

3. A Mechanical Press Controller

Jean-Raymond Abrial

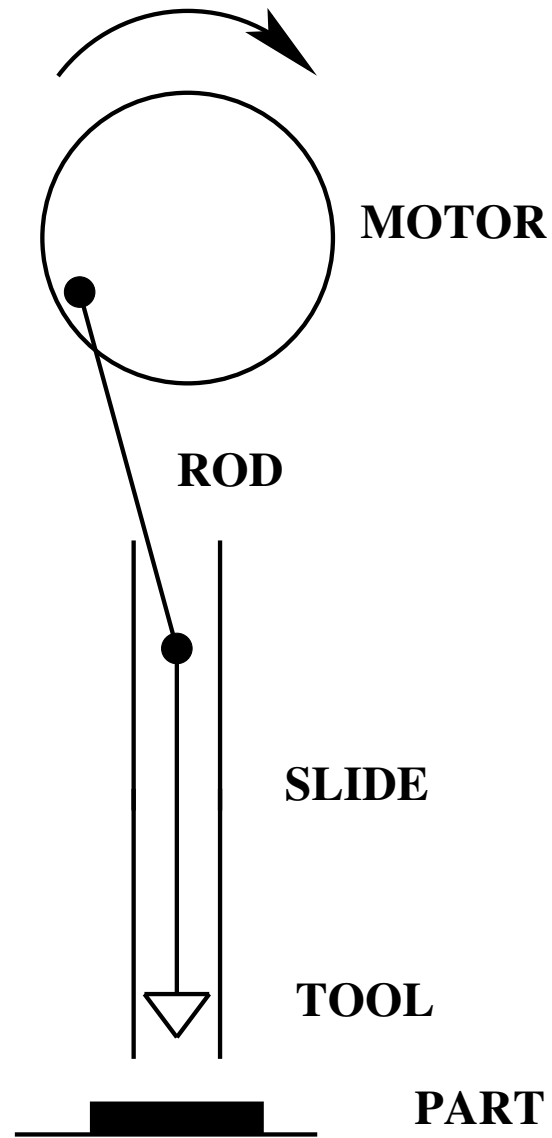
2009

-
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

- A mechanical **press controller**
- **Adapted** from a **real system**
- The real system is coming from **INRST**:

Institut **N**ational de la **R**echerche sur la **S**écurité du **T**ravail

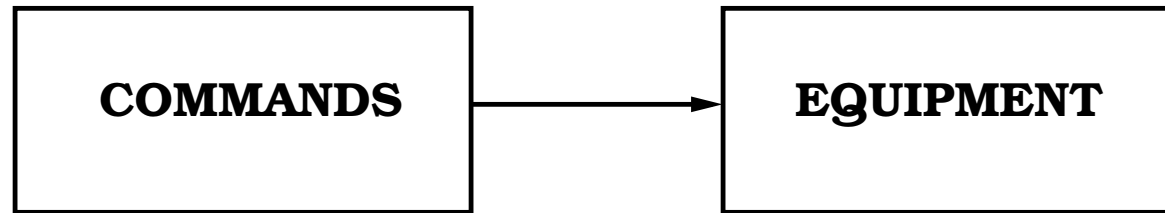


- B1
 - B2
 - B3
 - B4
- BUTTONS**

- 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”**

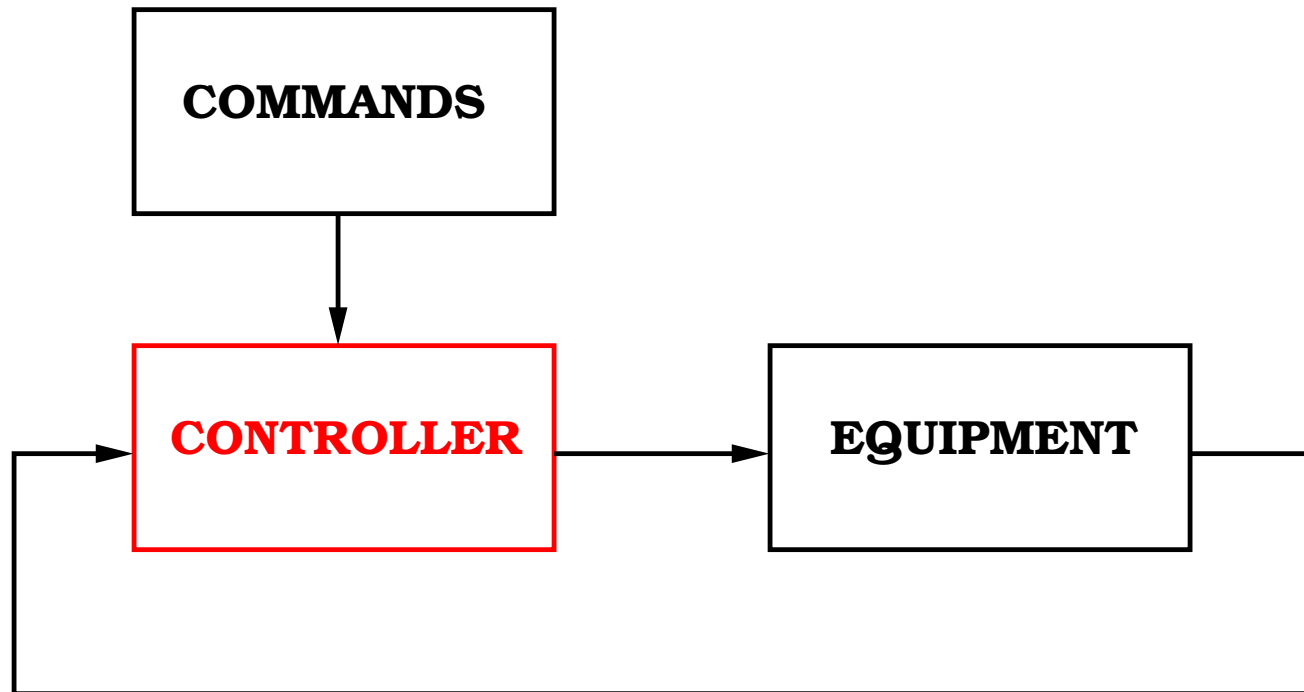
- Button B1: start motor
- Button B2: stop motor
- Button B3: engage clutch
- Button B4: disengage clutch

- 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**

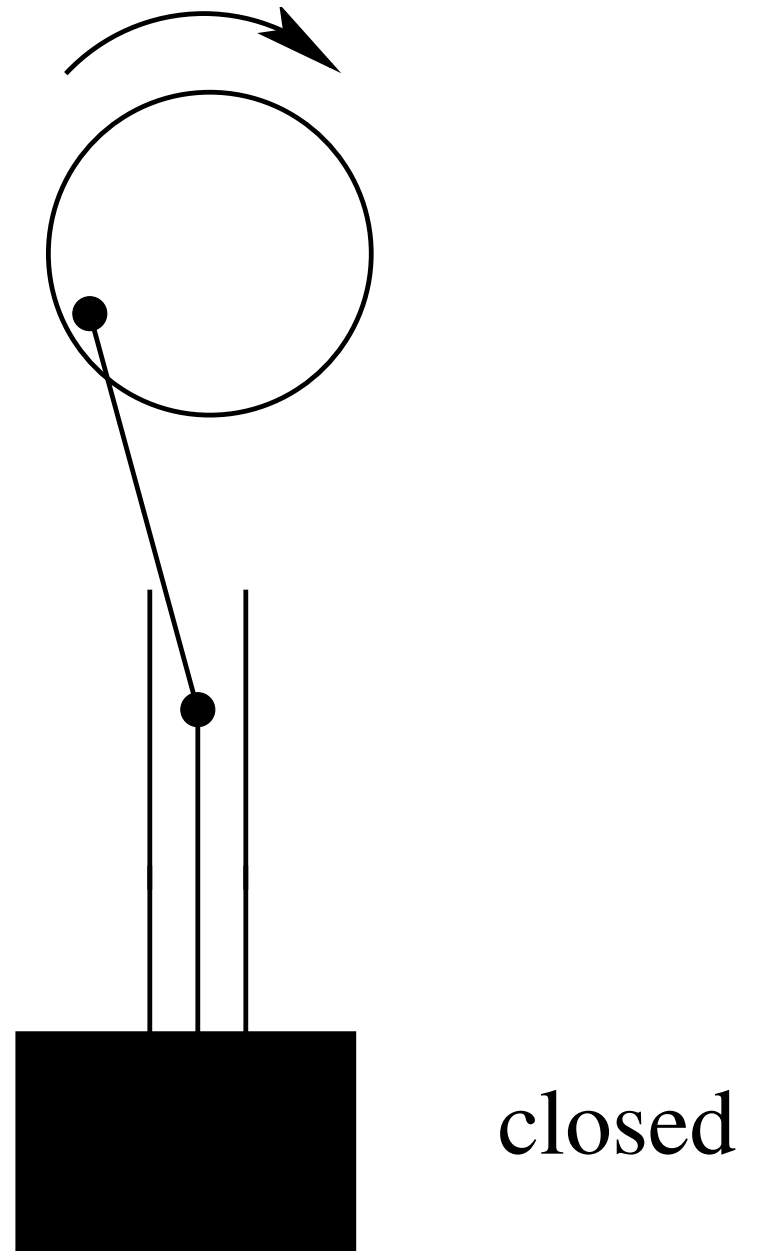
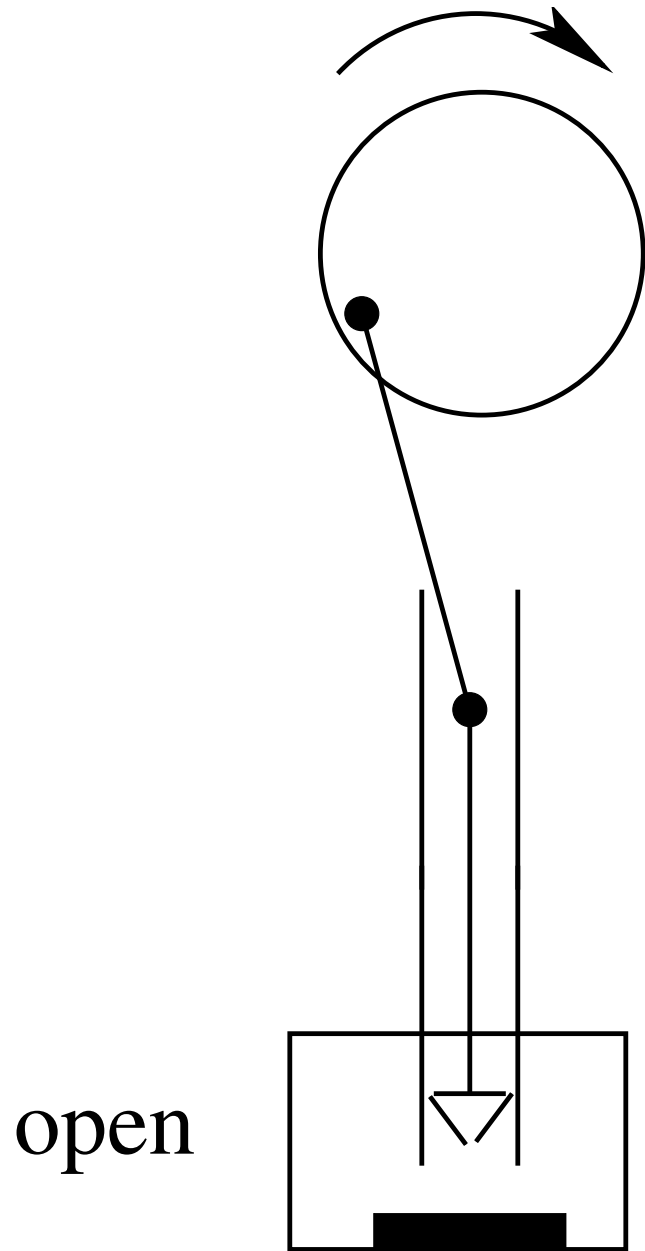


1. **start the motor** (button B1)
2. **change the tool** (action 1)
3. **put a part** (action 2),
4. **engage the clutch** (button B3): the press now **works**,
5. **disengage the clutch** (button B4): the press **does not work**,
6. **remove the part** (action 3),
7. **repeat** zero or more times steps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

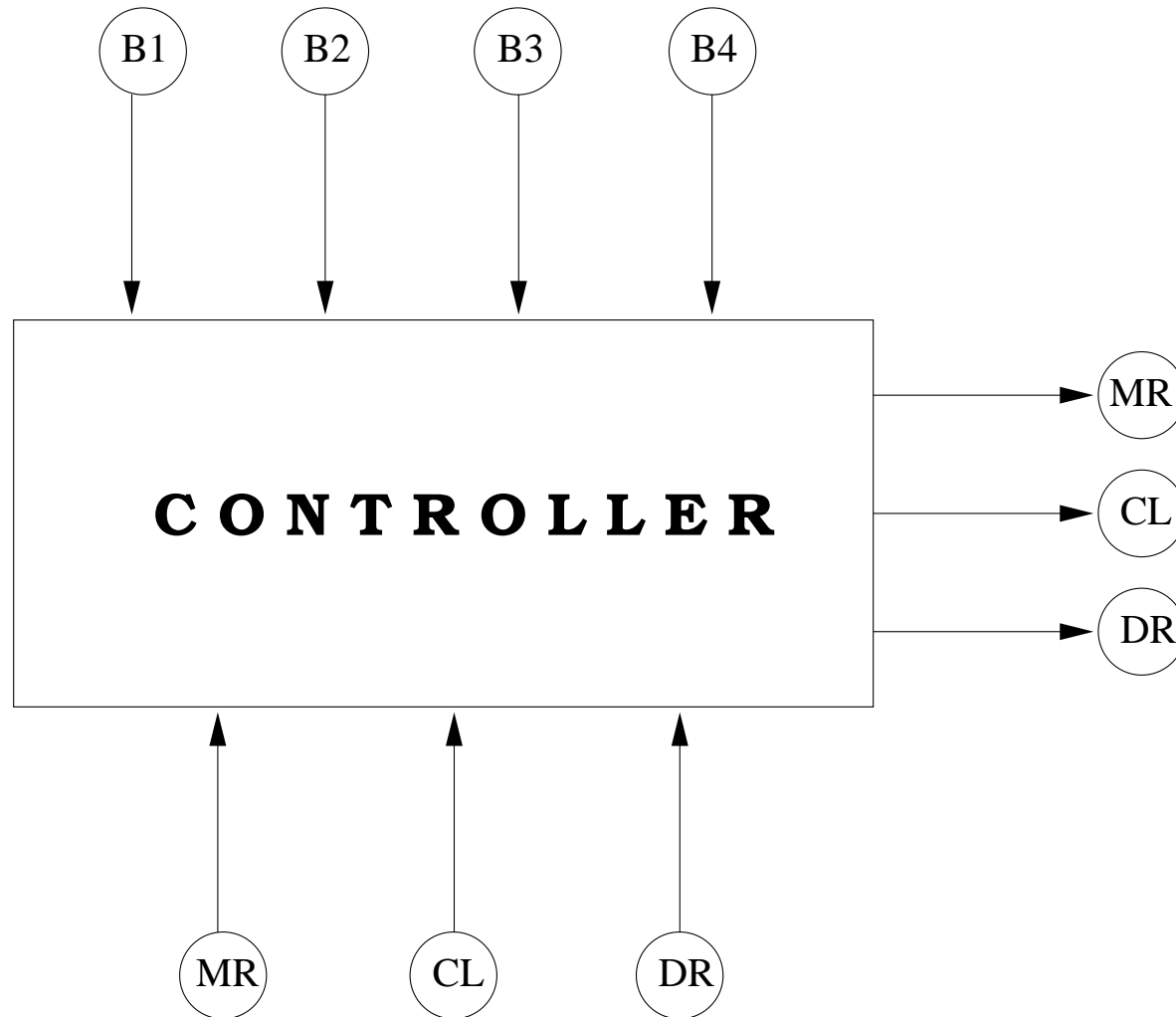
- step 2 (change the tool),
- step 3 (put a part),
- step 6 (remove the part) are all **DANGEROUS**

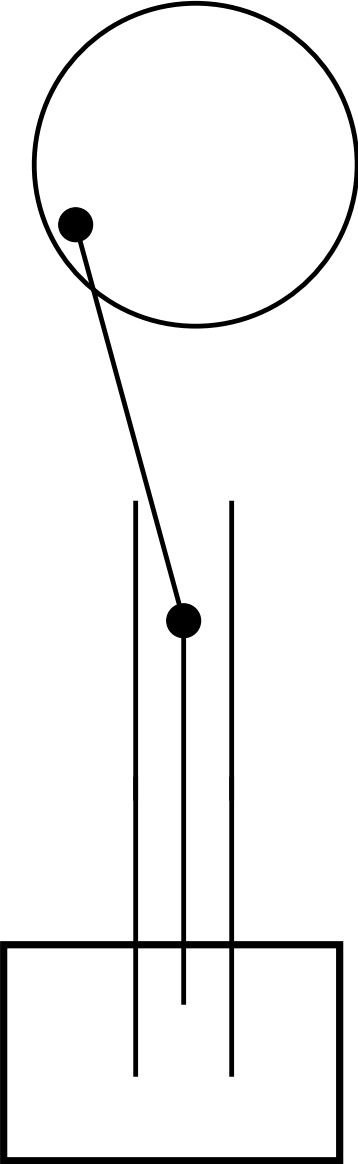


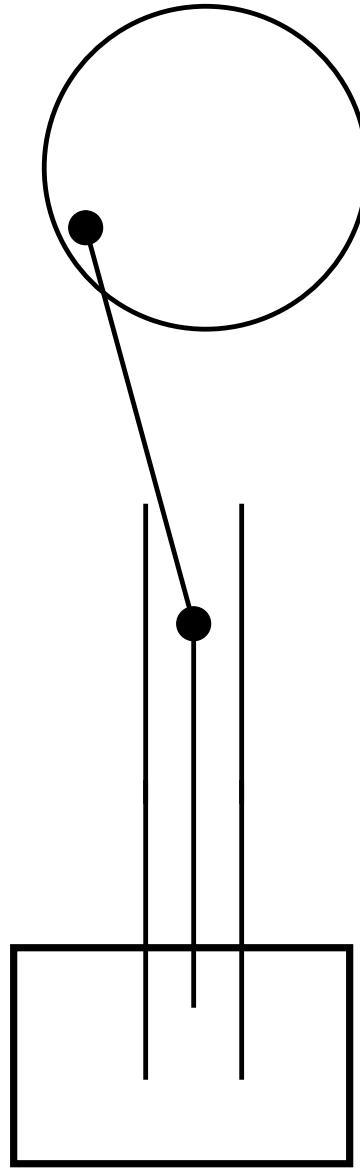
- Controlling the way the clutch is engaged or disengaged
- Protection by means of a Front Door

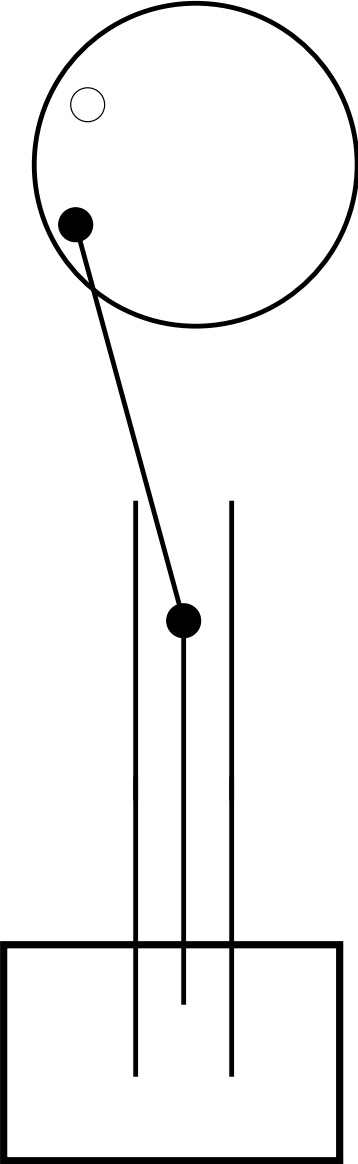


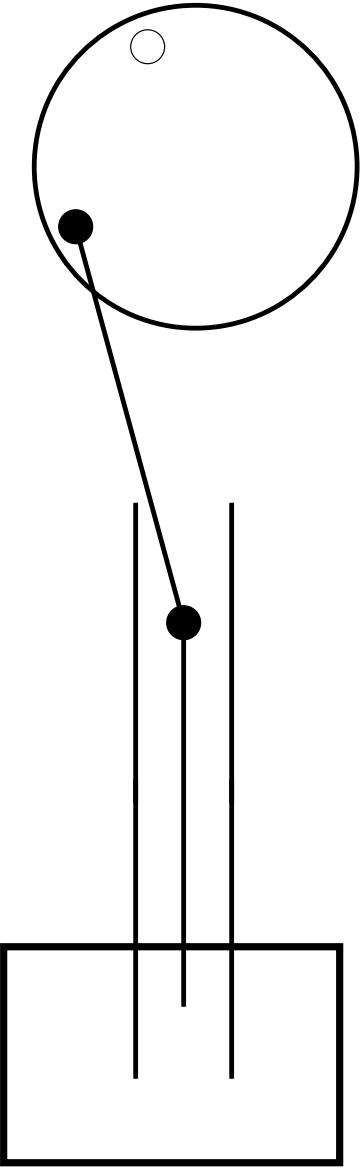
- Initially, the door is open
- When the user presses button B3 to engage the clutch, the door is first closed BEFORE engaging the clutch
- When the user presses button B4 to disengage the clutch, the door is opened AFTER disengaging the clutch
- Notice: The door has no button.

