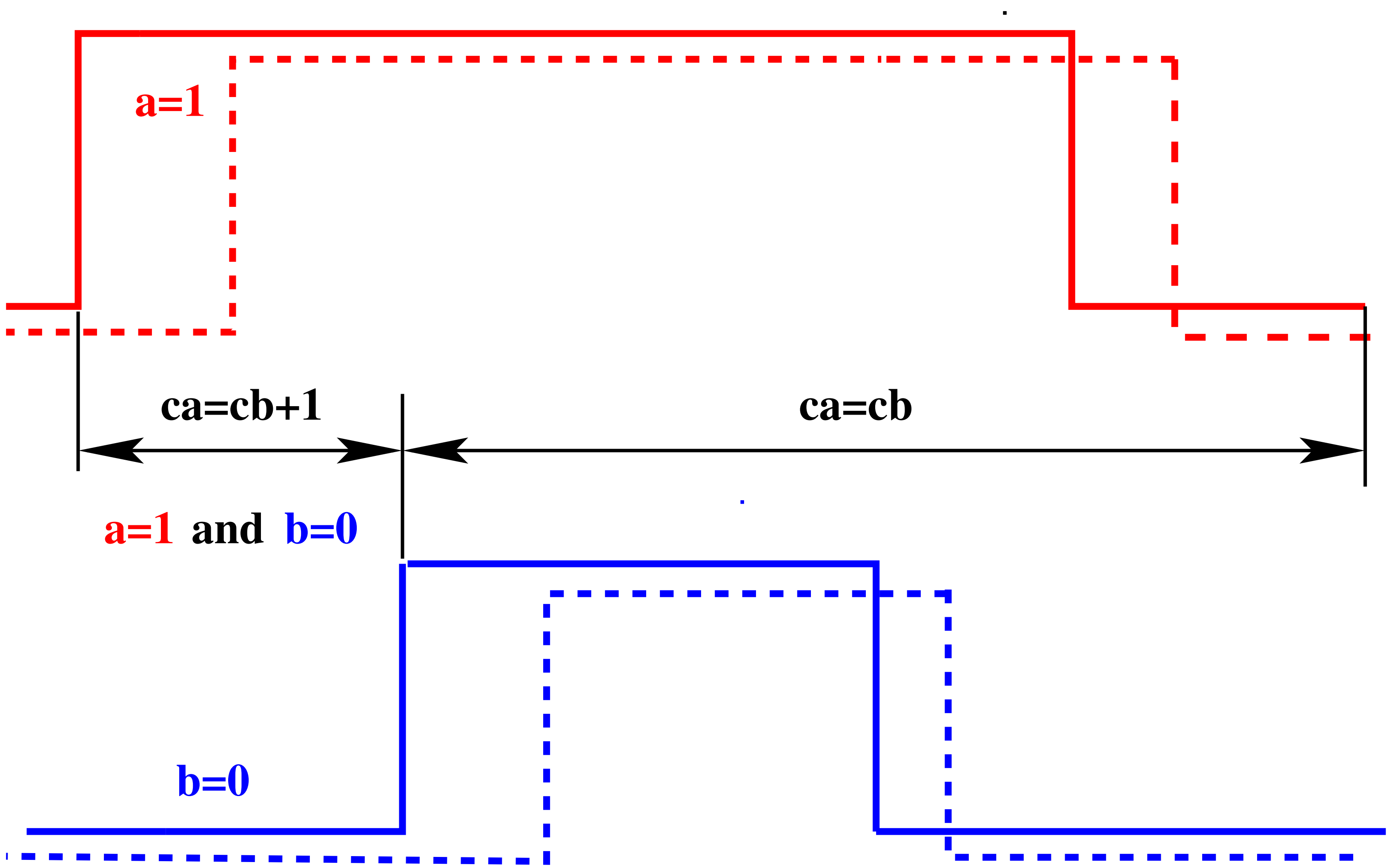
$$ca = cb \;\vee\; ca = cb + 1$$

$$cr = cs \;\vee\; cr = cs + 1$$

How about counters ca and cb?