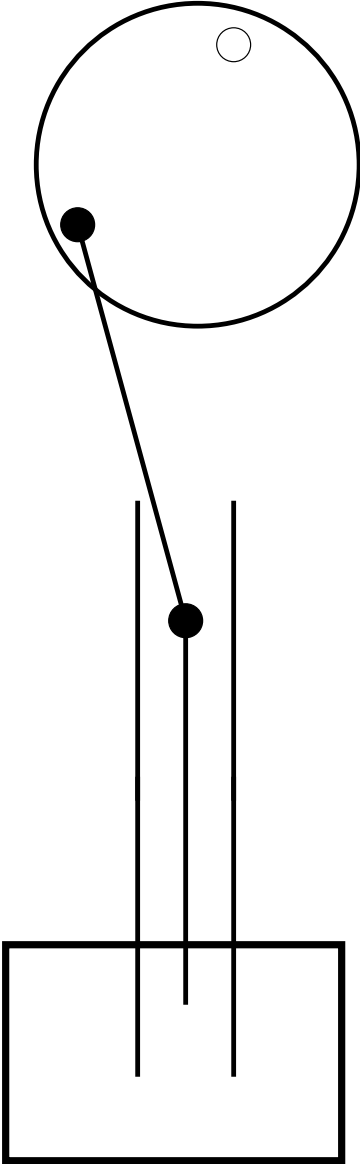


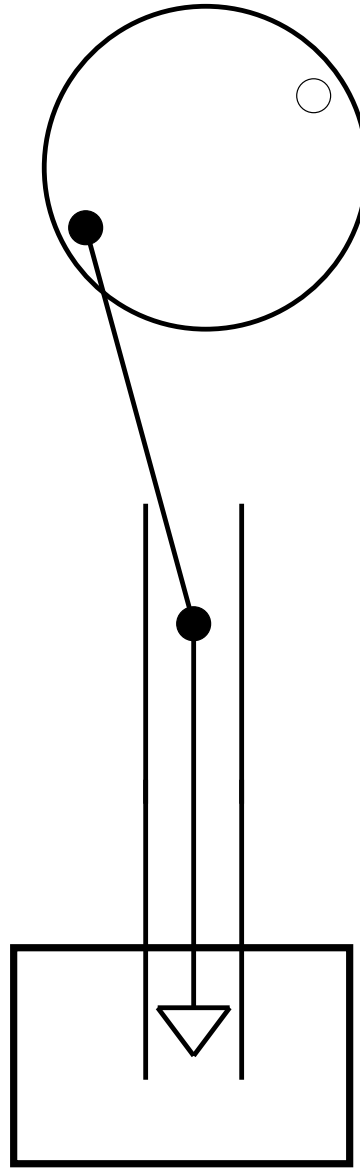


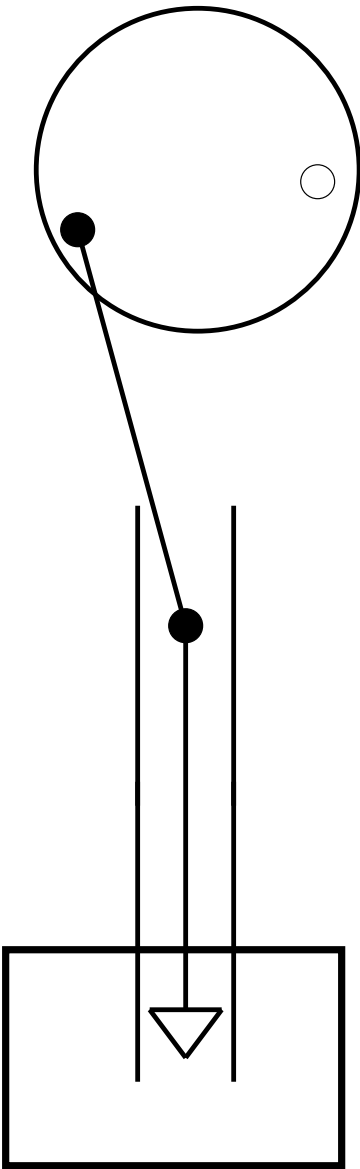


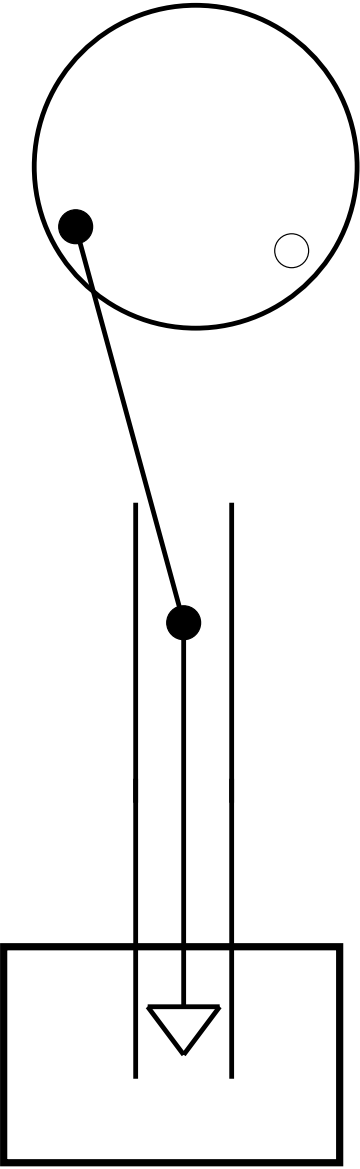


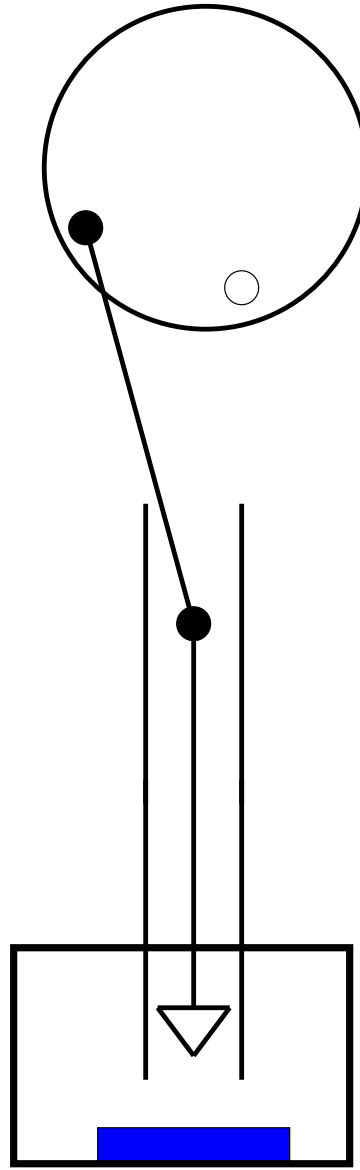


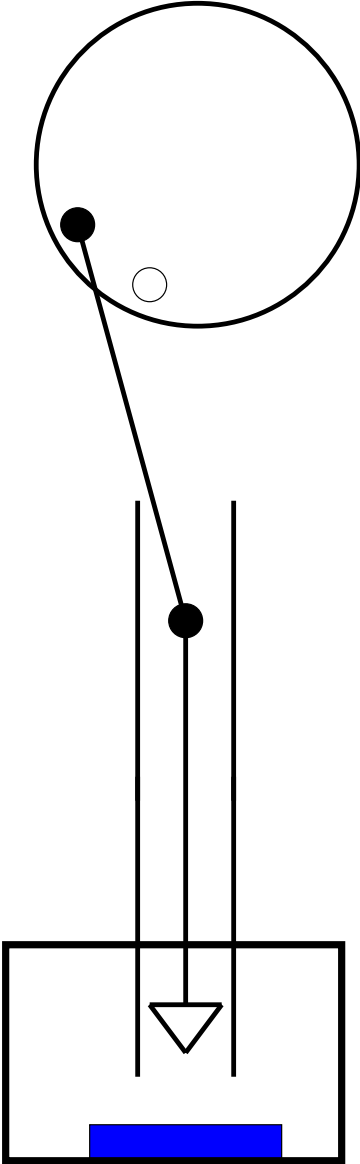


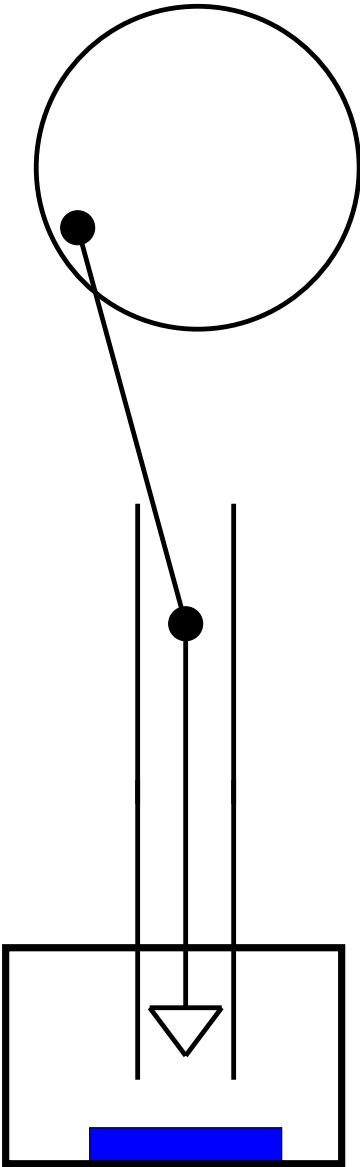


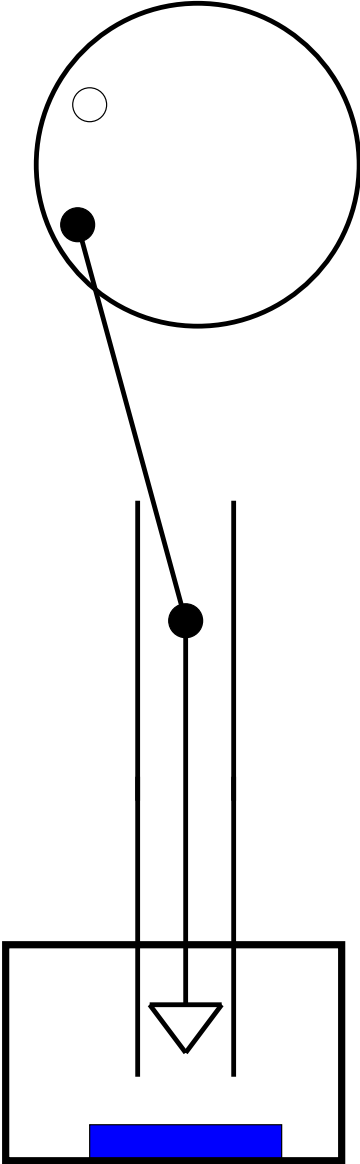


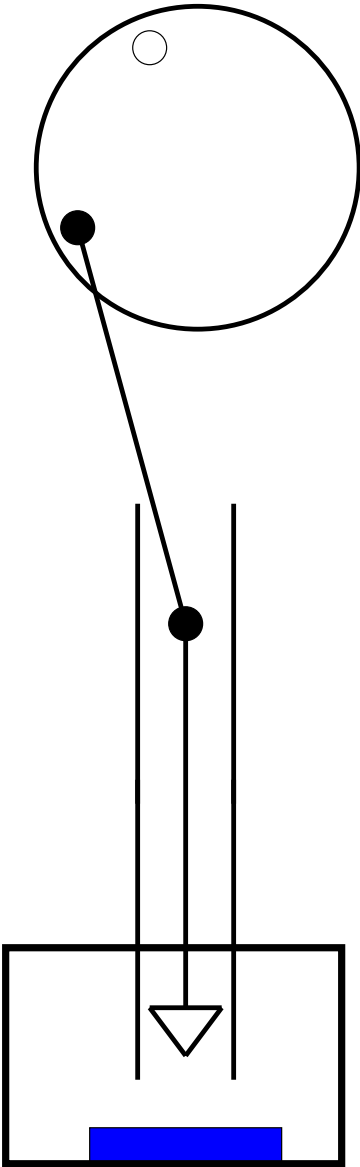


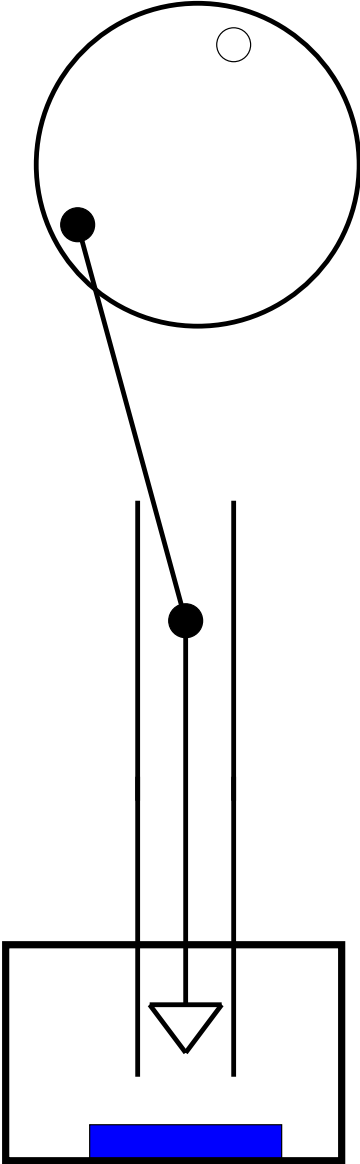


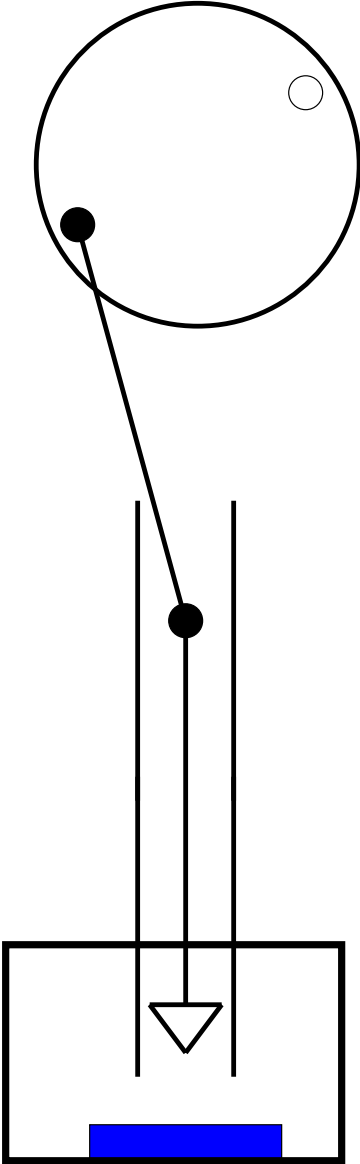


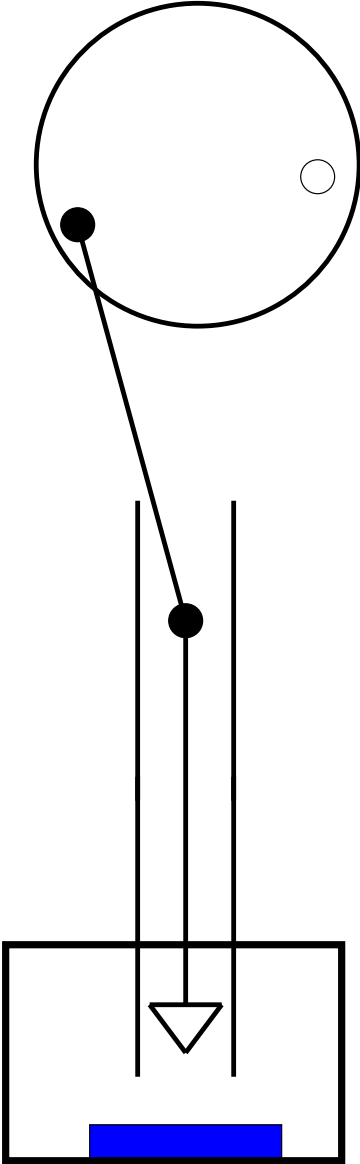


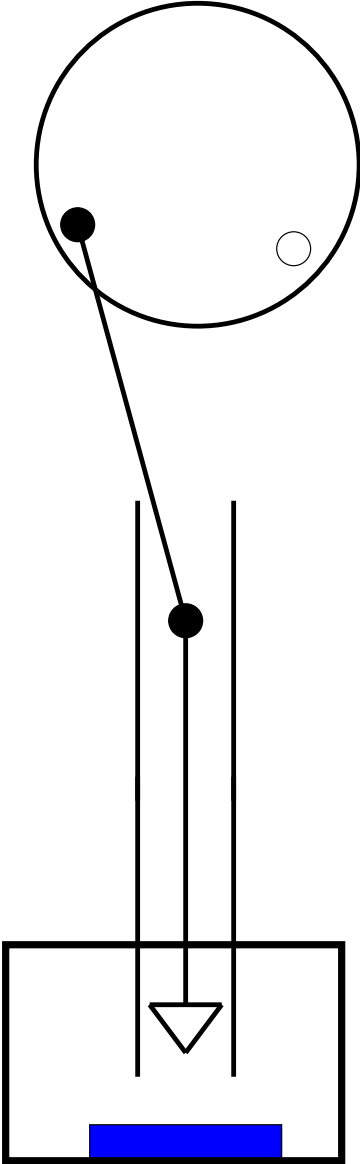


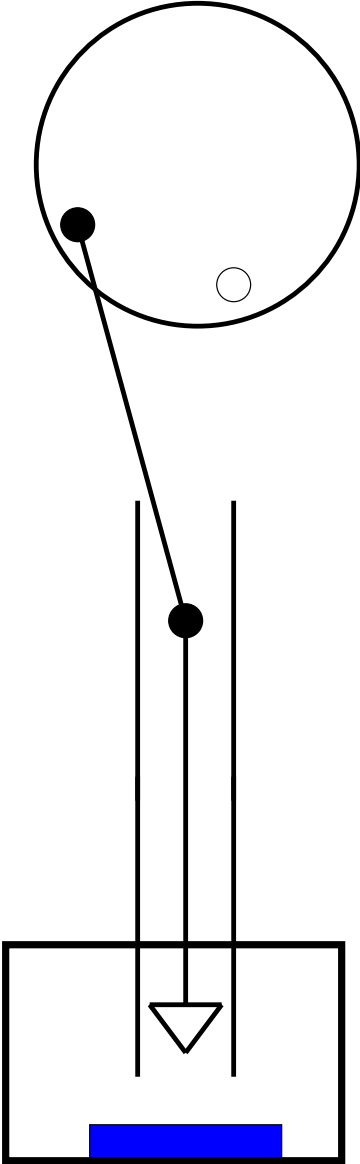


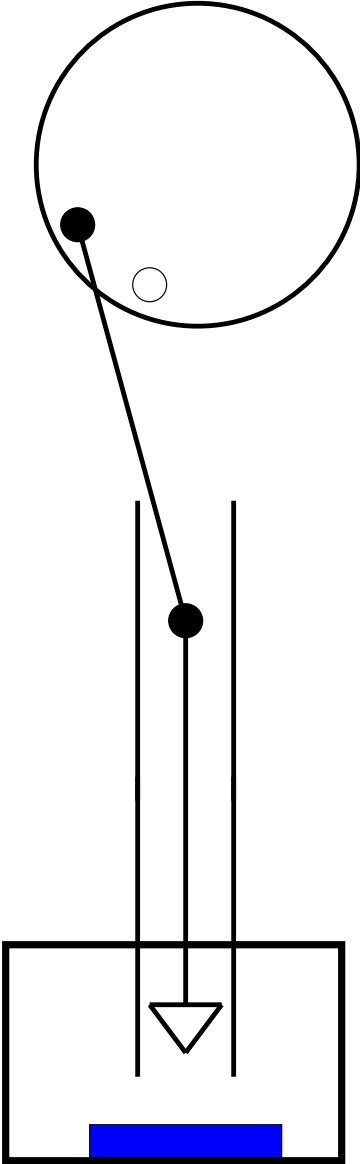


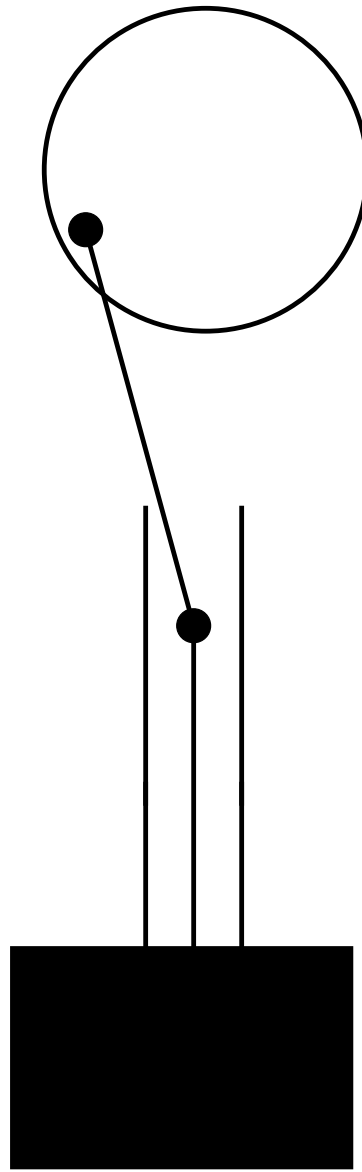


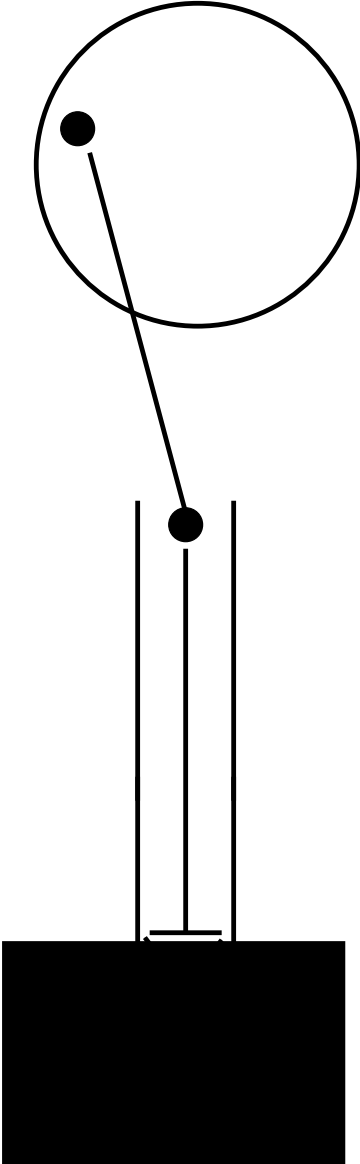


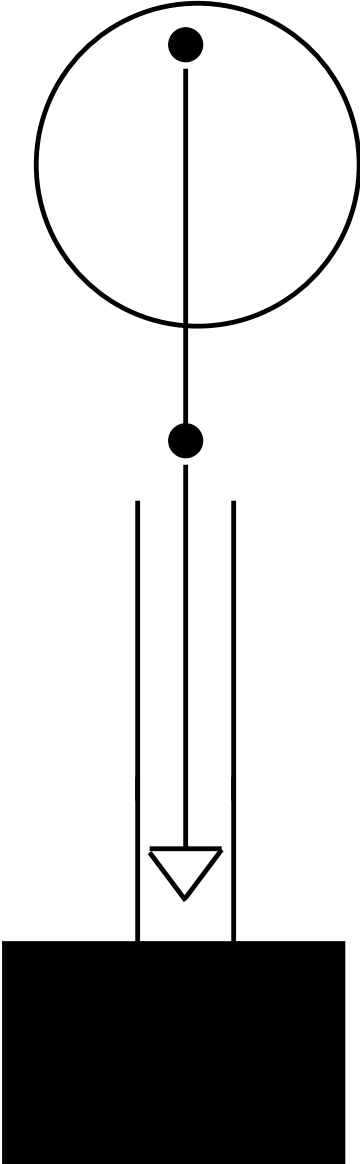


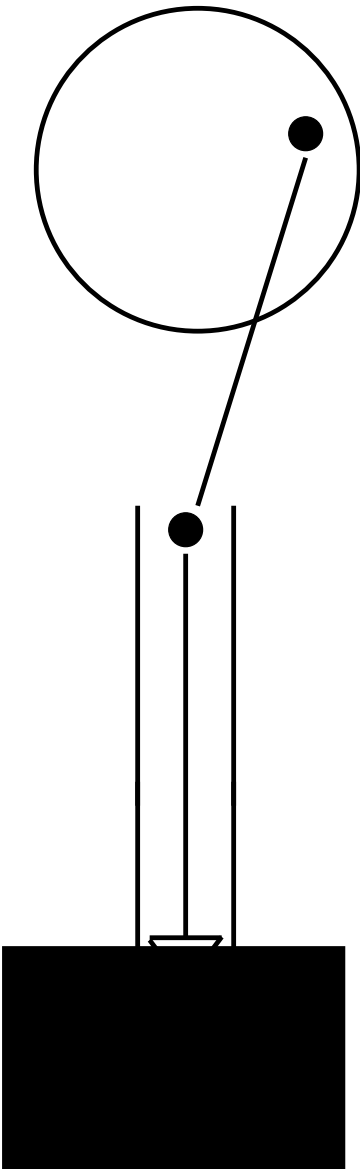


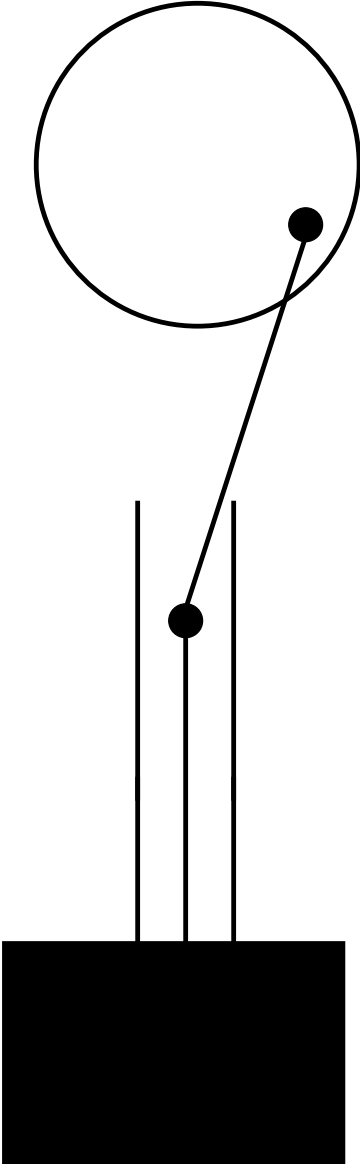


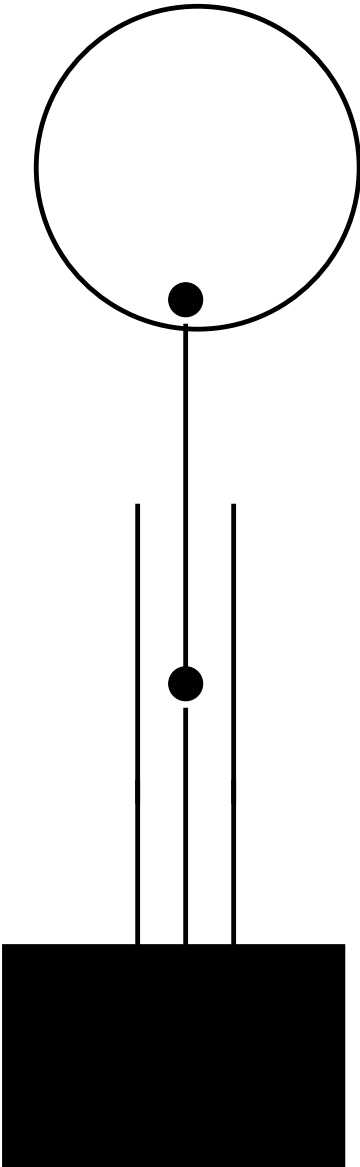