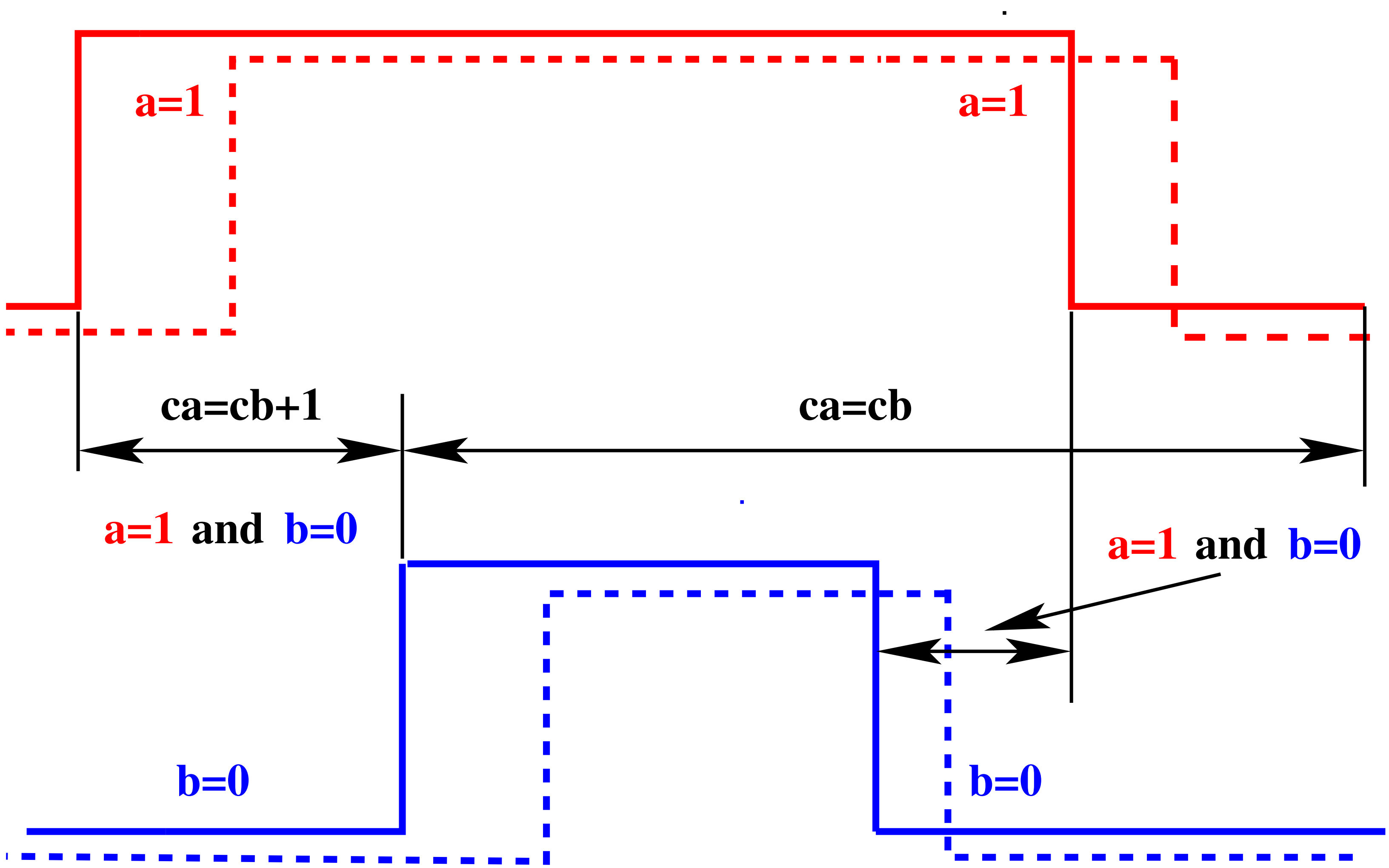



a=1
ca=cb+1
ca=cb
a=1 and b=0
b=0

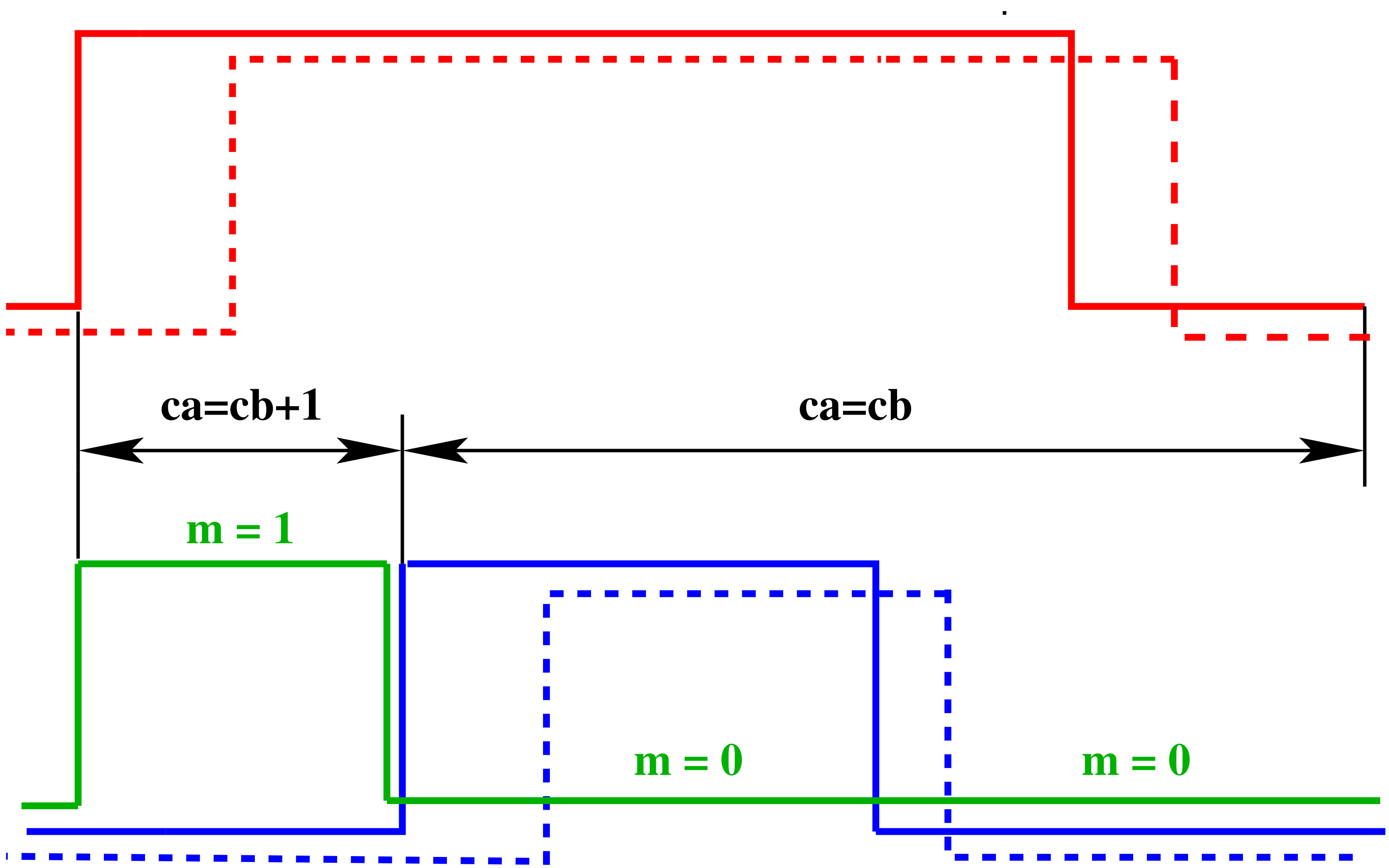


$$a = 1 \wedge b = 0 \Rightarrow ca = cb + 1 \qquad ?$$

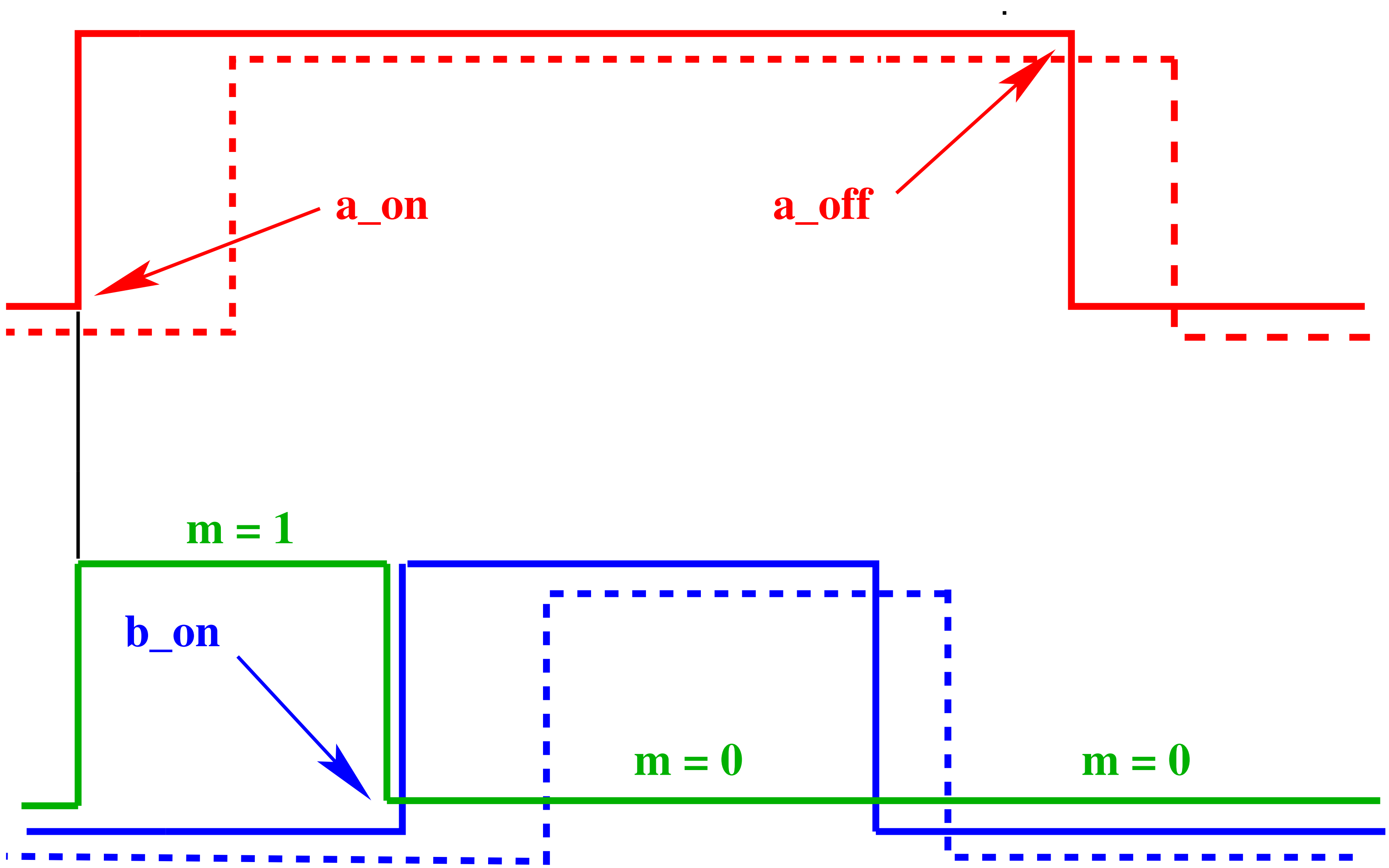
This Solution Does not Work



The Solution: an Additional Variable m



$$m = 1 \quad \Rightarrow \quad ca = cb + 1$$
$$m = 0 \quad \Rightarrow \quad ca = cb$$