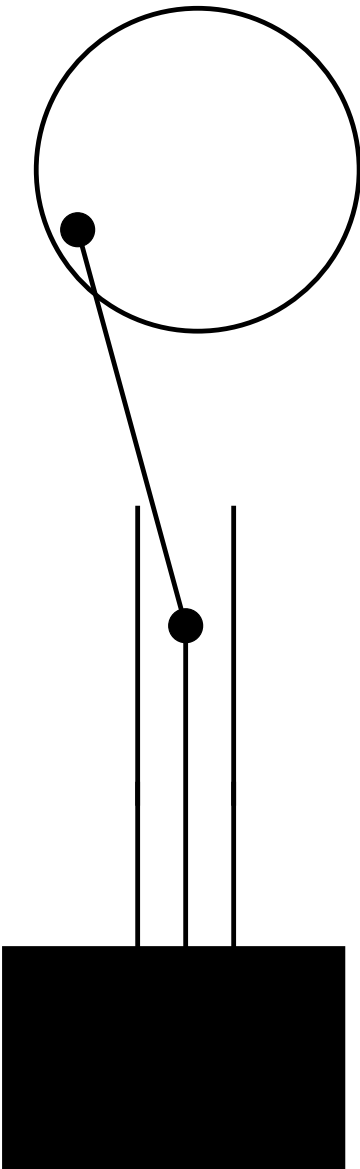


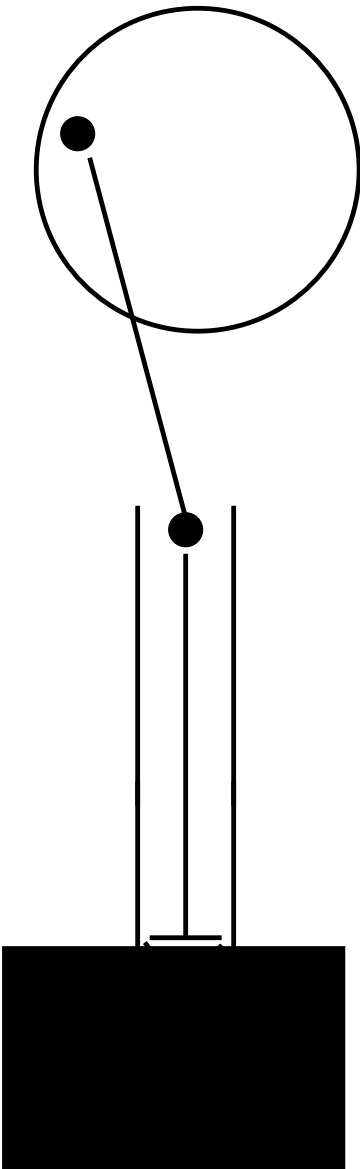


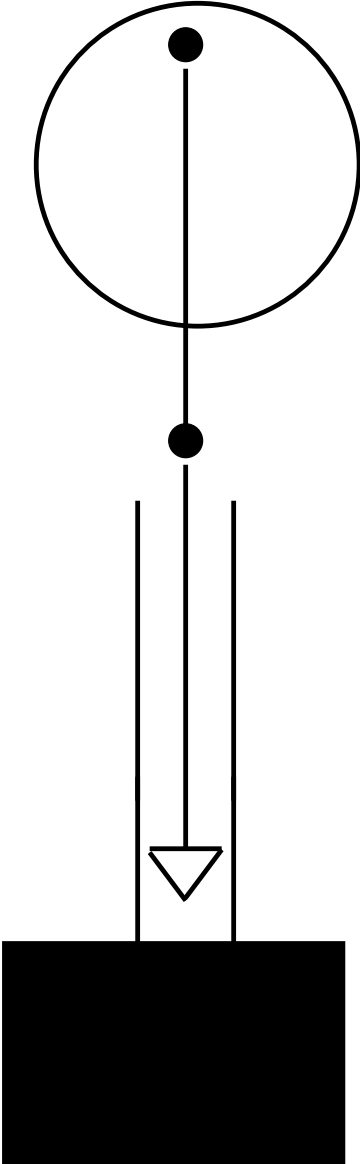


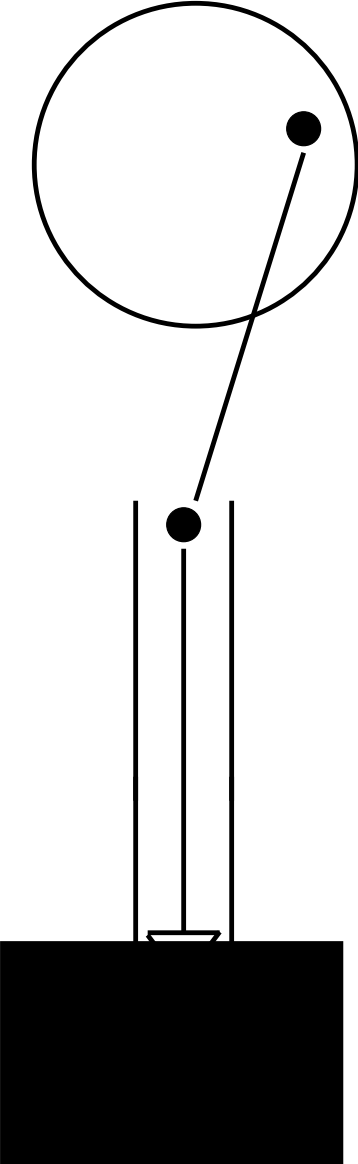


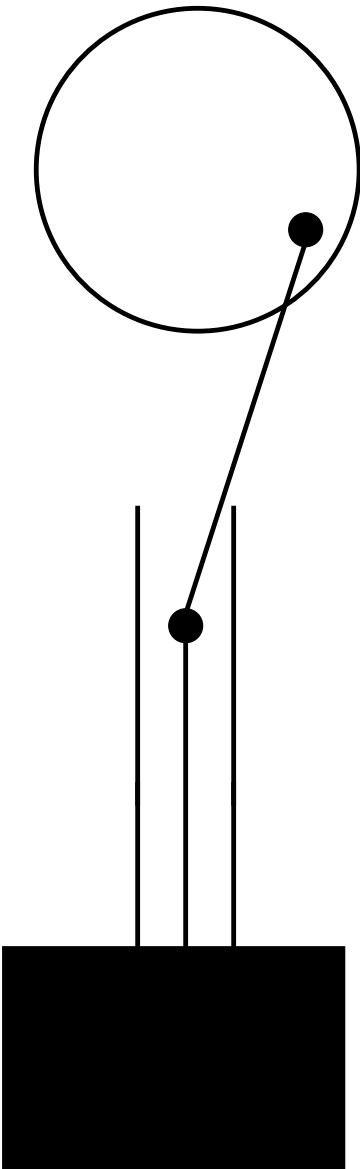


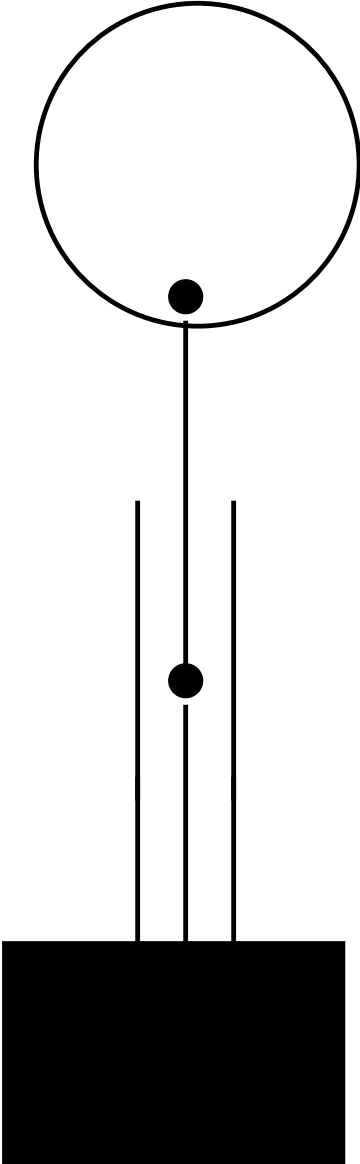


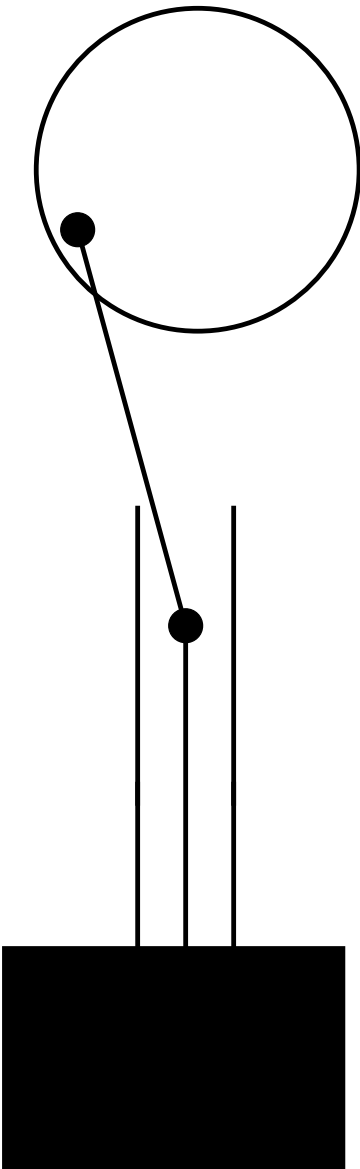


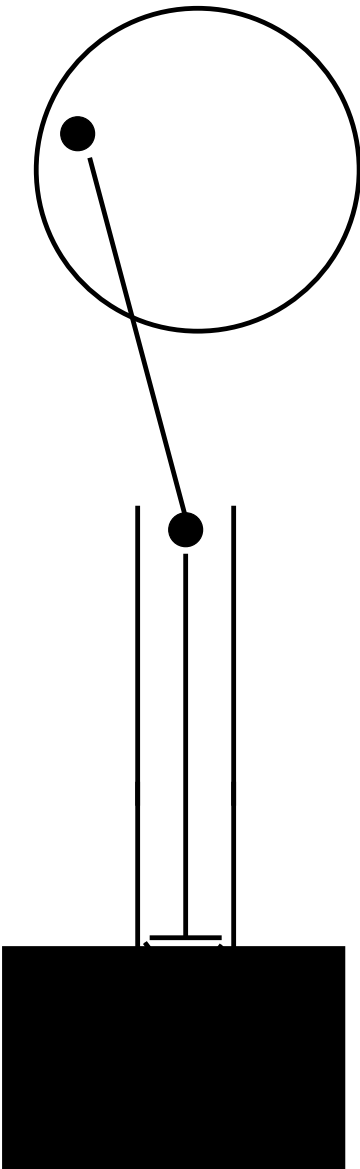


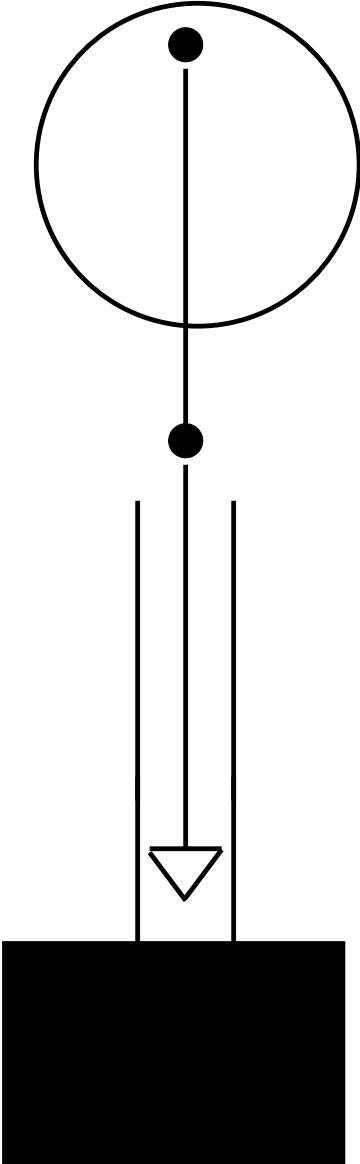


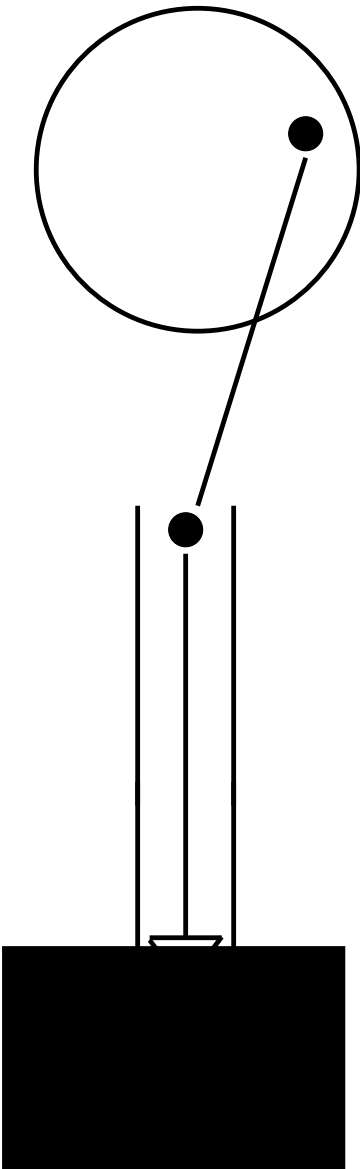


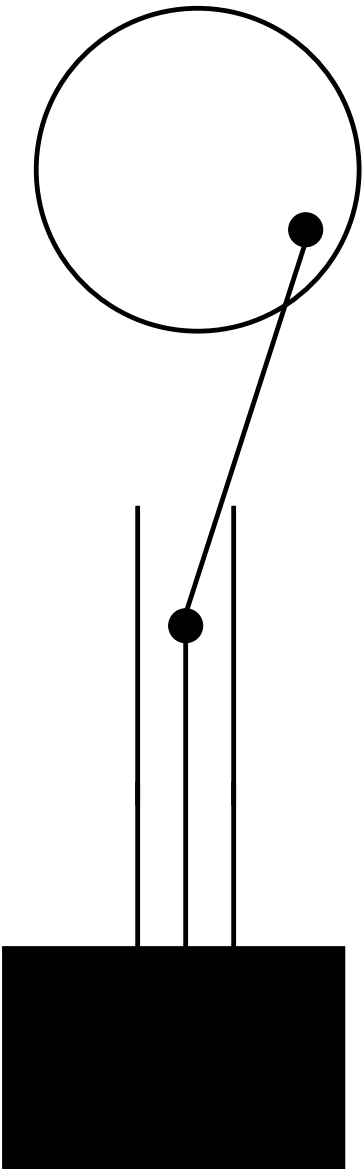


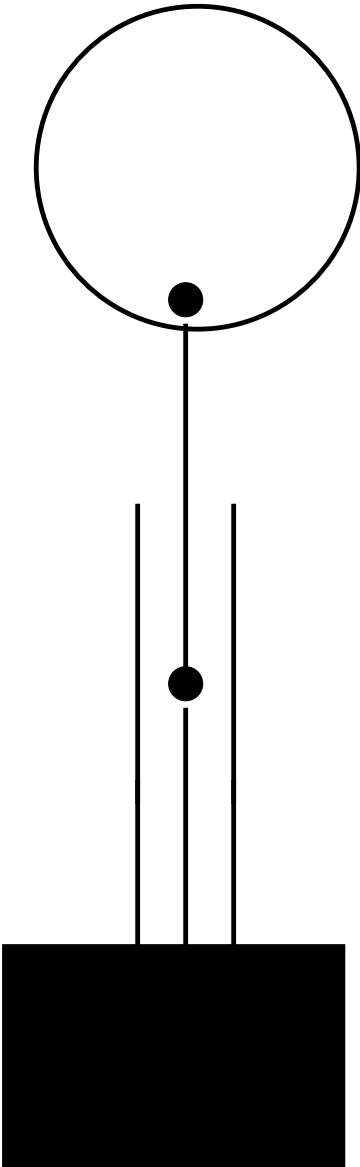


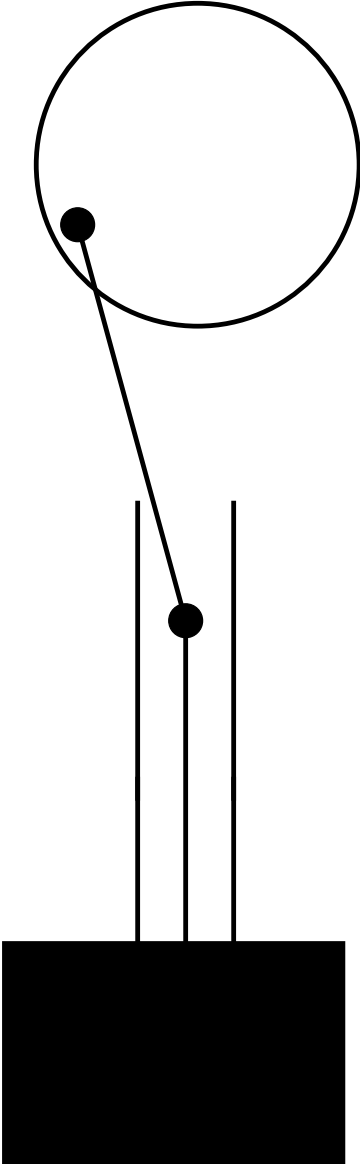


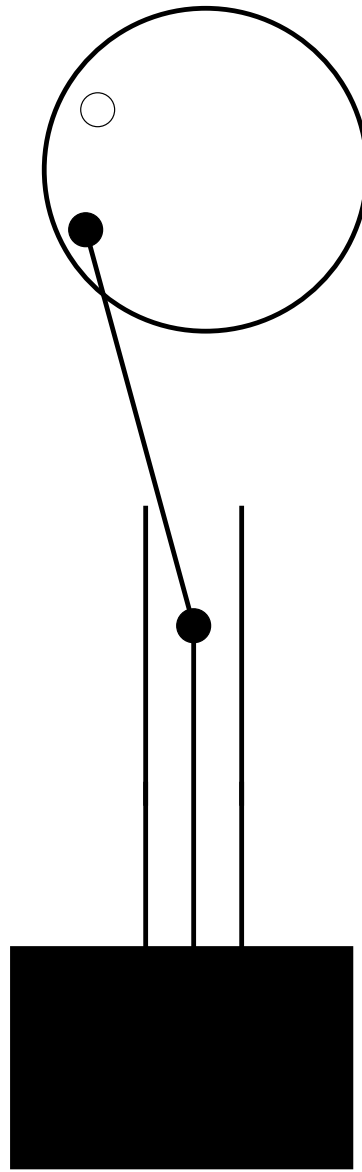


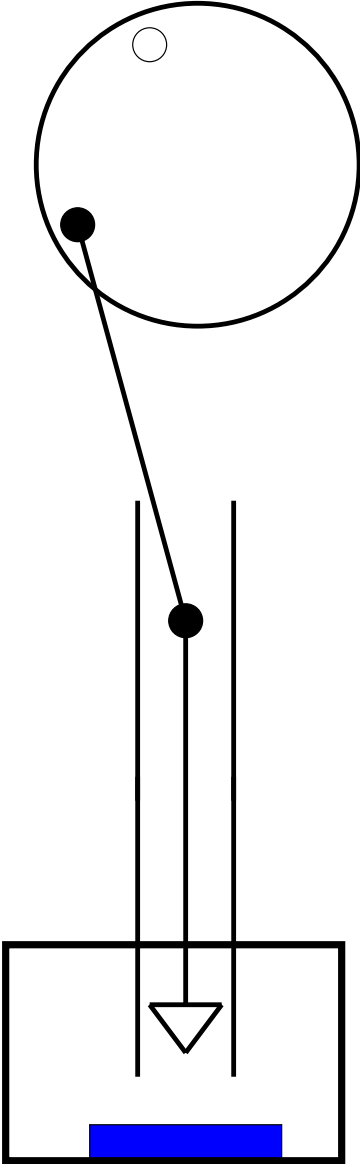


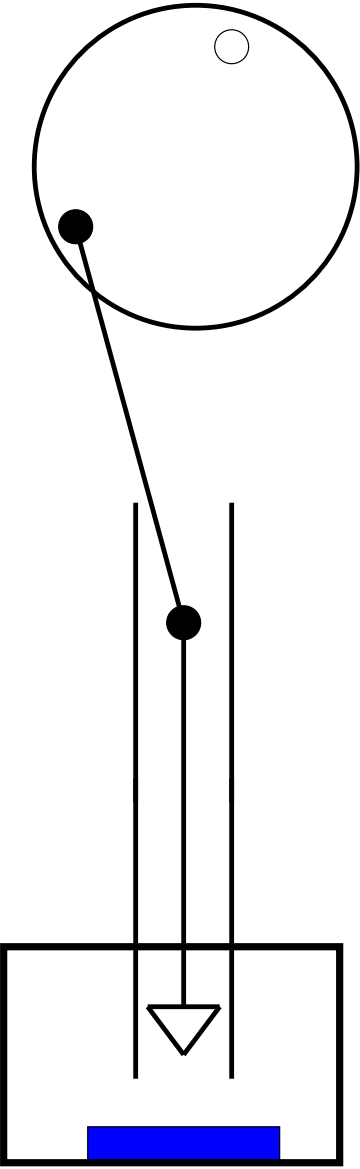


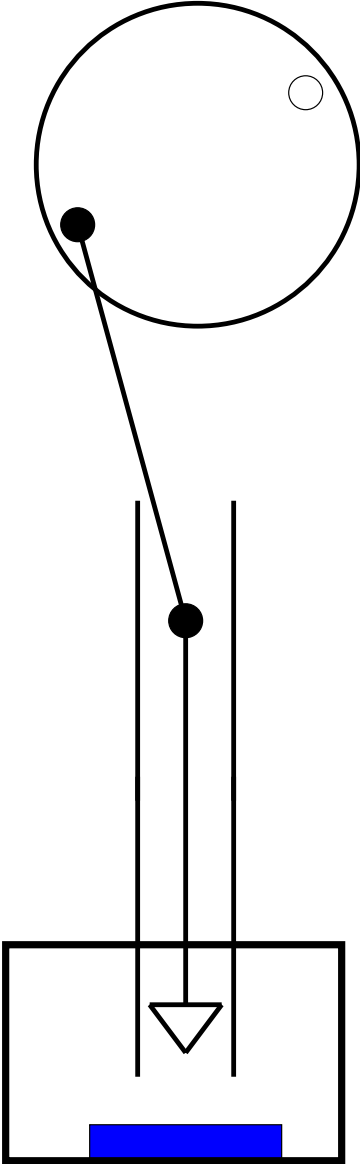


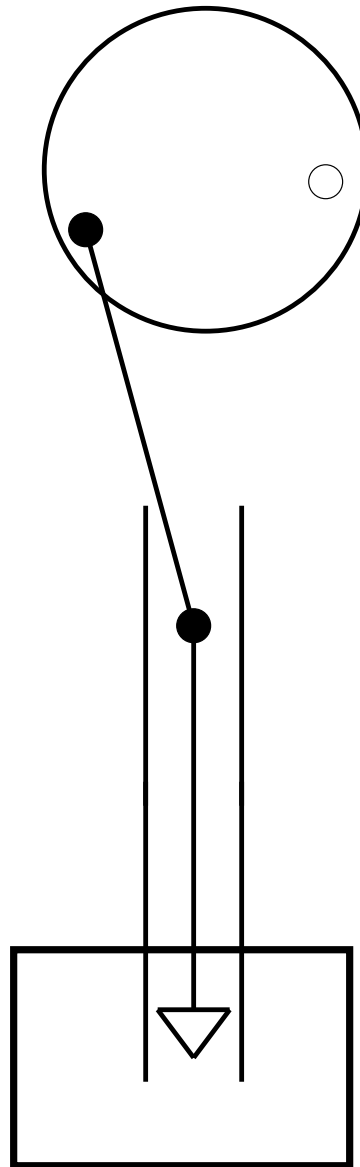


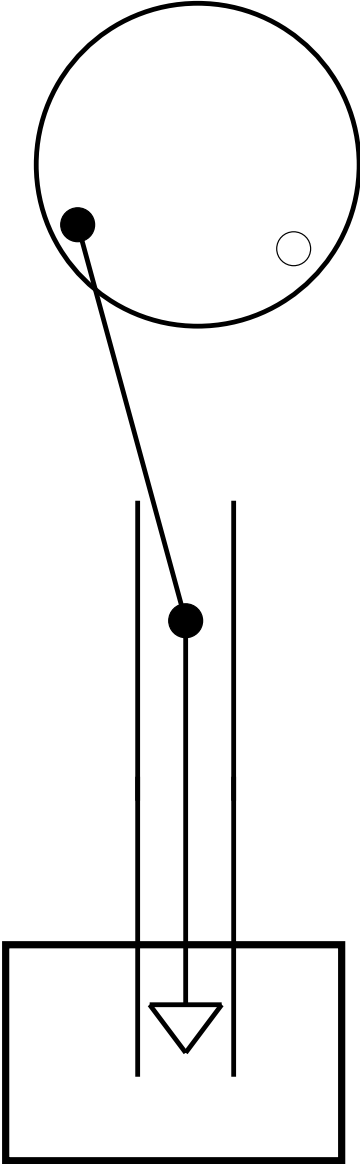


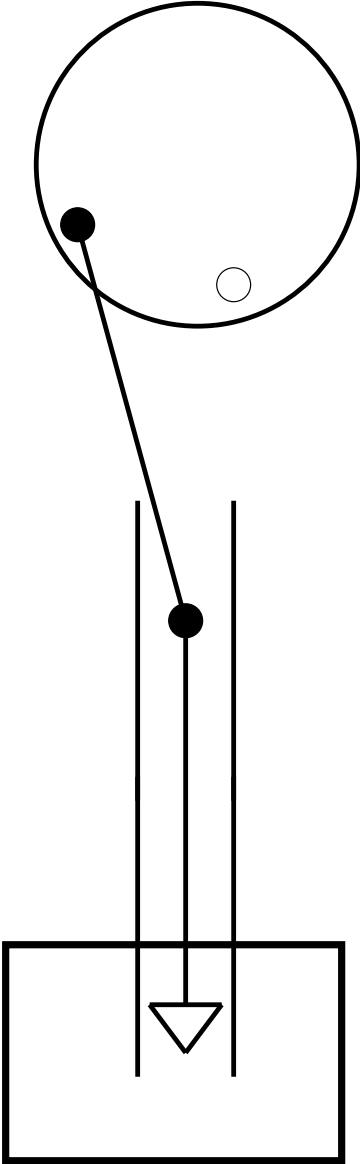