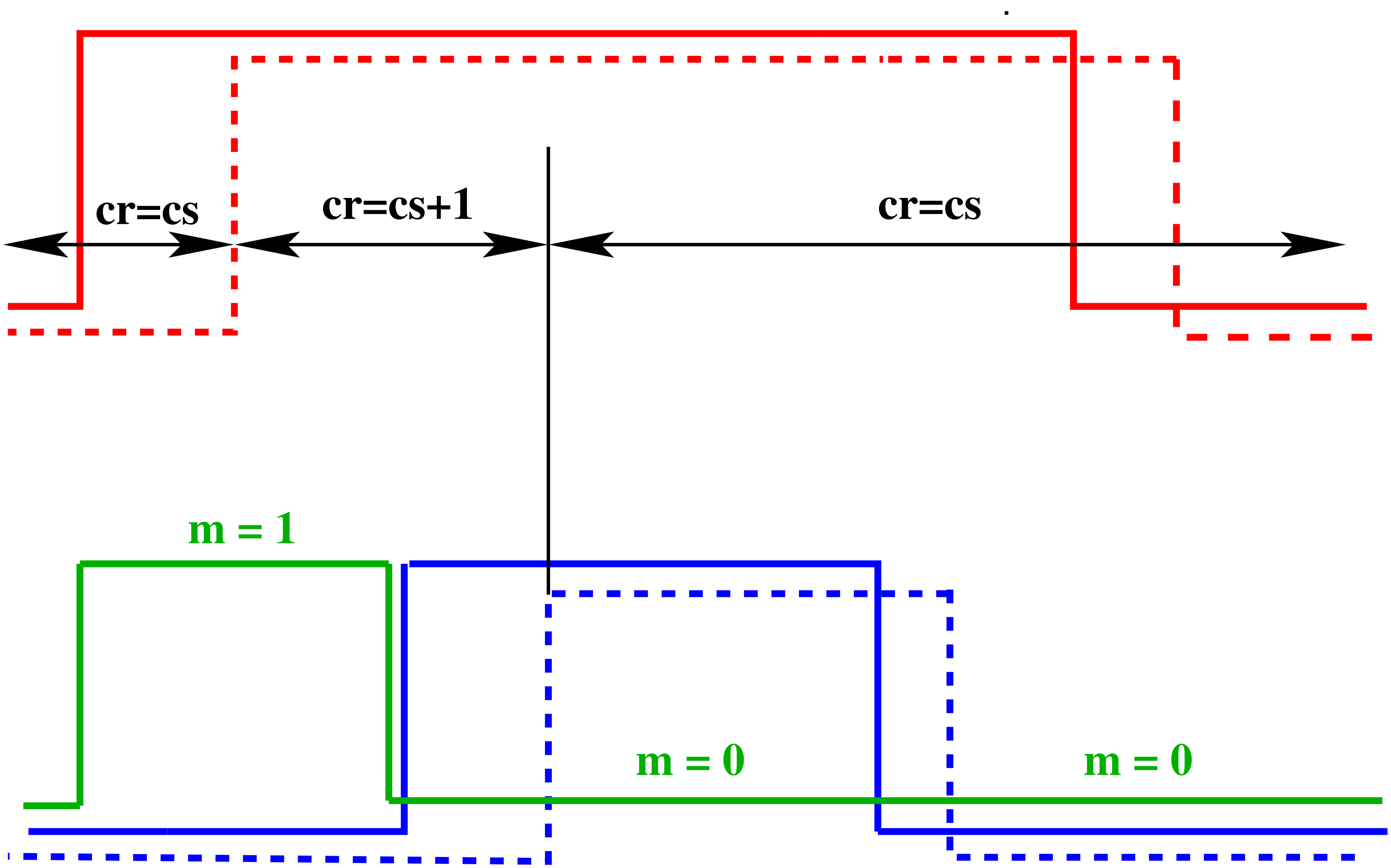

a_on
a_off
m = 1
b_on
m = 0
m = 0

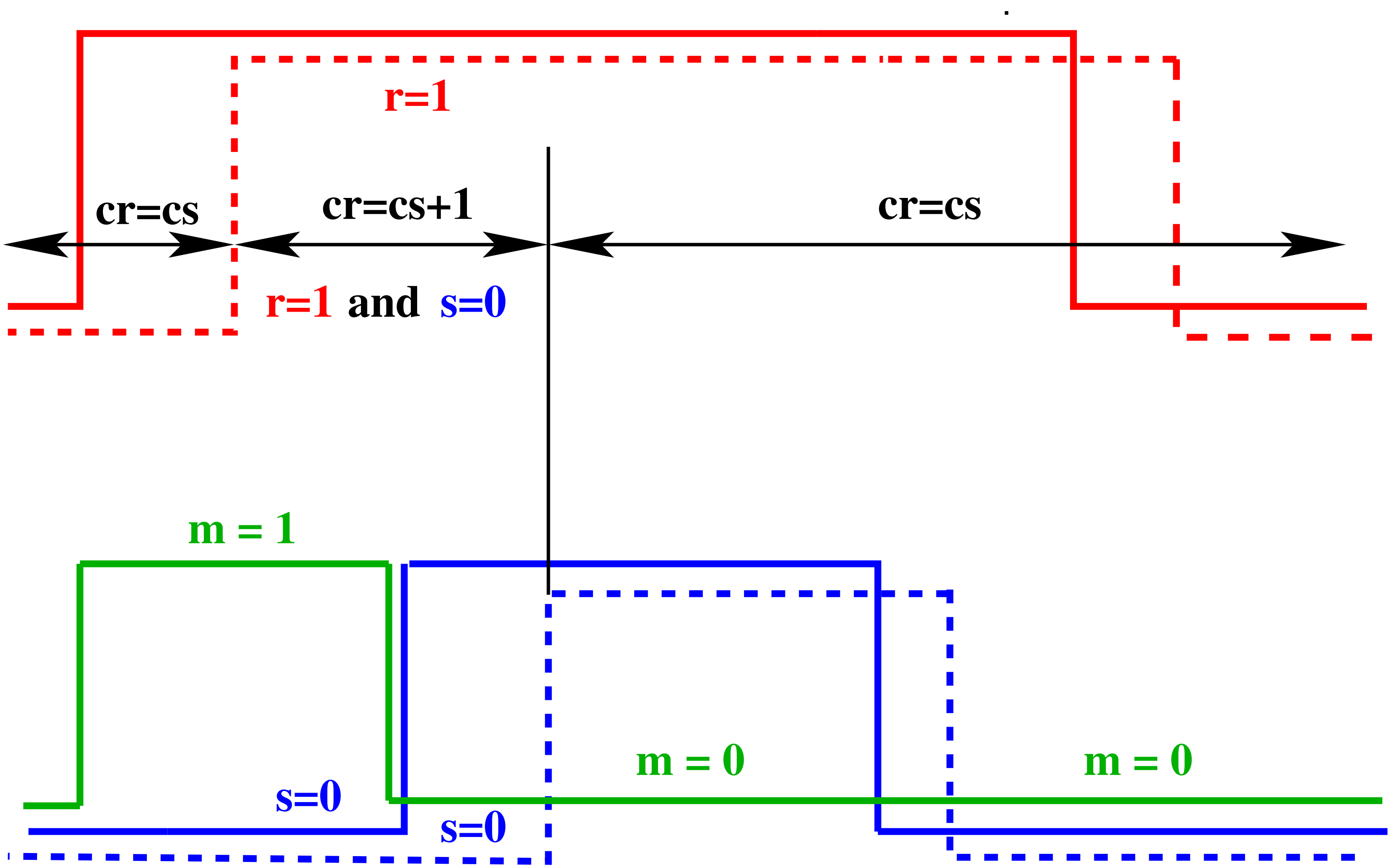
a_on
when
$a = 0$
$r = 0$
then
$a := 1$
$ca := ca + 1$
$m := 1$
end

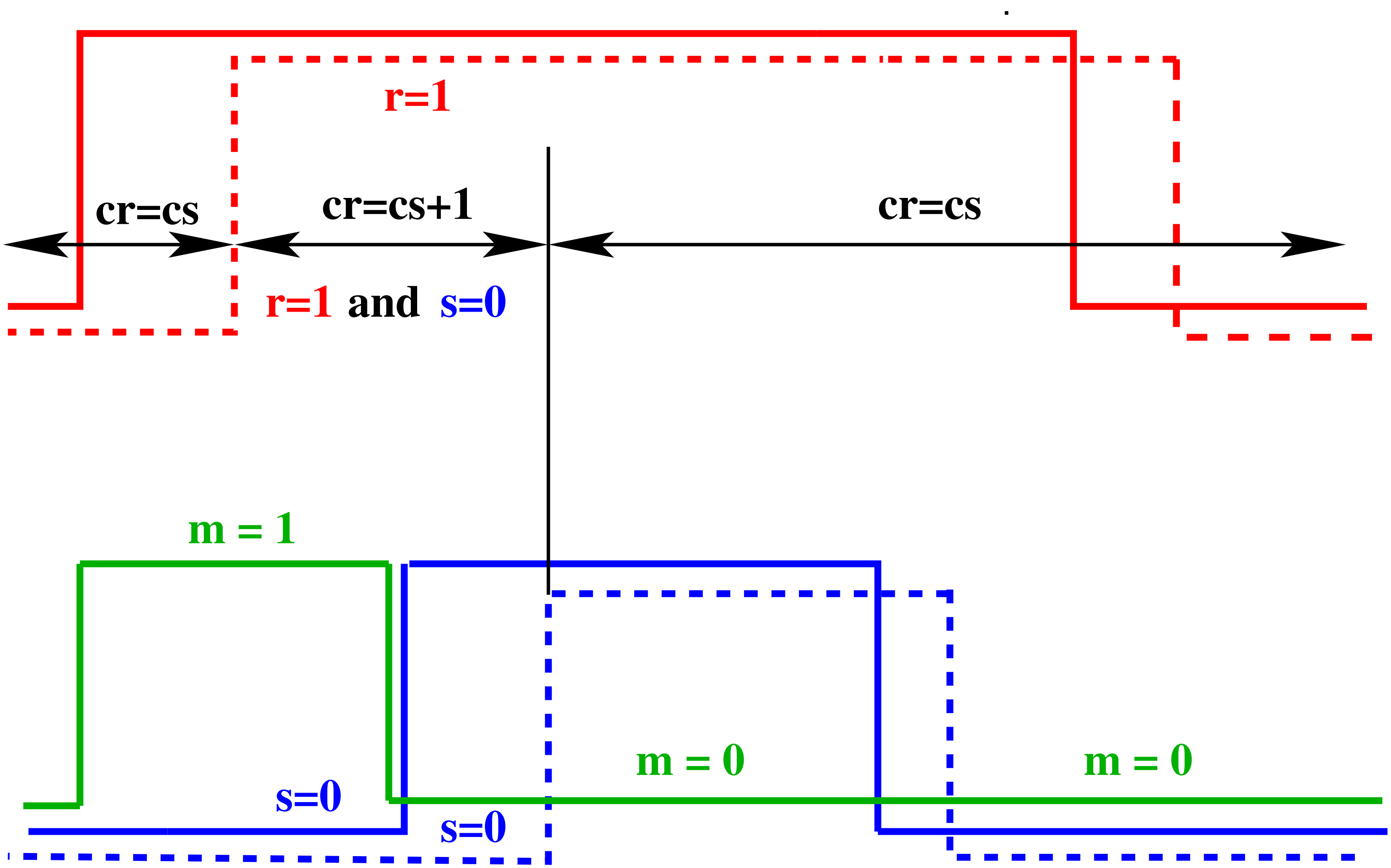
b_on
when
$r = 1$
$a = 1$
$b = 0$
$s = 0$
$m = 1$
then
$b := 1$
$cb := cb + 1$
$m := 0$
end