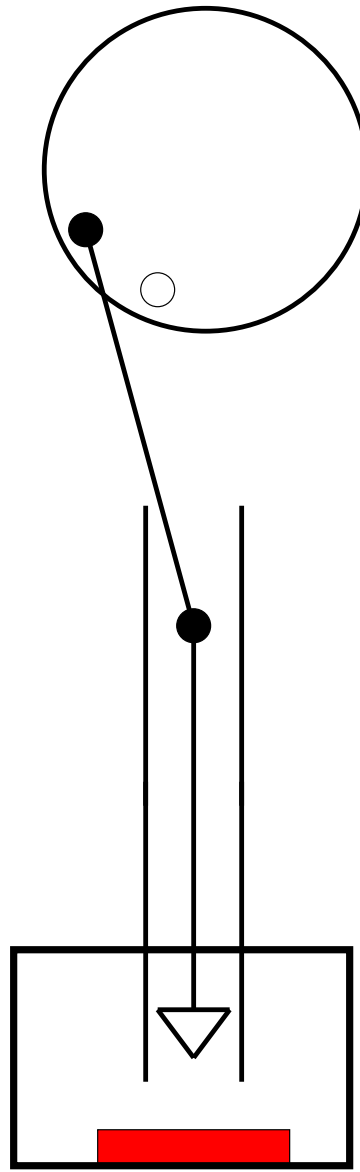


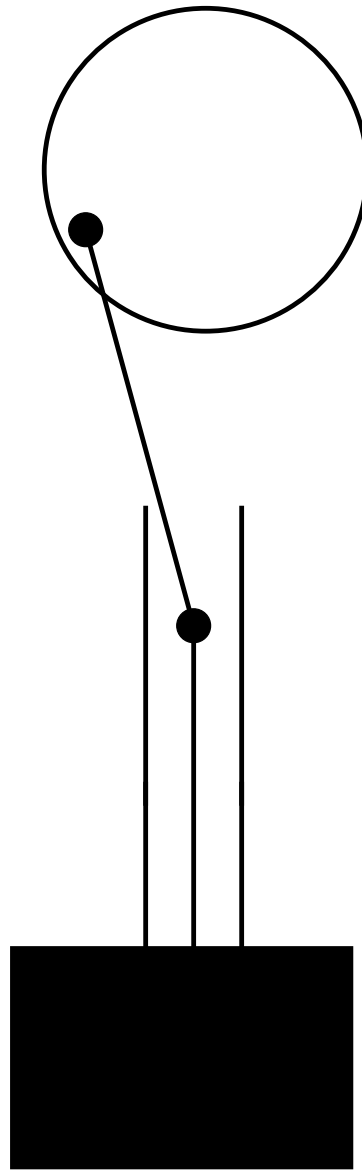


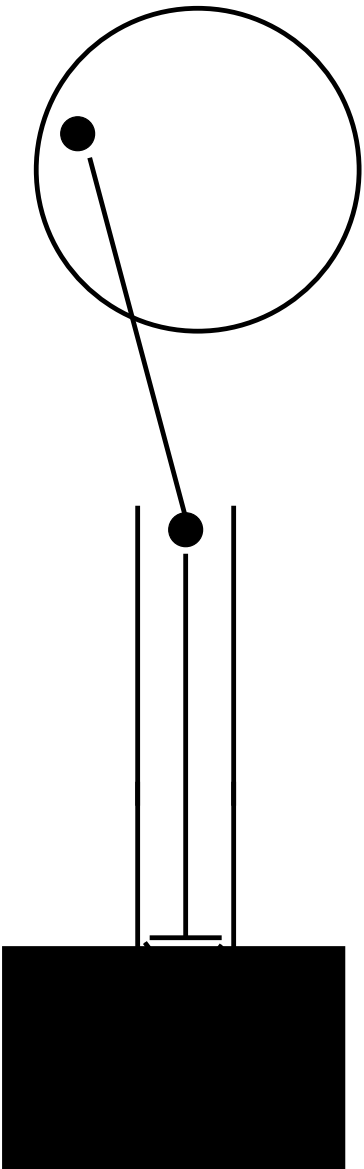


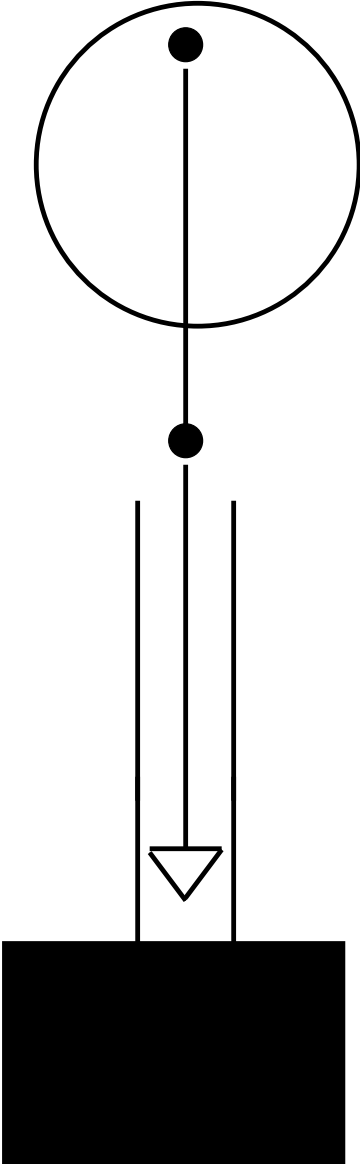


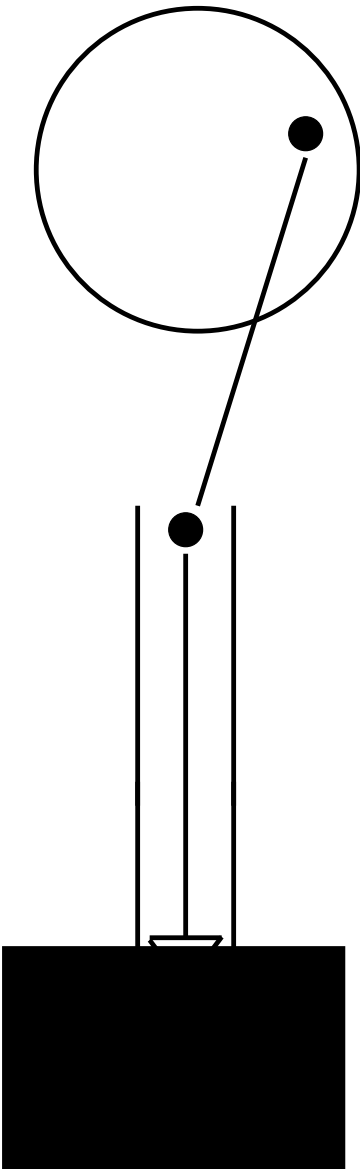


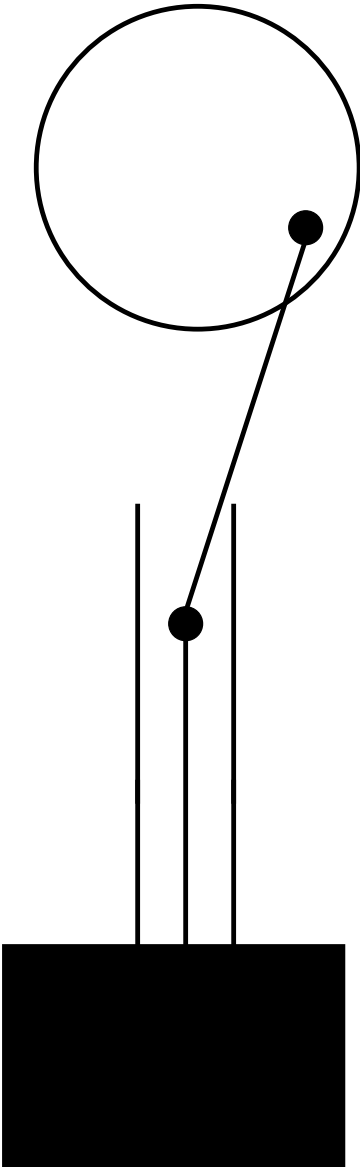


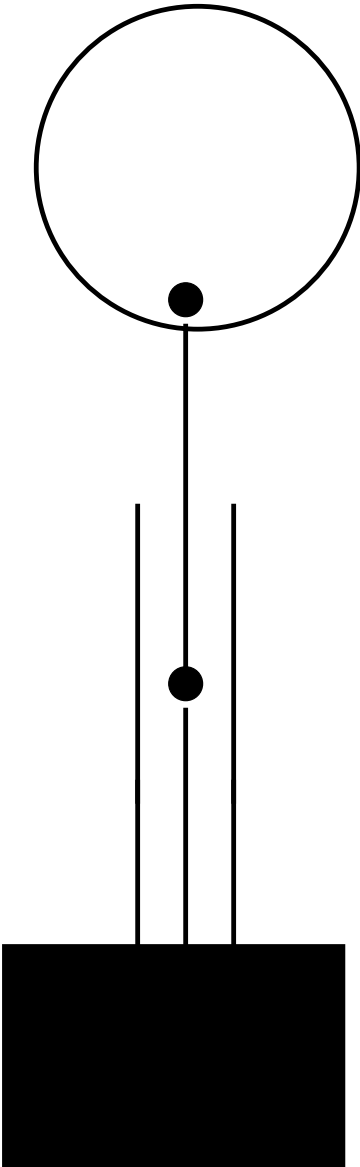


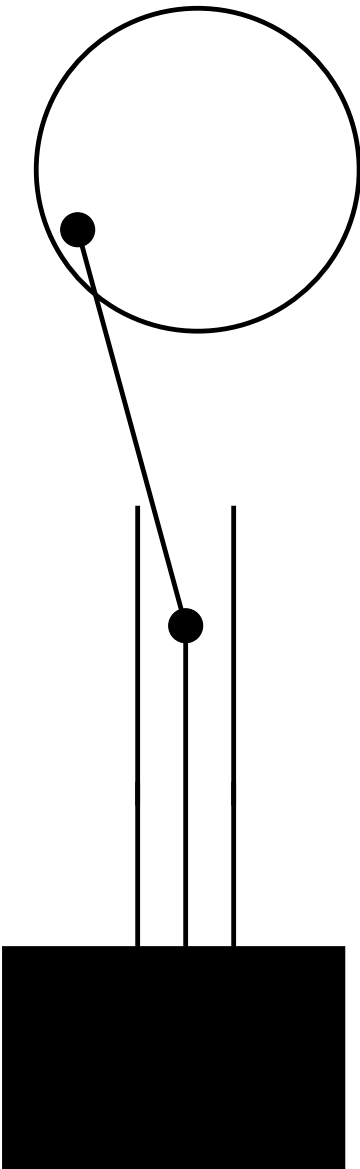


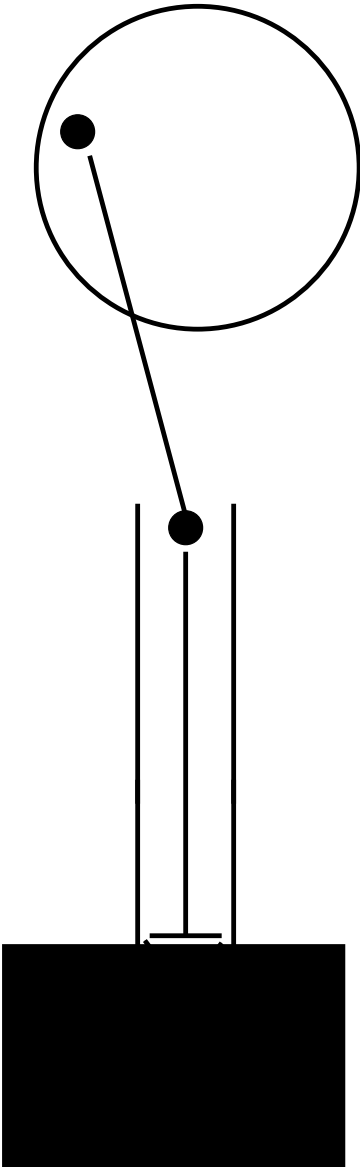


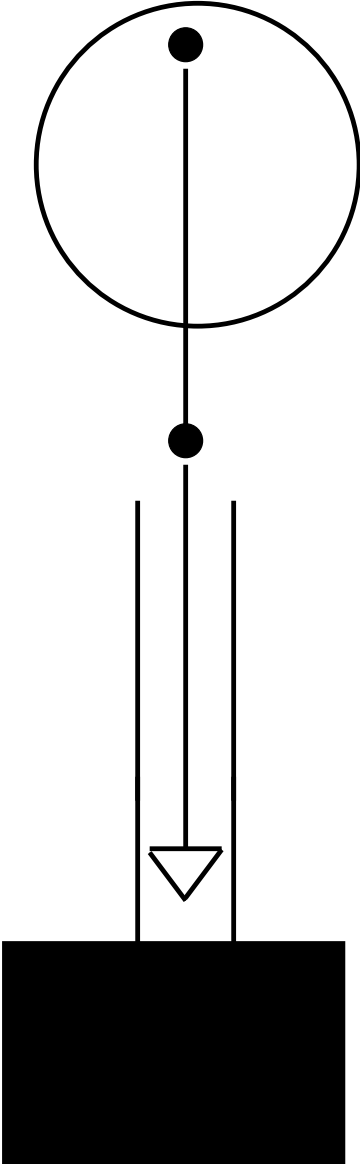


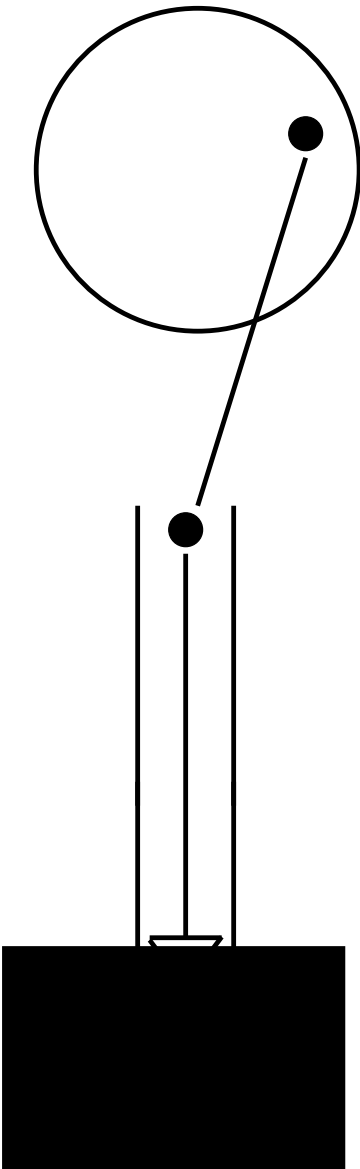


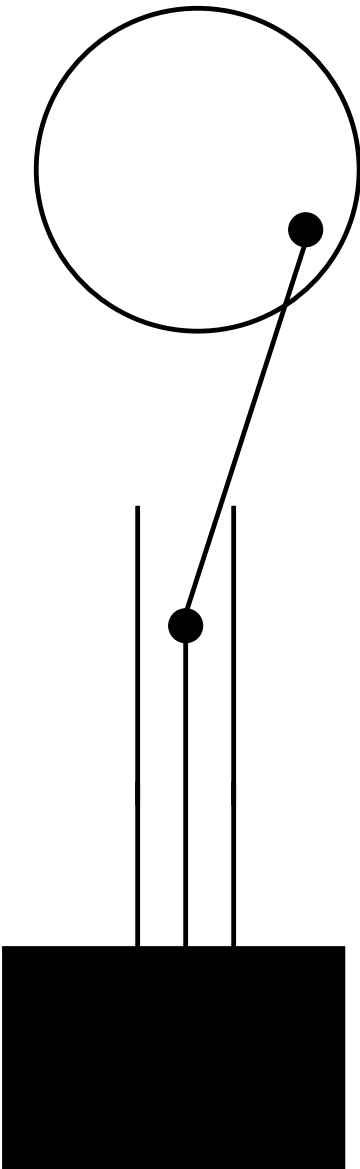


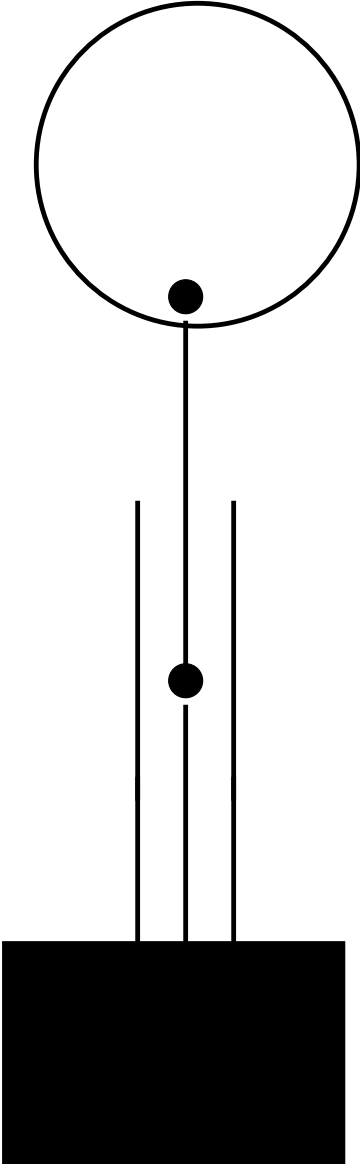


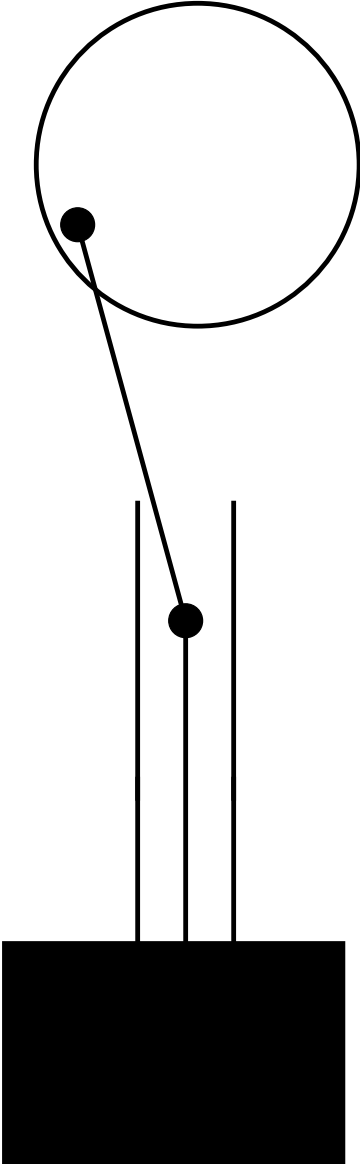


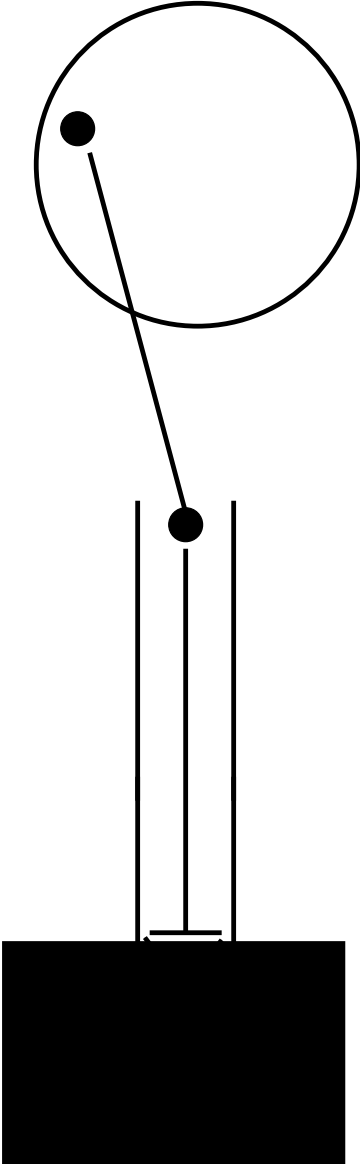


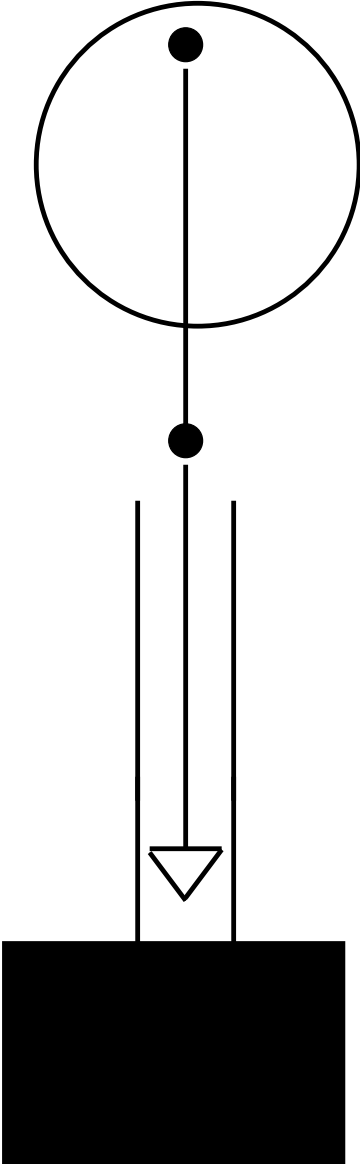


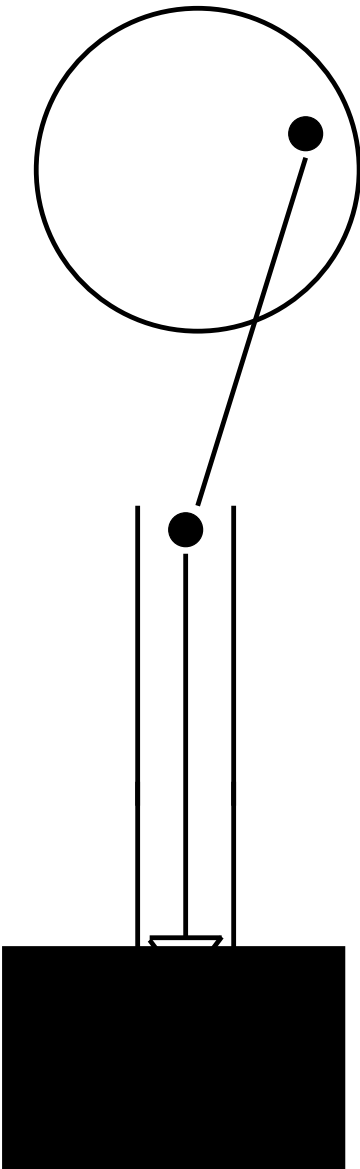


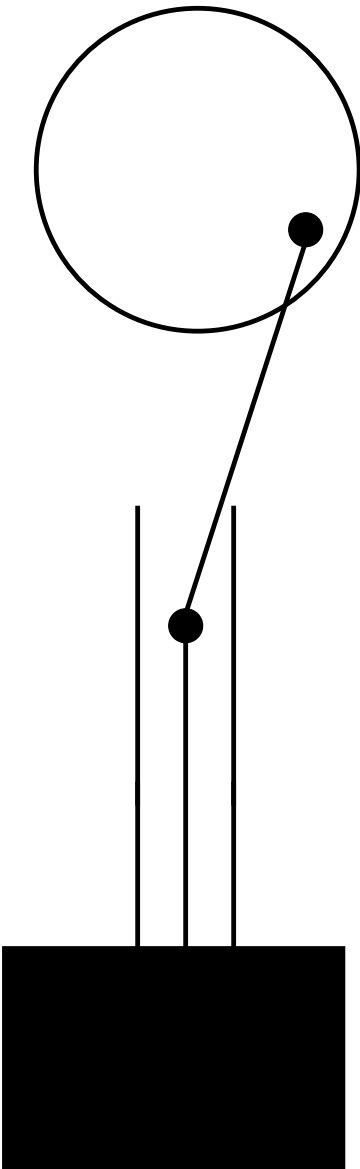


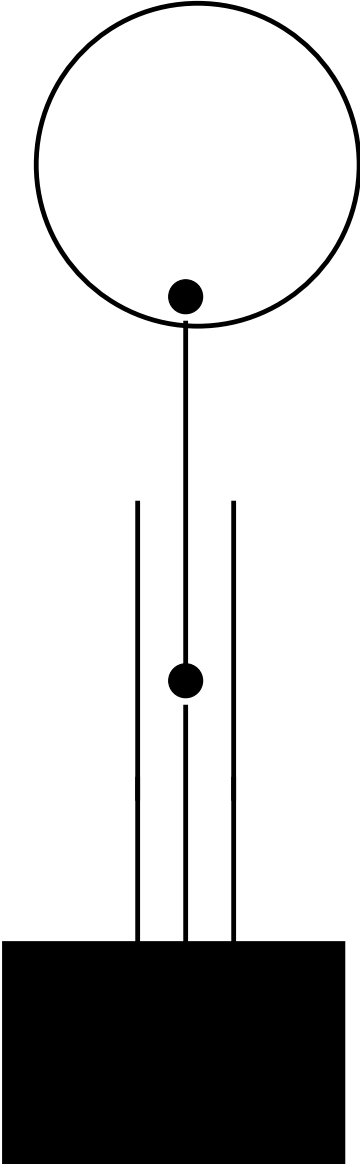


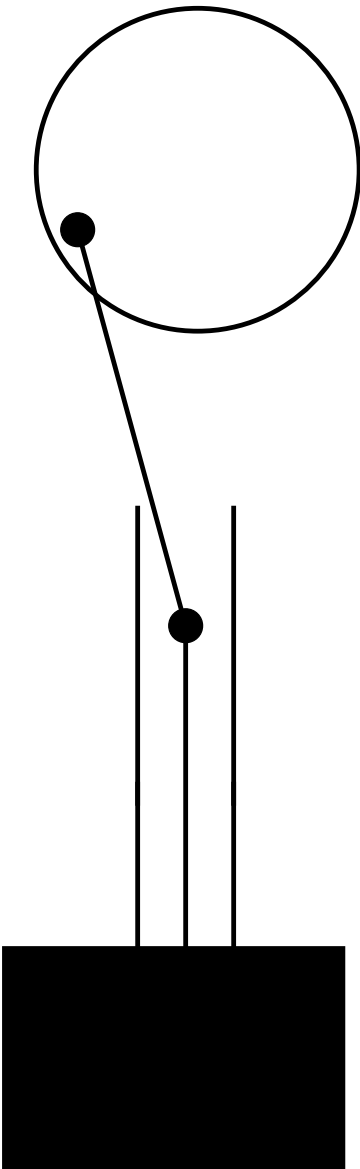


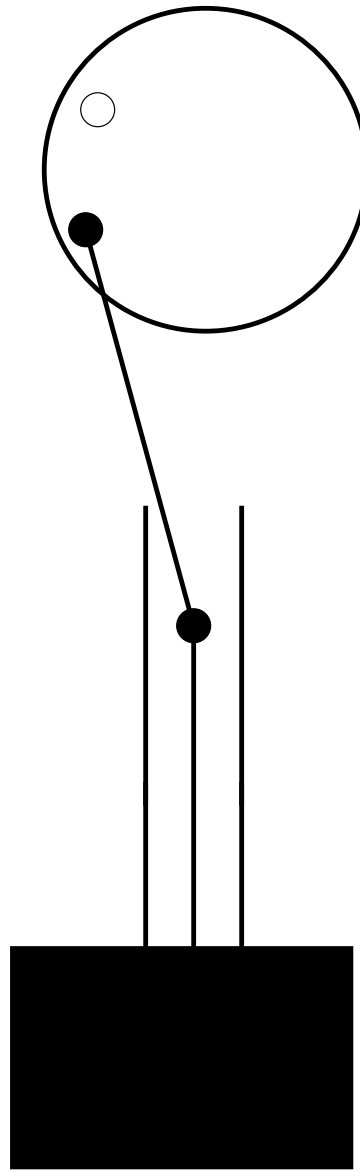