a_off
when
$a = 1$
$r = 1$
$b = 0$
$s = 0$
$m = 0$
then
$a := 0$
end

How about counters cr and cs

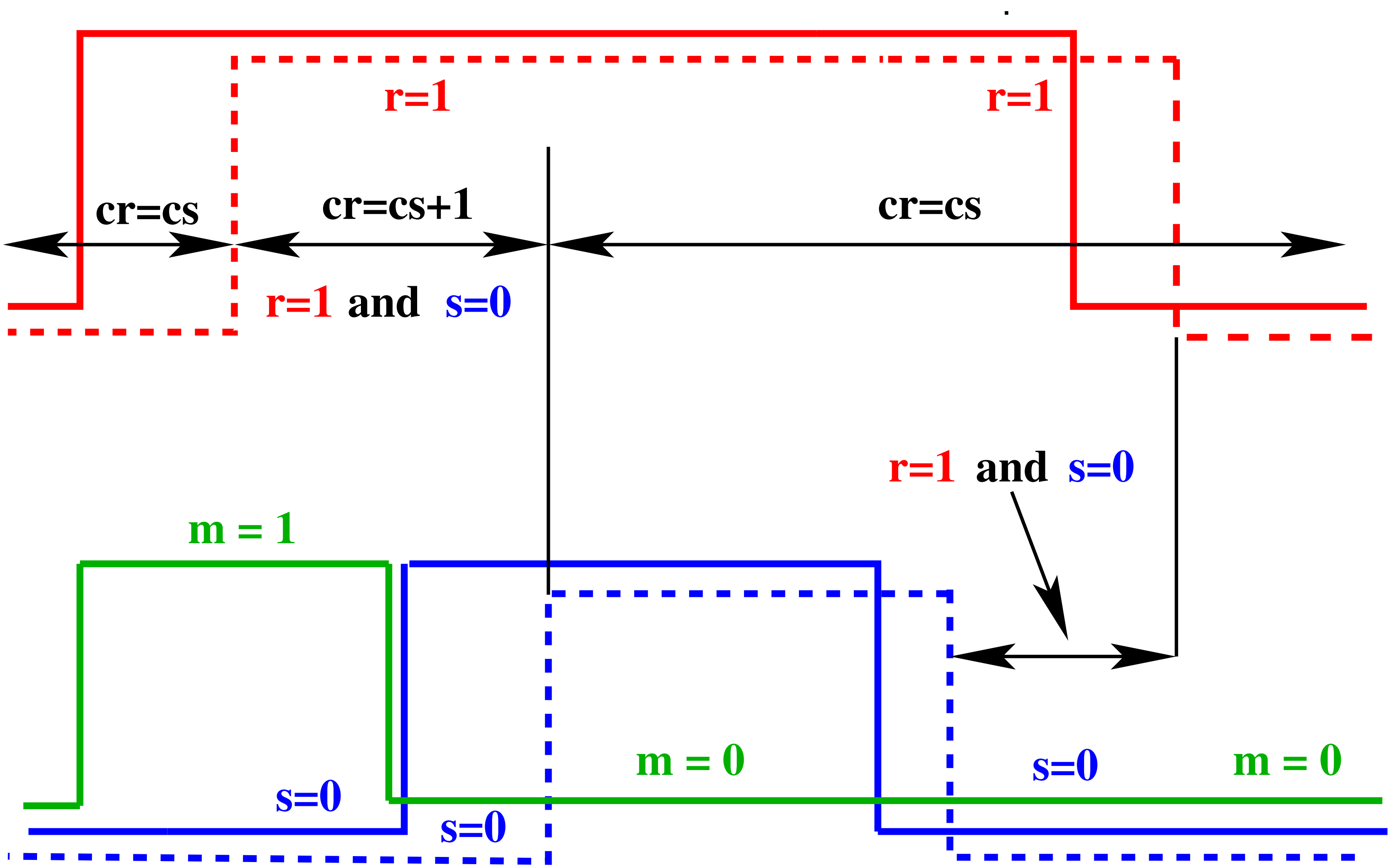


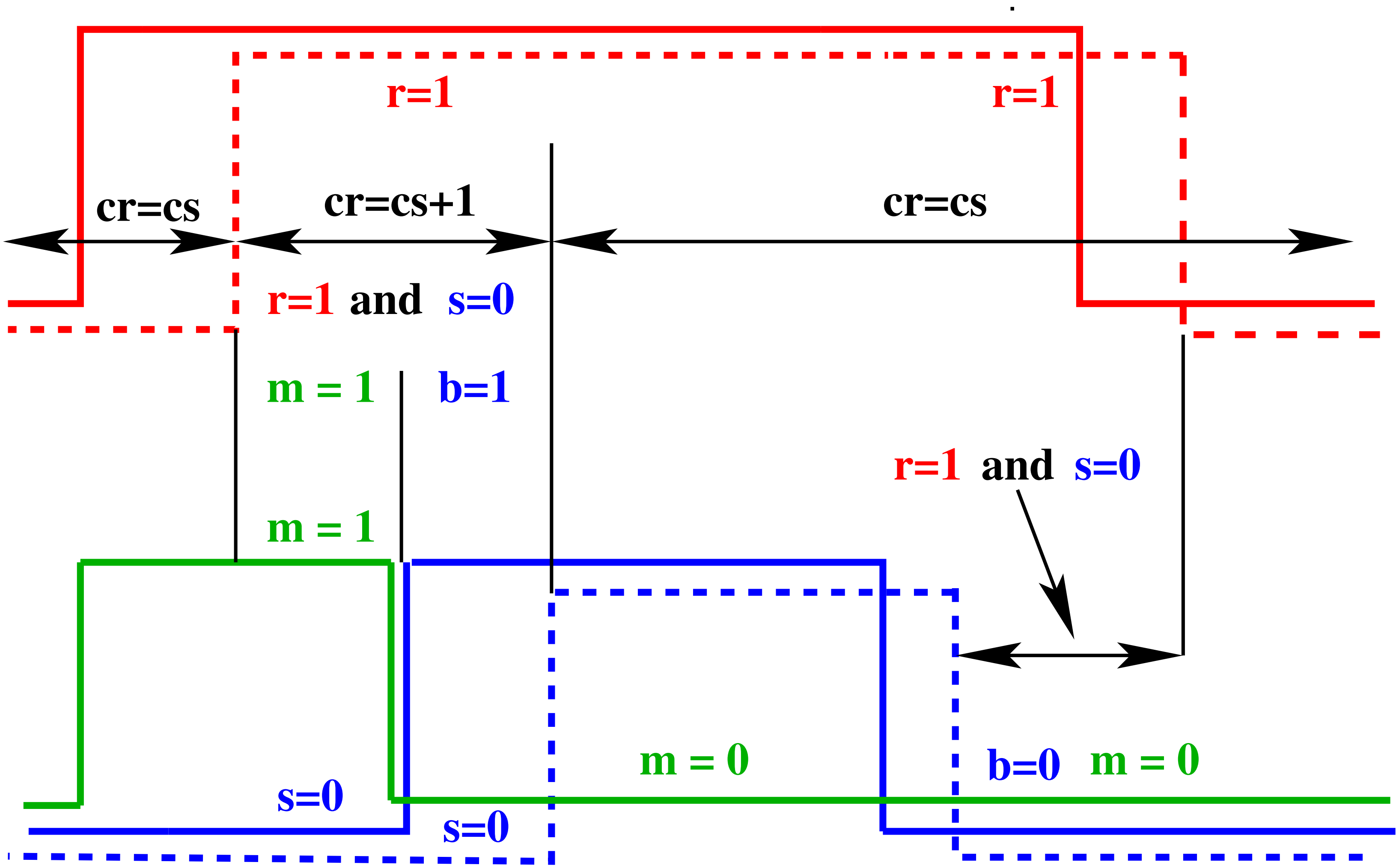

r=1
cr=cs
cr=cs+1
cr=cs
r=1 and s=0
m = 1
s=0
s=0
m = 0
m = 0

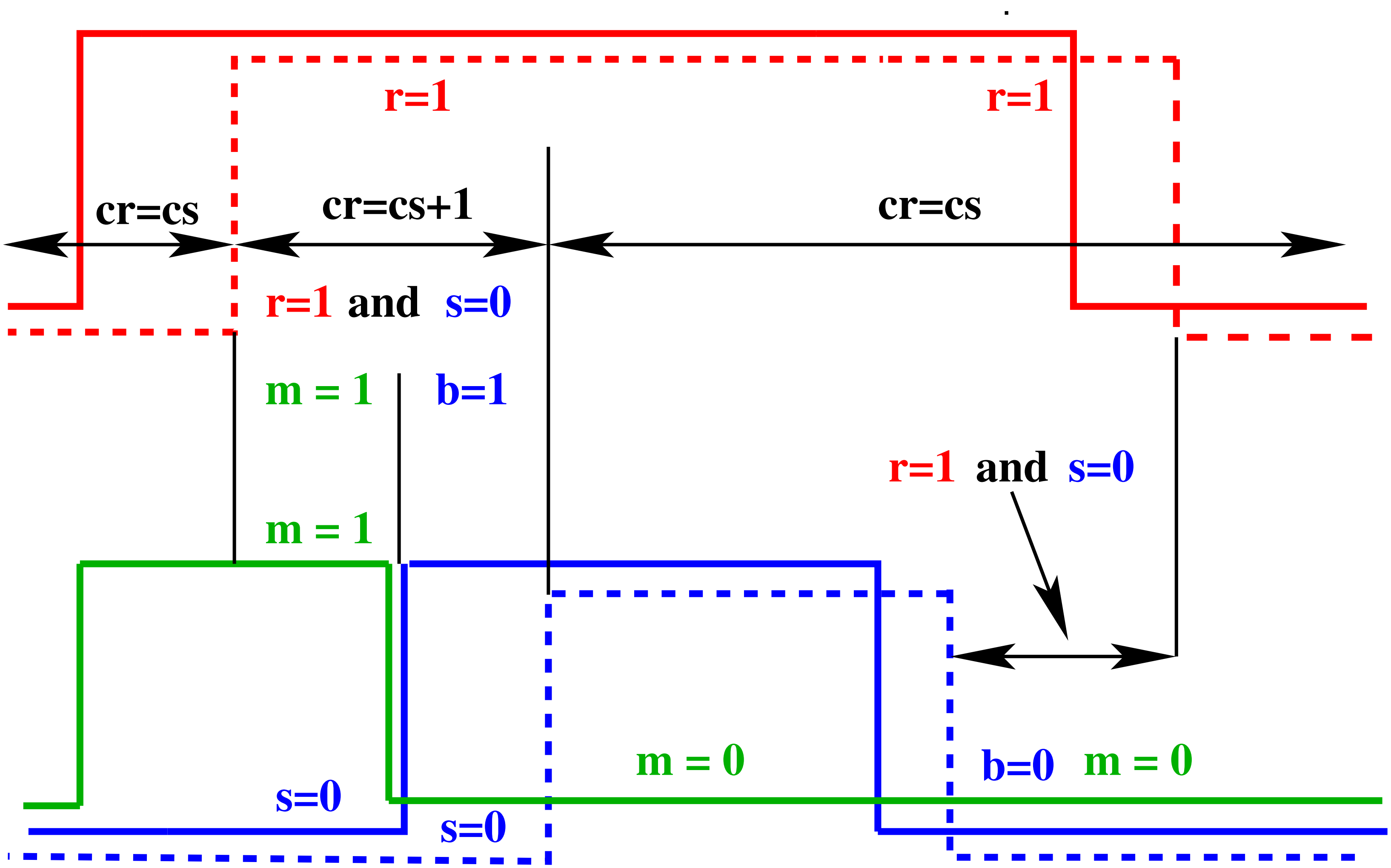


$$r = 1 \ \wedge \ s = 0 \ \Rightarrow \ cr = cs + 1 \qquad \textbf{?}$$

This Solution Does not Work




r=1
r=1
cr=cs
cr=cs+1
cr=cs
r=1 and s=0
m = 1
b=1
r=1 and s=0
m = 1
m = 0
b=0
m = 0
s=0
s=0



$$r = 1 \ \wedge \ s = 0 \ \wedge \ (m = 1 \ \vee \ b = 1) \ \Rightarrow \ cr = cs + 1$$

$$r = 0 \ \vee \ s = 1 \ \vee \ (m = 0 \ \wedge \ b = 0) \ \Rightarrow \ cr = cs$$

dbl2_1: $m \in \{0, 1\}$

dbl2_2: $m = 1 \Rightarrow ca = cb + 1$

dbl2_3: $m = 0 \Rightarrow ca = cb$

dbl2_4: $r = 1 \wedge s = 0 \wedge (m = 1 \vee b = 1) \Rightarrow cr = cs + 1$

dbl2_5: $r = 0 \vee s = 1 \vee (m = 0 \wedge b = 0) \Rightarrow cr = cs$

Summary of Refinement: the state

dbl2_1: $m \in \{0, 1\}$

dbl2_2: $m = 1 \;\Rightarrow\; ca = cb + 1$

dbl2_3: $m = 0 \;\Rightarrow\; ca = cb$

dbl2_4: $r = 1 \;\wedge\; s = 0 \;\wedge\; (m = 1 \;\vee\; b = 1) \;\Rightarrow\; cr = cs + 1$

dbl2_5: $r = 0 \;\vee\; s = 1 \;\vee\; (m = 0 \;\wedge\; b = 0) \;\Rightarrow\; cr = cs$

- The following theorems are easy to prove

thm2_1: $ca = cb \;\vee\; ca = cb + 1$

thm2_2: $cr = cs \;\vee\; cr = cs + 1$

More Invariants

dbl2_1: $m \in \{0, 1\}$

dbl2_2: $m = 1 \Rightarrow ca = cb + 1$

dbl2_3: $m = 0 \Rightarrow ca = cb$

dbl2_4: $r = 1 \wedge s = 0 \wedge (m = 1 \vee b = 1) \Rightarrow cr = cs + 1$

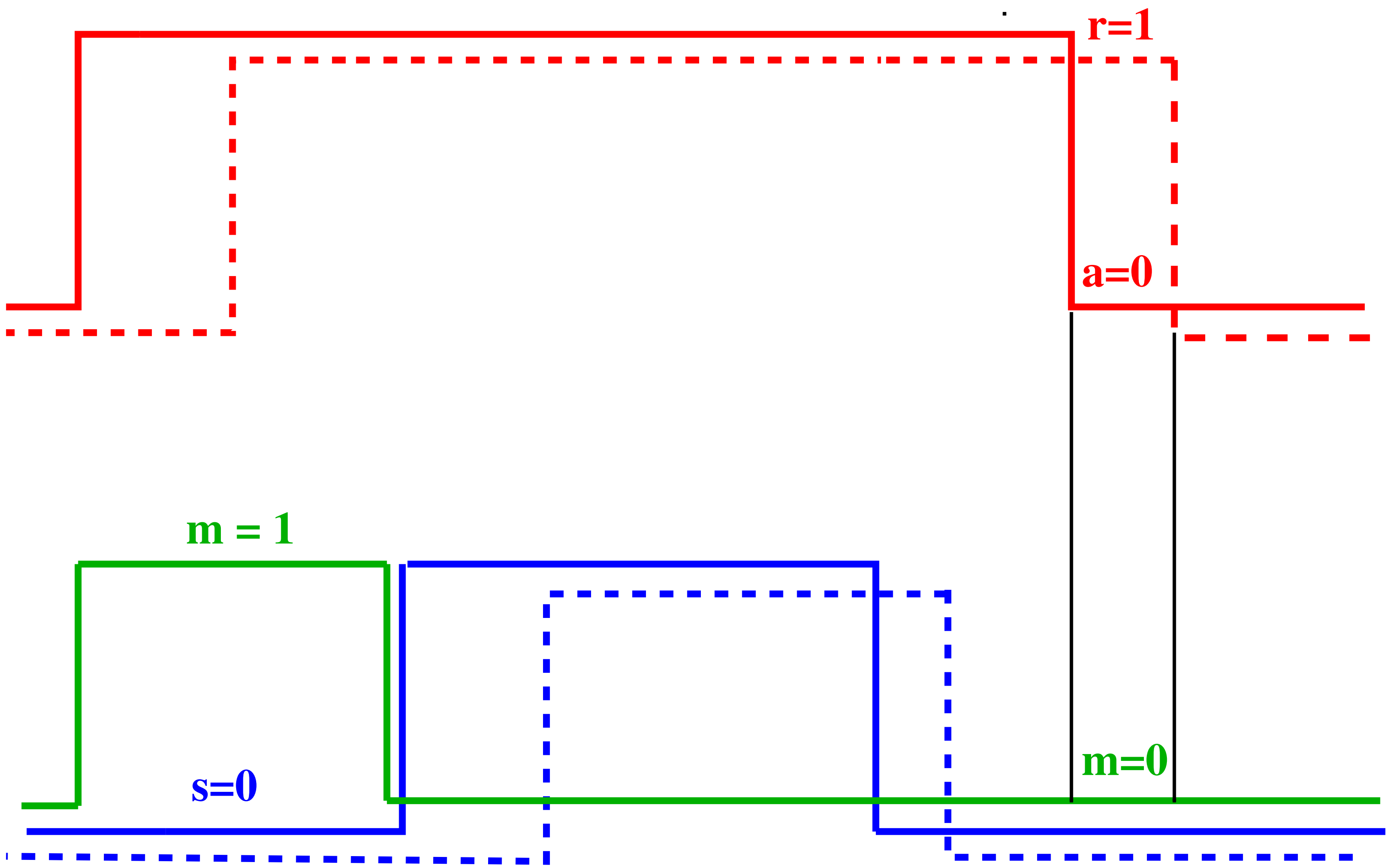
dbl2_5: $r = 0 \vee s = 1 \vee (m = 0 \wedge b = 0) \Rightarrow cr = cs$