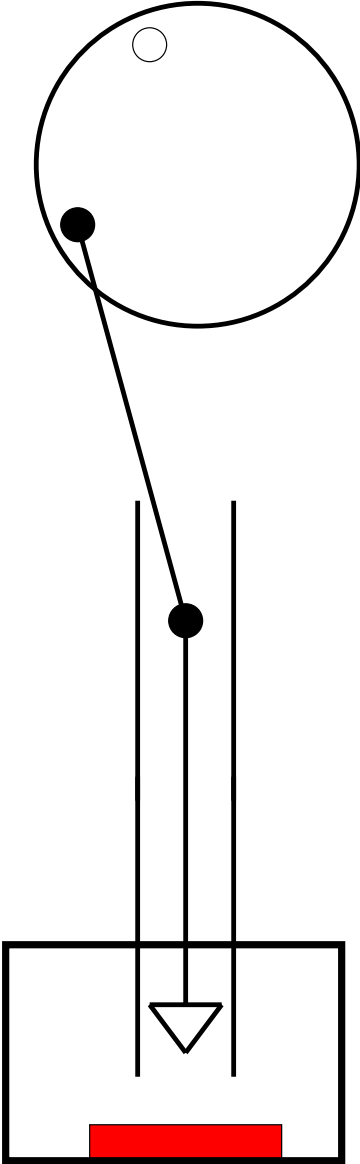


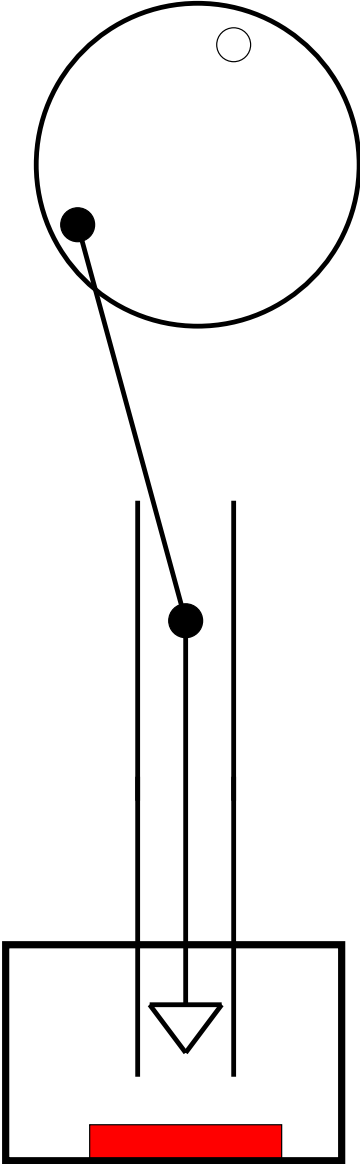


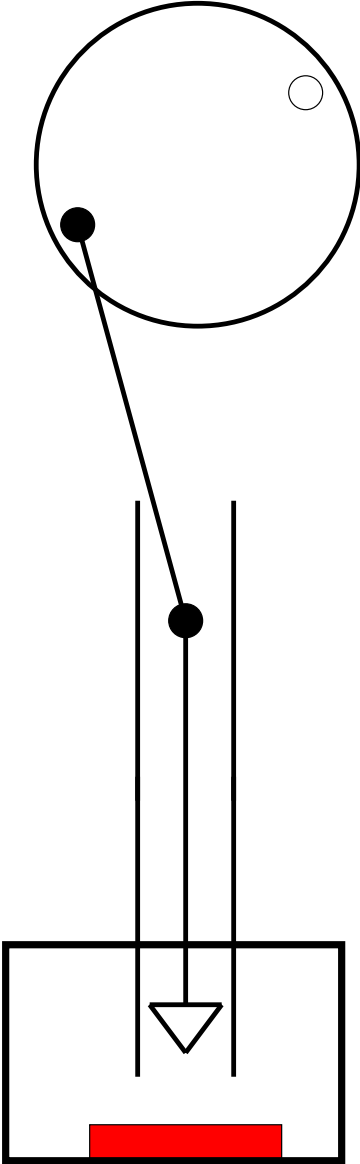


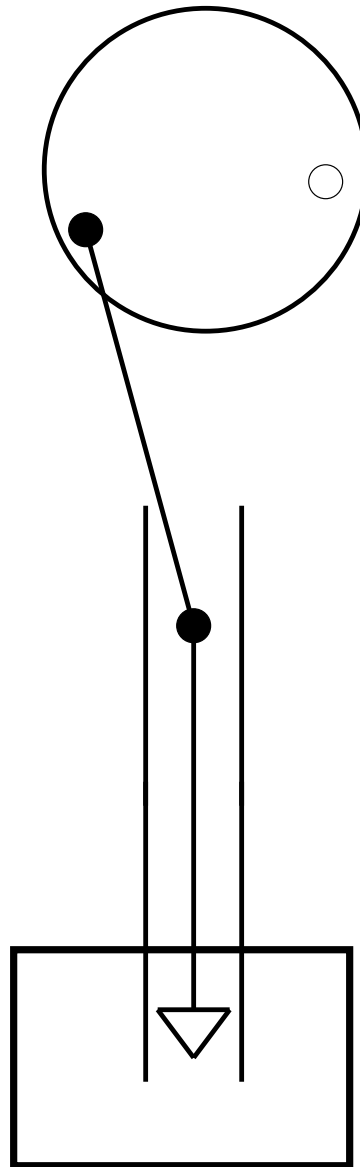


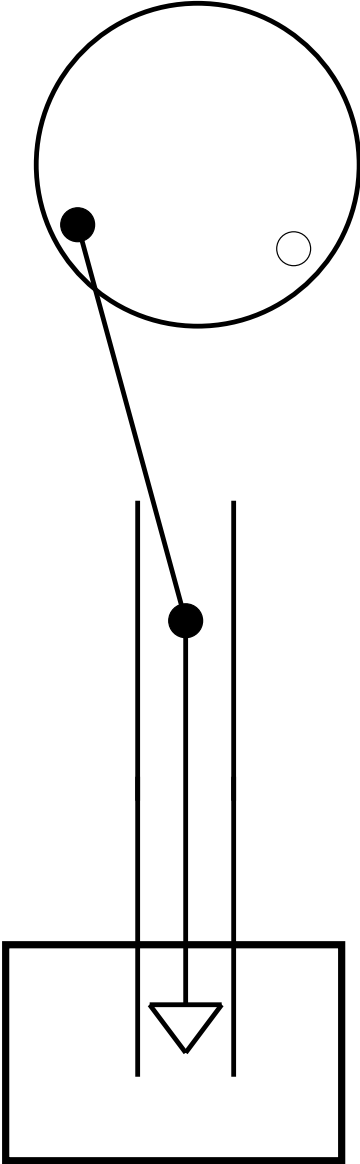


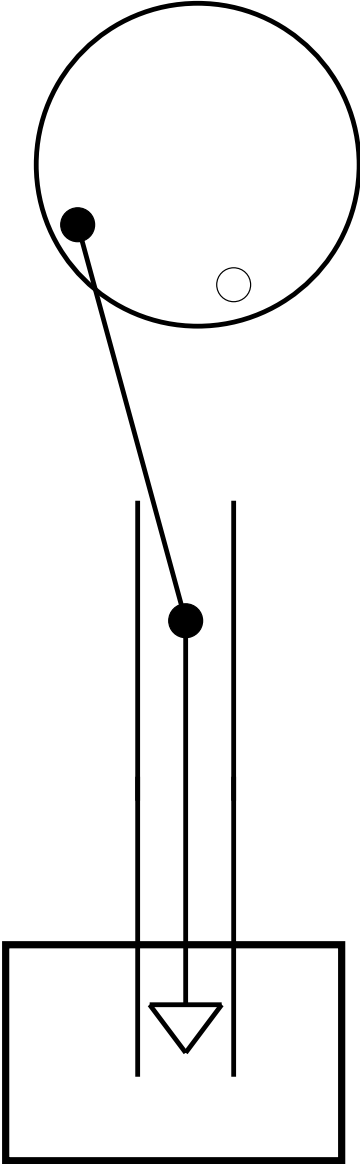


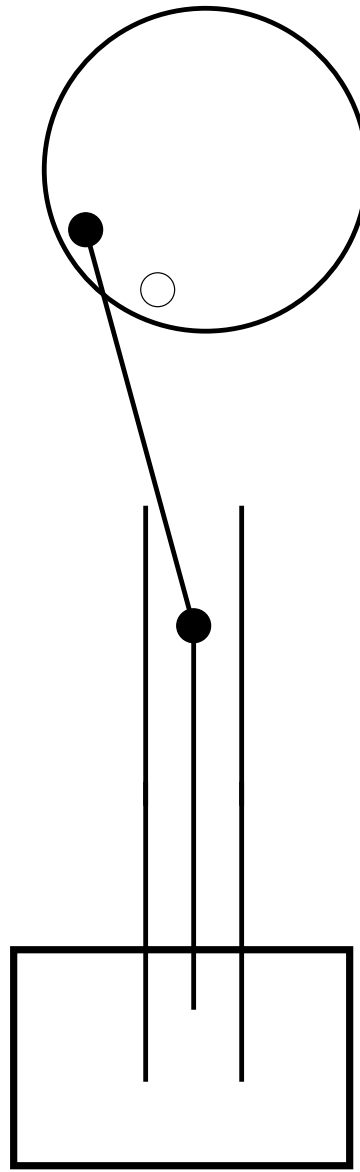


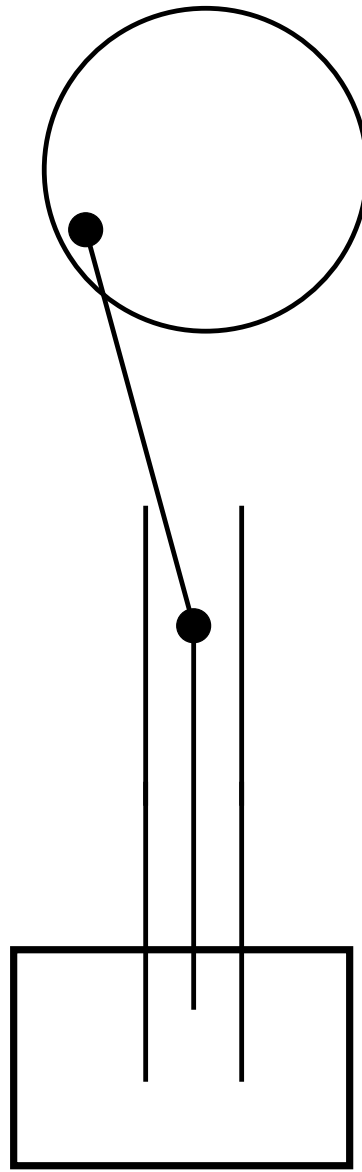


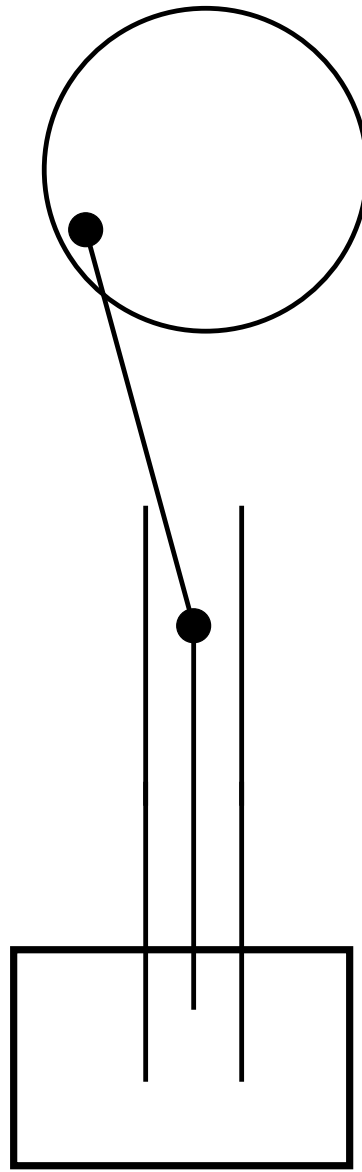


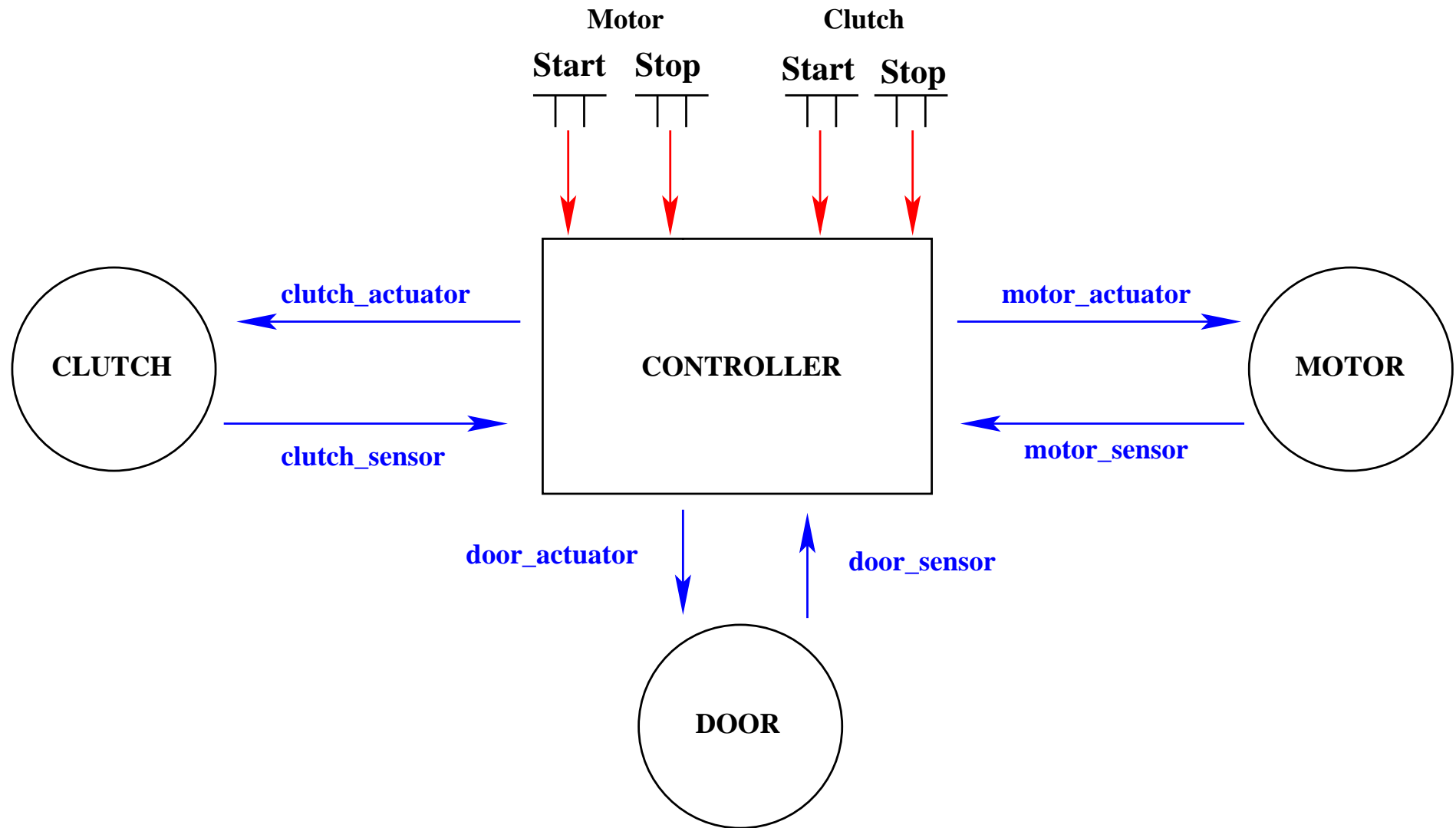












2. Presentation of some Design Patterns

- A number of **similar behaviors**
- Some **complex situations** to handle

- 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**
 - ...

- 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

- Making an **action dependent** of another one:
- **Engaging the clutch implies closing the door first**
- **Disengaging the clutch means opening the door afterwards**

- 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**