dbl2_6: $r = 1 \wedge a = 0 \Rightarrow m = 0$

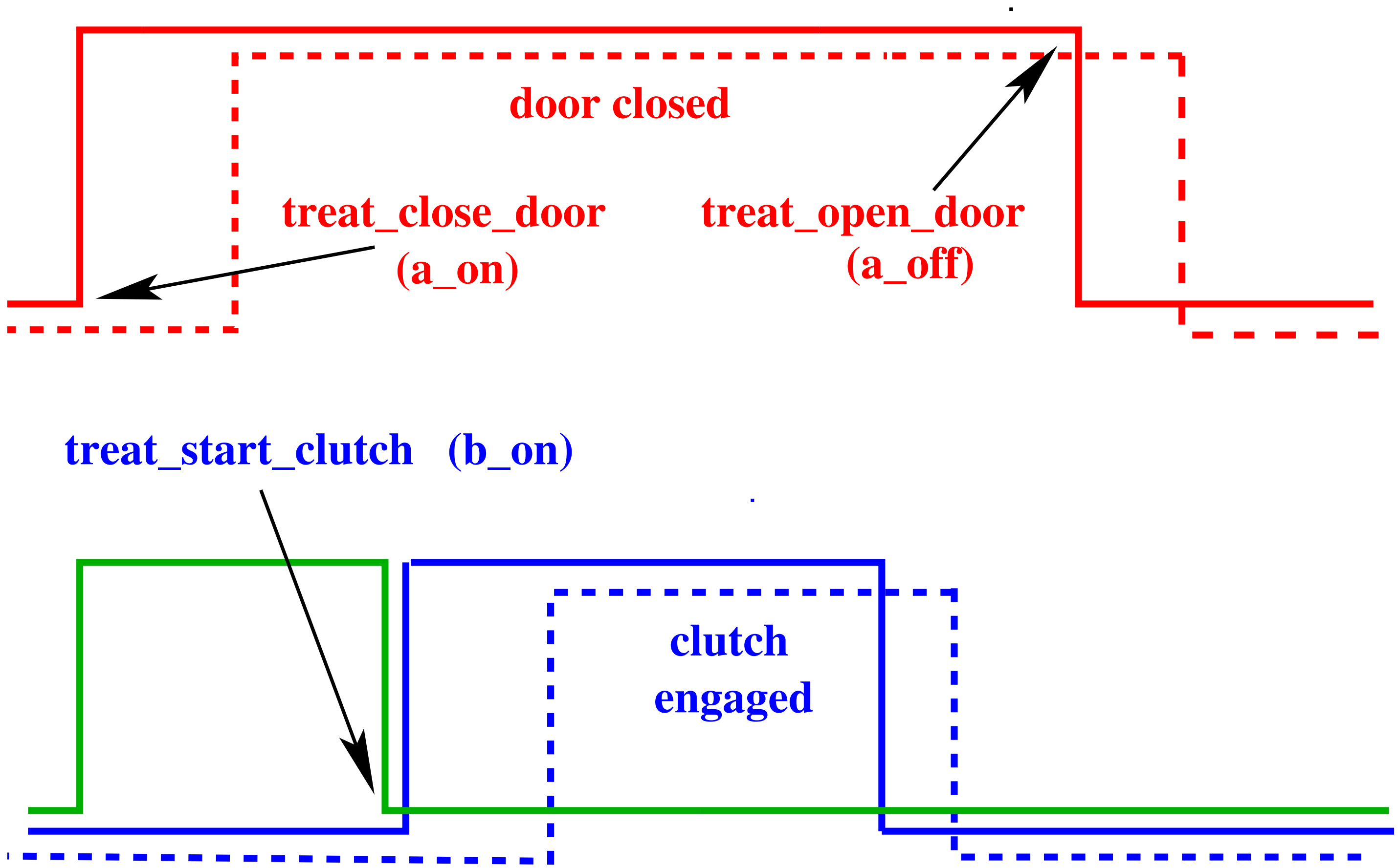
dbl2_7: $m = 1 \Rightarrow s = 0$

- The two new invariants were discovered while doing the proof

- The proofs are now completely automatic


r=1
a=0
m = 1
s=0
m=0

Instantiation



- We instantiate the pattern as follows:

$$
\begin{array}{lll}
a & \rightsquigarrow & door\_actuator \\
r & \rightsquigarrow & door\_sensor \\
0 & \rightsquigarrow & open \\
1 & \rightsquigarrow & closed
\end{array}
\qquad
\begin{array}{lll}
b & \rightsquigarrow & clutch\_actuator \\
s & \rightsquigarrow & clutch\_sensor \\
0 & \rightsquigarrow & disengaged \\
1 & \rightsquigarrow & engaged
\end{array}
$$

a_on	$\rightsquigarrow$	treat_close_door
a_off	$\rightsquigarrow$	treat_open_door
b_on	$\rightsquigarrow$	treat_start_clutch

6th Refinement: Adapting the events of the pattern (2)

a_on
when
$a = 0$
$r = 0$
then
$a := 1$
$m := 1$
end

treat_close_door
when
$door\_actuator = open$
$door\_sensor = open$
$motor\_actuator = working$
$motor\_sensor = working$
then
$door\_actuator := closed$
$m := 1$
end

b_on
when

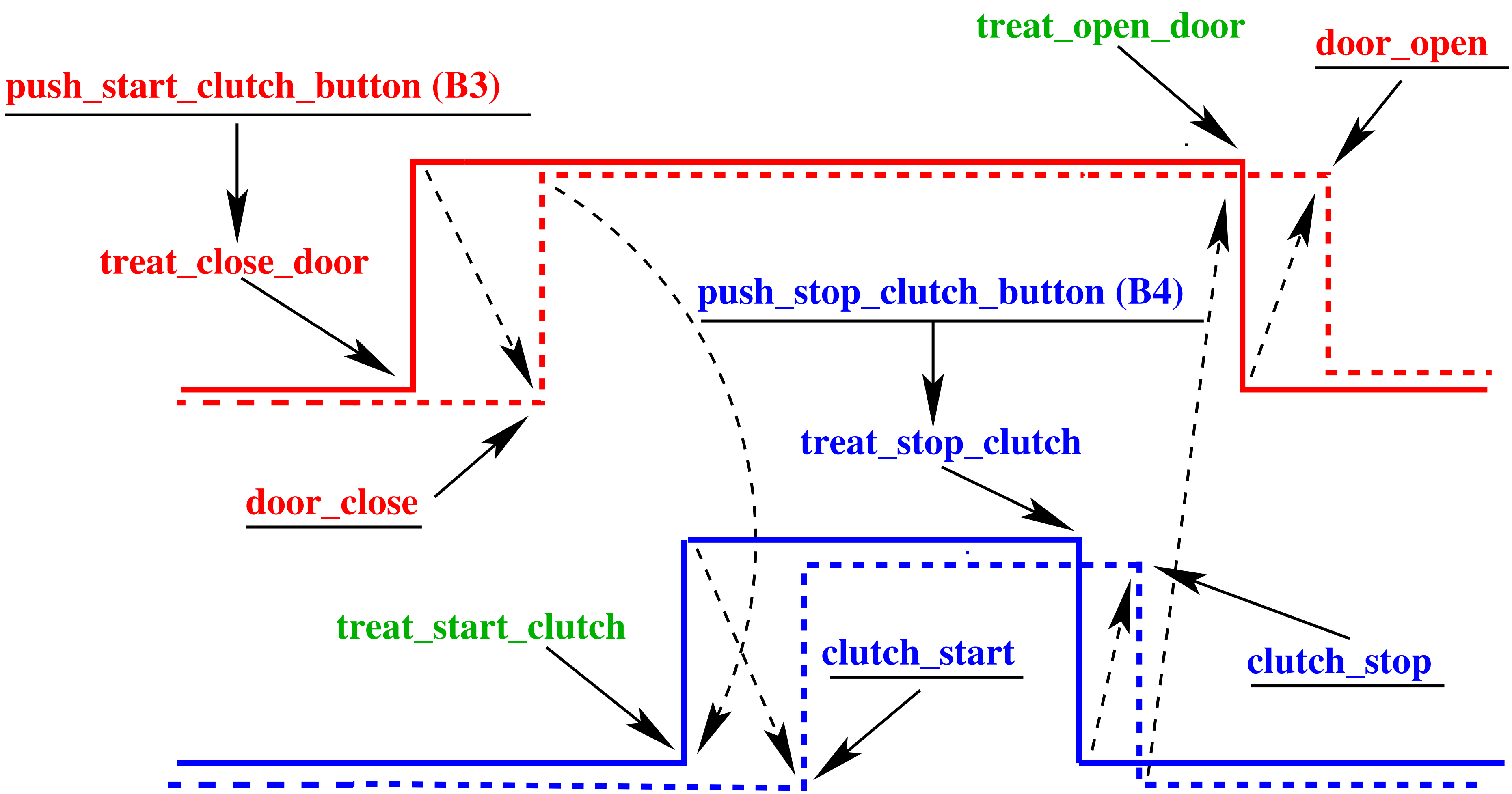
$b = 0$
$s = 0$
$r = 1$
$a = 1$
$m = 1$
then
$b := 1$
$m := 0$
end

treat_start_clutch
when
$motor\_actuator = working$
$motor\_sensor = working$
$clutch\_actuator = disengaged$
$clutch\_sensor = disengaged$
$door\_sensor = closed$
$door\_actuator = closed$
$m = 1$
then
$clutch\_actuator := engaged$
$m := 0$
end

a_off
when
$a = 1$
$r = 1$
$s = 0$
$b = 0$
$m = 0$
then
$a := 0$
end

treat_open_door
when
$door\_actuator = closed$
$door\_sensor = closed$
$clutch\_sensor = disengaged$
$clutch\_actuator = disengaged$
$m = 0$
then
$door\_actuator := open$
end

The Complete Synchronization of Door and Clutch



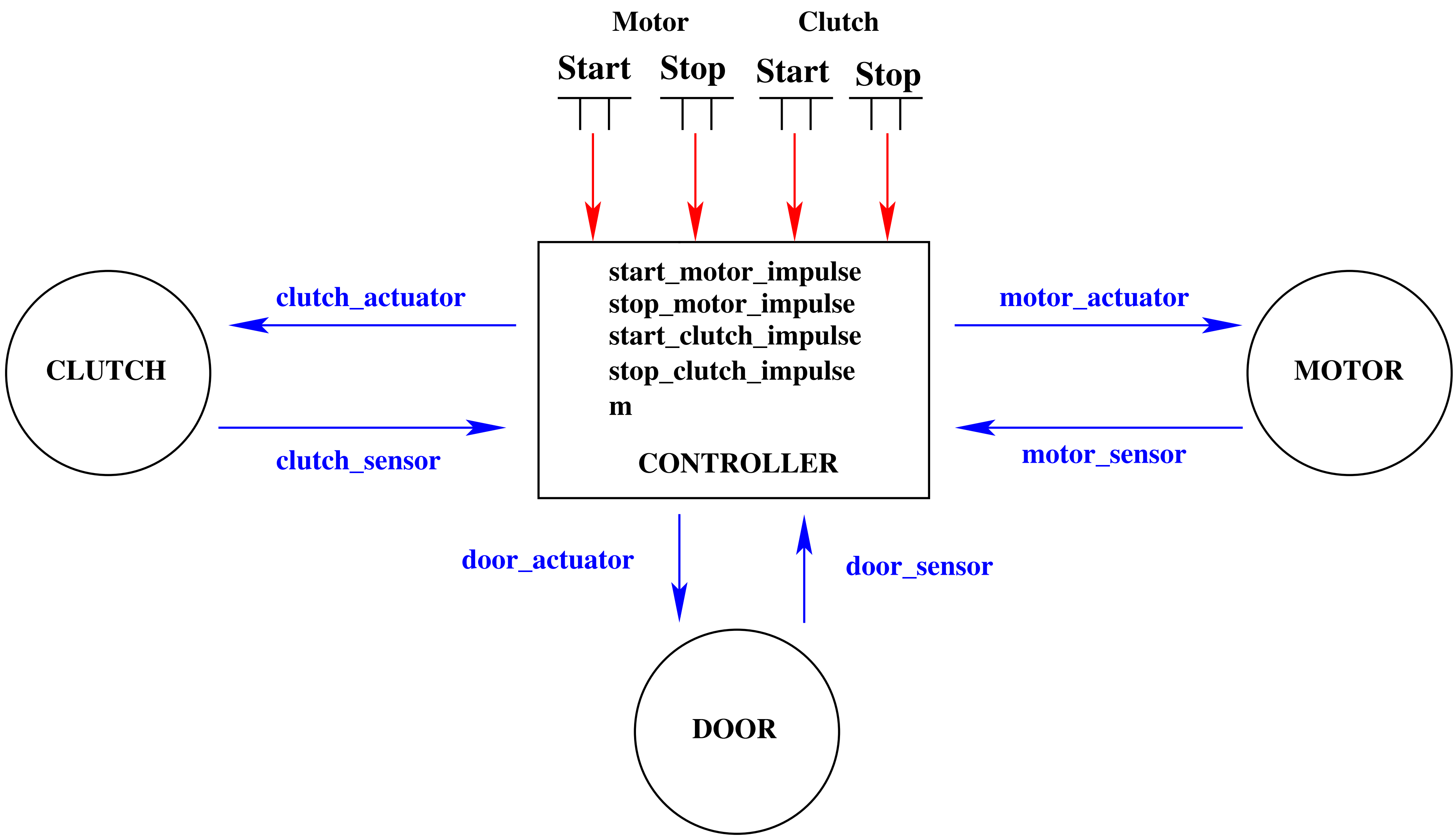
- treat_close_door is the result of depressing button B3
- treat_stop_clutch is the result of depressing button B4
- treat_start_clutch and treat_open_door are automatic

- Environment (no new events)
    - motor_start
    - motor_stop
    - clutch_start
    - clutch_stop
    - door_close
    - door_open
    - push_start_motor_button
    - release_start_motor_button
    - push_stop_motor_button
    - release_stop_motor_button

Sixth Refinement: Summary of the Events (2)

- Controller (no new events)
  - treat_push_start_motor_button
  - treat_push_start_motor_button_false
  - treat_push_stop_motor_button
  - treat_push_stop_motor_button_false
  - treat_release_start_motor_button
  - treat_release_stop_motor_button
  - treat_start_clutch
  - treat_stop_clutch
  - treat_close_door
  - treat_open_door

7th Reft.: Connecting the Controller to the Clutch Buttons



Reminder: Synchronizing Door and Clutch

- There are no door buttons

- The door must be closed before engaging the clutch

- The door must be opened after disengaging the clutch

- It is sufficient to connect:
    - button B3 to the door (closing the door)
    - button B4 to the clutch (disengaging the clutch)

- motor_start
- motor_stop
- clutch_start
- clutch_stop
- door_close
- door_open
- push_start_motor_button
- release_start_motor_button
- push_stop_motor_button
- release_stop_motor_button
- push_start_clutch_button
- release_start_clutch_button
- push_stop_clutch_button
- release_stop_clutch_button

- treat_push_start_motor_button
- treat_push_start_motor_button_false
- treat_push_stop_motor_button
- treat_push_stop_motor_button_false
- treat_release_start_motor_button
- treat_release_stop_motor_button
- treat_start_clutch
- treat_stop_clutch
- treat_close_door
- treat_open_door
- treat_close_door_false
- treat_stop_clutch_false
- treat_release_start_clutch_button
- treat_release_stop_clutch_button

Decomposing the Final Model: Environment

- The environment events

- The environment variables modified by environment events

- The sensor variables modified by environment events

- The actuator variables read by environment events

- The controller variables not seen by environment events

- No environment variables in this model

Decomposing the Final Model: Controller

- The controller events

- The controller variables modified by controller events

- The sensor variables read by controller events

- The actuator variables modified by controller events

- The environment variables not seen by controller events

- No environment variables in this model

Summary: Variables of the Last Refinement (1)

- 7 sensor variables:
  - $motor\_sensor$
  - $clutch\_sensor$
  - $door\_sensor$
  - $start\_motor\_button$
  - $stop\_motor\_button$
  - $start\_clutch\_button$
  - $stop\_clutch\_button$

- 3 actuator variables:
  - $motor\_actuator$
  - $clutch\_actuator$
  - $door\_actuator$
- 5 controller variables (without the counter variables):
  - $start\_motor\_impulse$
  - $stop\_motor\_impulse$
  - $start\_clutch\_impulse$
  - $stop\_clutch\_impulse$
  - m

Summary: Events of the Last Refinement

- 14 environment events,

- 14 controller events,

- 130 lines for environment events,

- 180 lines for controller events.

Summary: Usage of the Design Patterns

- 4 weak reactions: 4 buttons (B1, B2, B3, B4)
- 3 strong reactions: 3 devices (motor, clutch, door)
- 3 strong-weak reactions: motor-clutch, clutch-door, motor-door
- 1 strong-strong reaction: clutch-door

Summary: Number of Invariants

- Weak reaction: 6
- Strong reaction: 3
- Strong-weak reaction: 16
- Strong-strong reaction: 7
- Total: 32

- Press (typing): 15
- Press (redundant with those of patterns): 12
- Total: 27

Summary: Number of Proof Obligations

- Weak reaction: 18
- Strong reaction: 12
- Strong-weak reaction: 60
- Strong-strong reaction: 40
- Total: 130

- Press (redundant with those of design patterns): 60

- PO saving: 4x18 + 3x12 + 3x60 + 40 = 328

Summary: Proofs

- Design patterns: 2 easy interactive, out of 130

- Press: all automatic, out of 60

- 600 lines of C code for the simulation,

- 470 lines come from a direct translation of the last refinement,

- 130 lines correspond to the hand-written interface.

Conclusion

- This design pattern approach seems to be fruitful

- It results in a very systematic formal development

- Many other patterns have to be developed

- More automation has to be provided (plug-in)