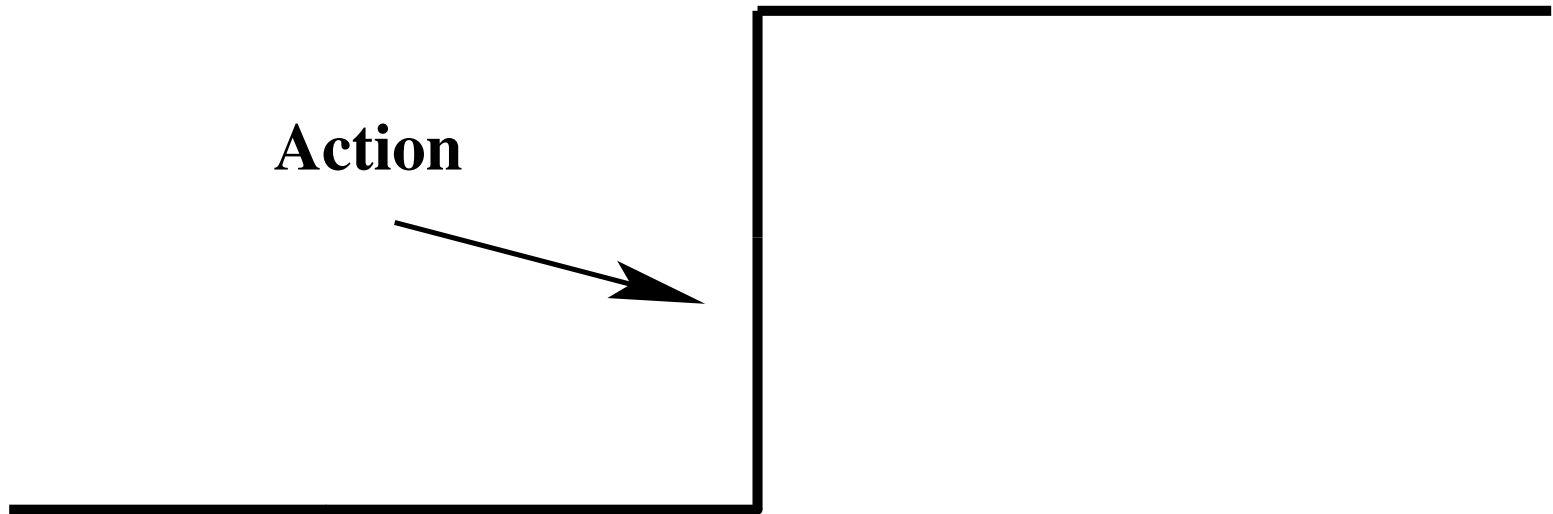
-
- Here is a sequence of events:
 - (1) **User** pushes button B1 (start motor)
 - (2) **Controller** sends the starting command to the motor
 - (3.1) **Motor** starts and sends feedback to the controller
 - (3.2) **User** pushes button B2 (stop motor)
 - (3.1) and (3.2) may occur **simultaneously**
 - 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**

- 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**

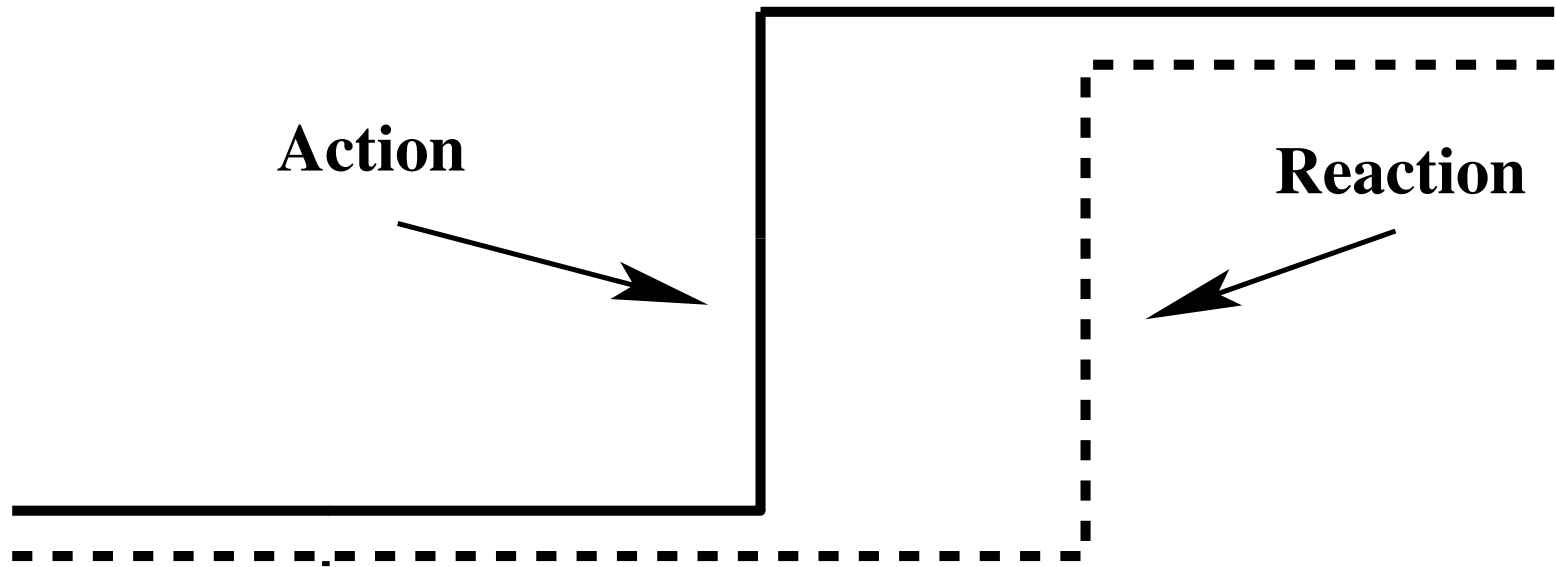
-
- A design pattern **isn't a finished design** that can be transformed into code
 - It is **a template for how to solve a problem** that can be used in many different situations
 - Patterns originated as an **architectural concept** by Christopher Alexander
 - **"Design Patterns: Elements of Reusable Object-Oriented Software"** published in 1994 (Gamma et al)

- 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**

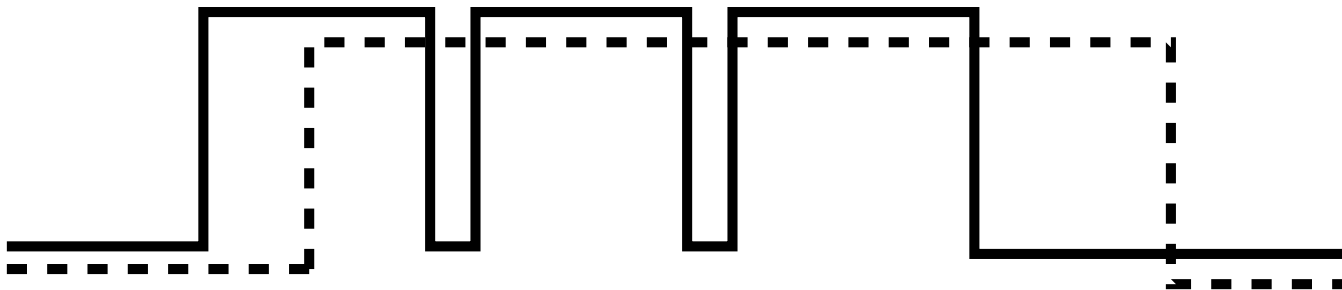
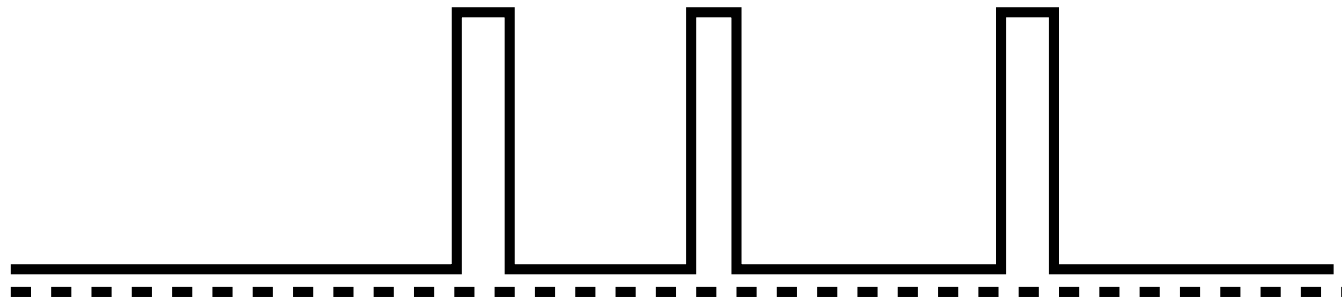


Action

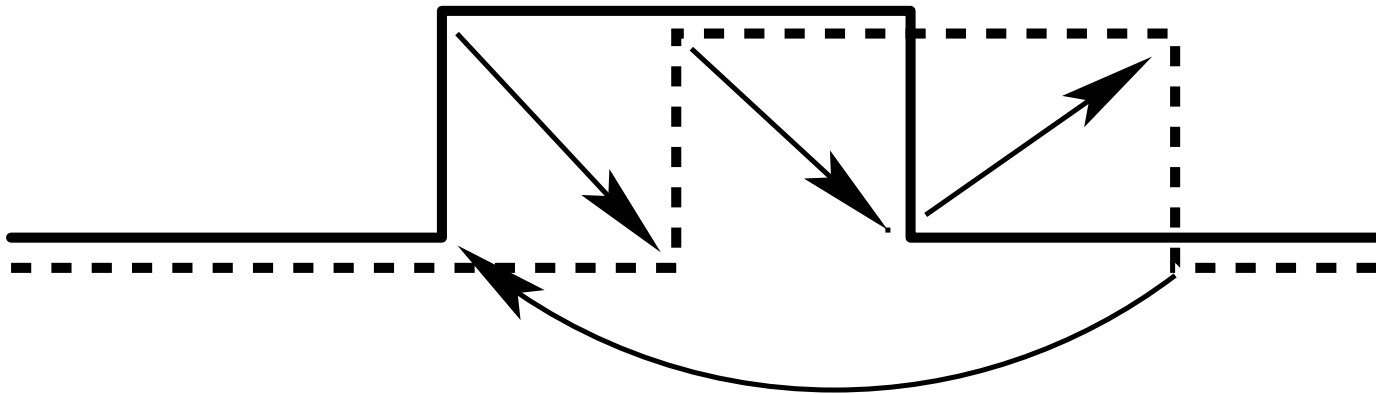




- Sometimes, the reaction has **not enough time** to react
- Because the action moves **too quickly**



- 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

variables: a
 r
 ca
 cr

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

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

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

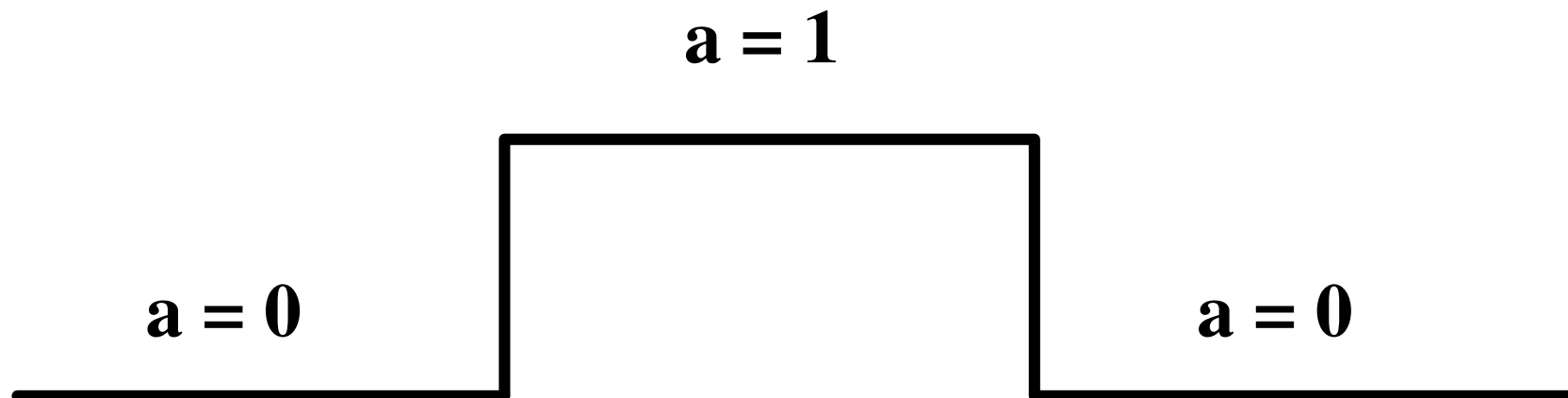
pat0_4: $cr \in \mathbb{N}$

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

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

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

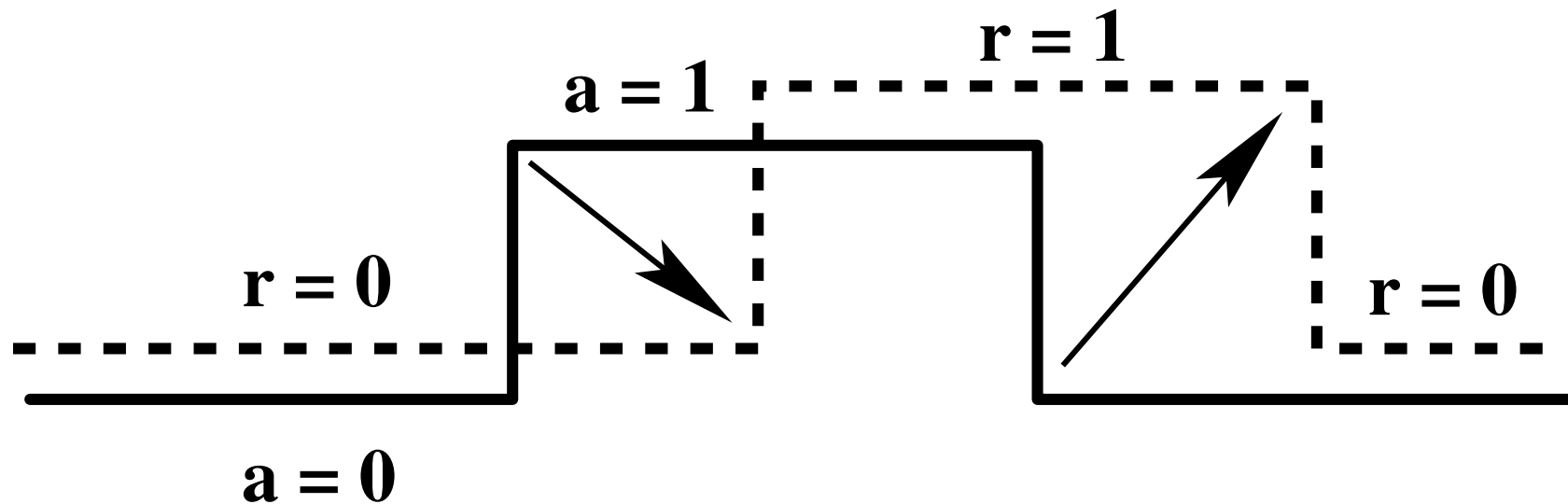


```

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
    
```

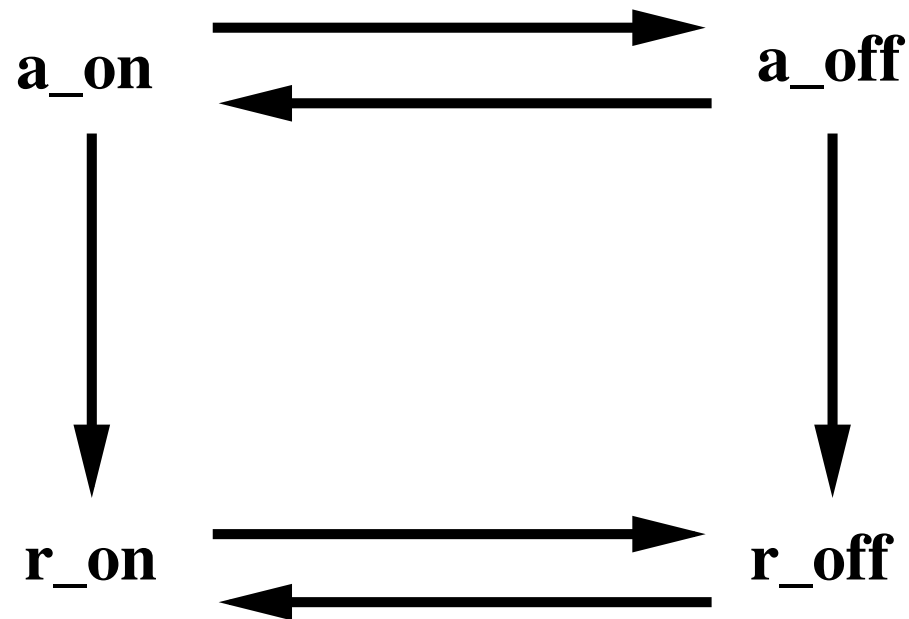


```
a_on
  when
    a = 0
  then
    a := 1
    ca := ca + 1
  end
```

```
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
```



```
variables:  a,  
             r,  
             ca,  
             cr
```

```
pat0_1:  a ∈ {0,1}
```

```
pat0_2:  r ∈ {0,1}
```

```
pat0_3:  ca ∈ ℕ
```

```
pat0_4:  cr ∈ ℕ
```

```
pat0_5:  cr ≤ 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

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**

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

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

- 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
    
```

<p>...</p> <p>$cr \leq ca$</p> <p>$r = 0$</p> <p>$a = 1$</p> <p>⊢</p> <p>$cr + 1 \leq ca$</p>	<p>...</p> <p>pat0_5</p> <p>guard of <i>r_on</i></p> <p>guard of <i>r_on</i></p> <p>modified pat0_5</p>
---	---

- 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
    
```

<p>...</p> <p>$cr \leq ca$</p> <p>$r = 0$</p> <p>$a = 1$</p> <p>⊢</p> <p>$cr + 1 \leq ca$</p>	<p>...</p> <p>pat0_5</p> <p>guard of r_on</p> <p>guard of r_on</p> <p>modified pat0_5</p>
---	---

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

- 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
    
```

<p>...</p> <p>$cr \leq ca$</p> <p>$r = 0$</p> <p>$a = 1$</p> <p>⊢</p> <p>$cr + 1 \leq ca$</p>	<p>...</p> <p>pat0_5</p> <p>guard of <i>r_on</i></p> <p>guard of <i>r_on</i></p> <p>modified pat0_5</p>
---	---

- 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

...

$cr \leq ca$

$r = 0$

$a = 1$

$cr < ca$

⊢

$cr + 1 \leq ca$

...

pat0_5

guard of r_on

guard of r_on

new guard of r_on

modified **pat0_5**

- Drawback: **One has to keep variables cr and ca in the guard**
- These variables were introduced **for the modelling only**

- 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
    
```

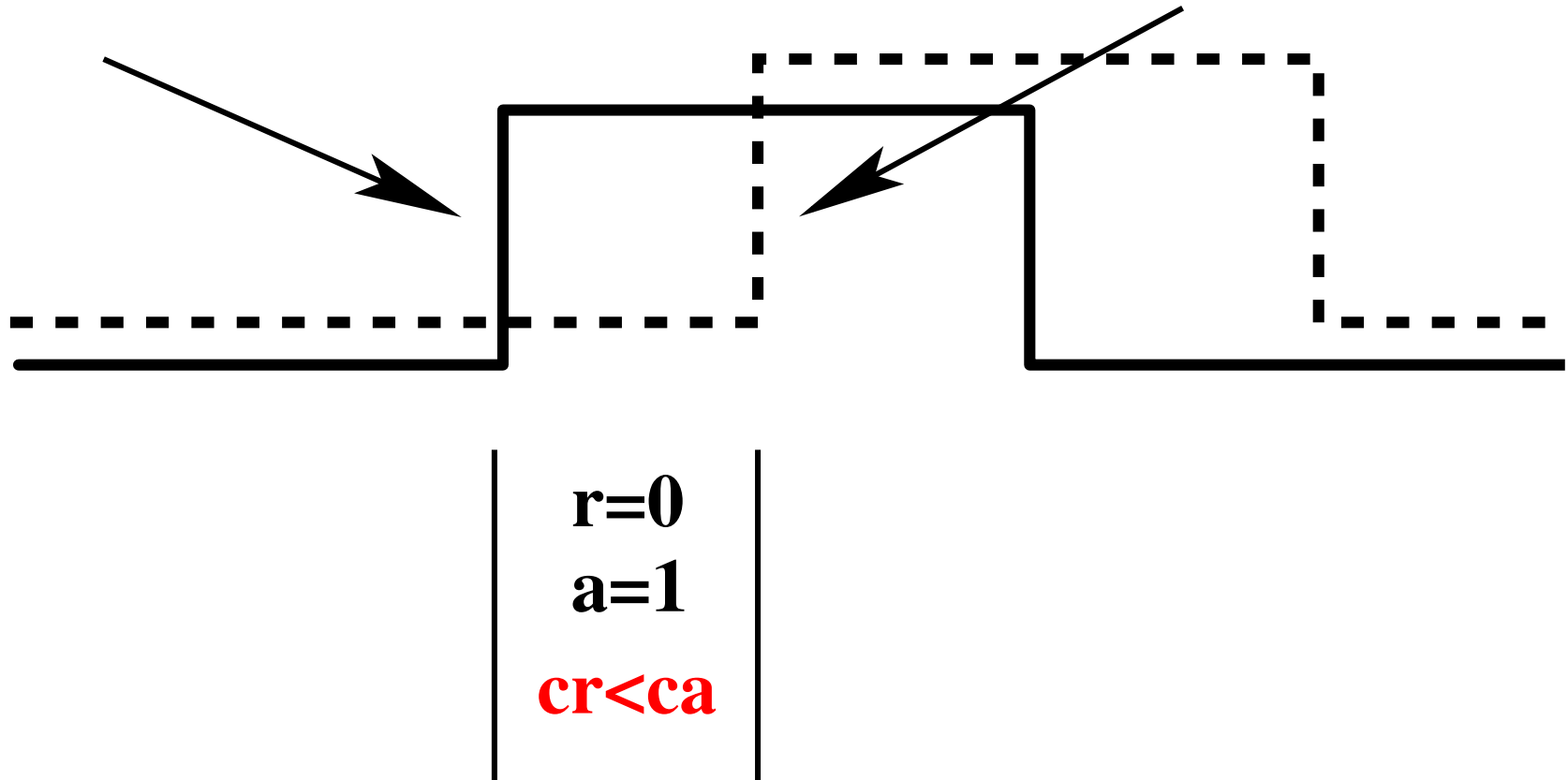
<p>...</p> <p>$cr \leq ca$</p> <p>$r = 0 \wedge a = 1 \Rightarrow cr < ca$</p> <p>$r = 0$</p> <p>$a = 1$</p> <p>⊢</p> <p>$cr + 1 \leq ca$</p>	<p>guard of r_on</p> <p>guard of r_on</p>	<p>pat0_5</p> <p>pat0_6</p> <p>modified pat0_5</p>
---	---	--

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

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

ca is incremented

cr is incremented



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)

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

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

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

$$r = 0$$

$$a = 1$$

⊢

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

which holds trivially

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

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

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

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

$$\text{pat0_5: } cr \leq ca$$

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

The counters have
been removed

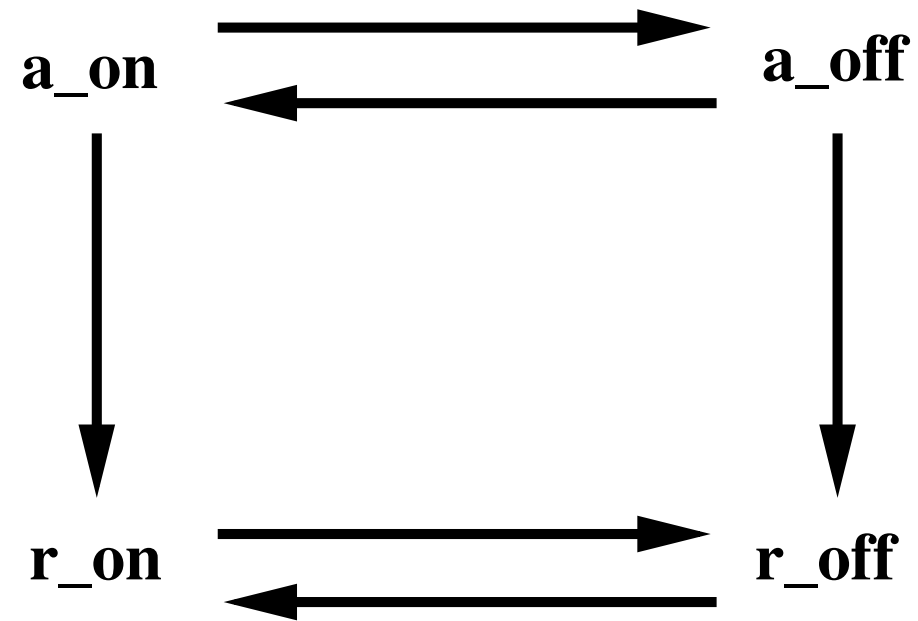
```
init
  a := 0
  r := 0
```

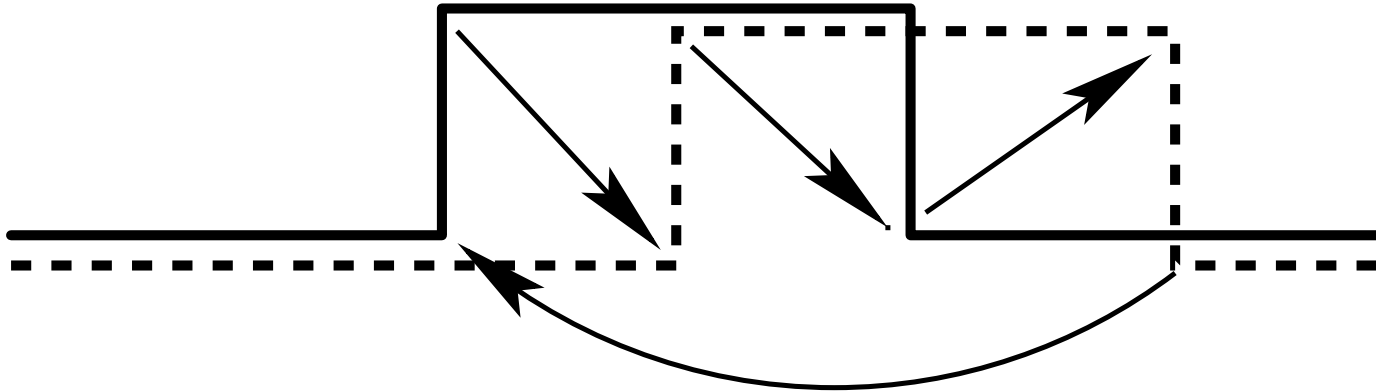
```
a_on
  when
    a = 0
  then
    a := 1
  end
```

```
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
```





- We add the following invariant

$$\text{pat1_1: } ca \leq cr + 1$$

- Remember invariant **pat0_5**

$$\text{pat0_5: } cr \leq ca$$

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

- 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 <code>a_on</code>
\vdash	
$ca + 1 \leq cr + 1$	modified pat1_1

- 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 <code>a_on</code>
\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 <code>a_on</code>
\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**)

- 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 <code>a_on</code>
\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 <code>a_on</code>
\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: $a = 0 \Rightarrow cr = ca$

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**

$$\text{pat1_2: } a = 0 \Rightarrow cr = ca$$

- The proof of maintenance of this invariant by `a_off` fails

`a_off`

when

`a = 1`

then

`a := 0`

end

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

$$a = 1$$

⊢

$$0 = 0 \Rightarrow cr = ca$$

pat1_2

guard of `a_off`

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** ($a = 0 \Rightarrow cr = ca$)

<pre> a_off when a = 1 then a := 0 end </pre>	$a = 0 \Rightarrow cr = ca$	pat1_2
<pre> a = 1 then a := 0 end </pre>	$a = 1 \wedge r = 1 \Rightarrow cr = ca$	pat1_3
<pre> a := 0 </pre>	$a = 1$	guard
<pre> end </pre>	\vdash	
<pre> end </pre>	$0 = 0 \Rightarrow cr = ca$	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
⊢	
$0 = 0 \Rightarrow cr = ca$	modified pat1_2

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

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

- Event `a_on` cannot maintain invariant **pat1_3**

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

`a_on`

when

`a = 0`

then

`a := 1`

`ca := ca + 1`

end

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

$$a = 0$$

⊢

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

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

- Event `a_on` cannot maintain invariant **pat1_3**

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

```

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

$$\begin{array}{l}
 a = 1 \wedge r = 1 \Rightarrow cr = ca \\
 a = 0 \\
 r = 0 \quad \text{new guard} \\
 \vdash \\
 1 = 1 \wedge r = 1 \Rightarrow cr = ca + 1
 \end{array}$$

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

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

```
a_on
  when
    r = 0
    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
```

- 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
```

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

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
    
```

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

$$ca \leq cr + 1$$

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

$$r = 0$$

$$a = 1$$

⊢

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

- Everything is proved now

- We do not need to add more invariants

The counters have
been removed

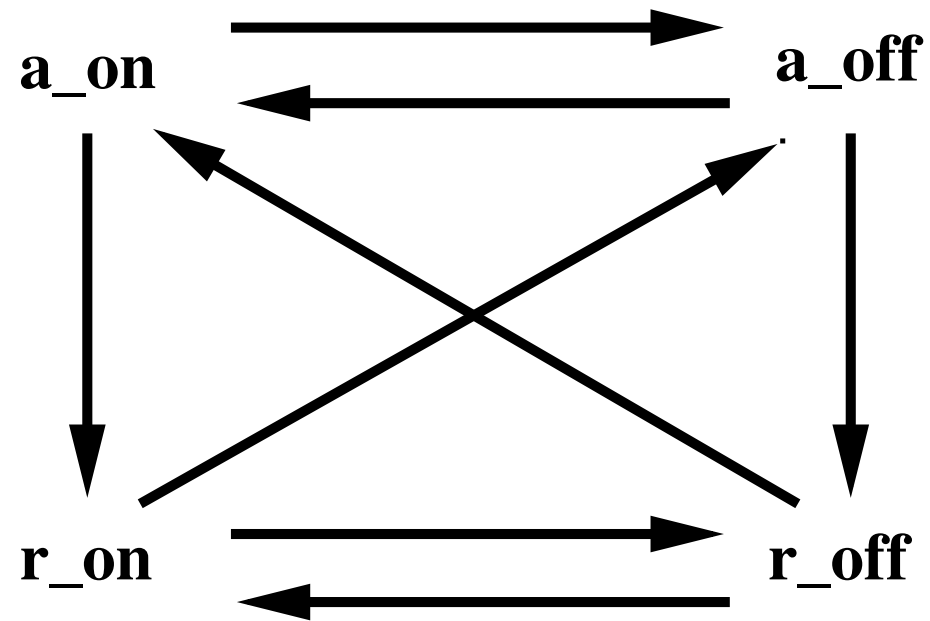
```
init
   $a := 0$ 
   $r := 0$ 
```

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

```
a_off
  when
     $a = 1$ 
     $r = 1$ 
  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
```



- Putting together these two invariants

$$\mathbf{pat1_2:} \quad a = 0 \Rightarrow ca = cr$$

$$\mathbf{pat1_3:} \quad a = 1 \wedge r = 1 \Rightarrow cr = ca$$

- leads to the following

$$\mathbf{pat1_4:} \quad a = 0 \vee r = 1 \Rightarrow ca = cr$$

$$\text{pat0_5: } cr \leq ca$$

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

$$\text{pat1_1: } ca \leq cr + 1$$

$$\text{pat1_4: } a = 0 \vee r = 1 \Rightarrow ca = cr$$

This can be simplified to

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

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

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

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

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

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

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

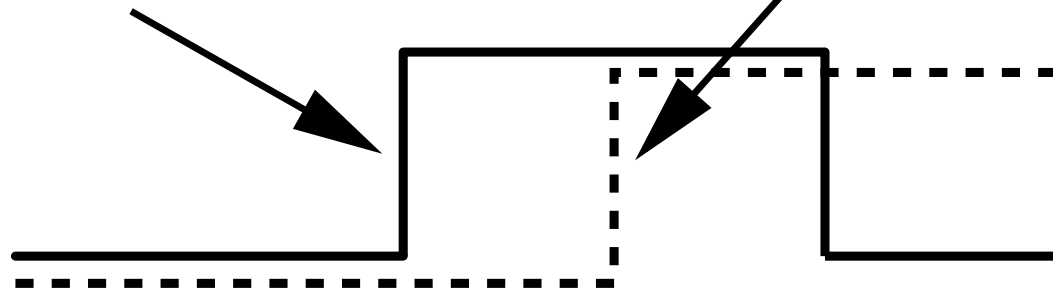
$$\text{pat2_2: } a = 0 \vee r = 1 \Rightarrow ca = cr$$

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

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

ca is incremented

cr is incremented



pat2_2	pat2_1	pat2_2
a=0	a=1	r=1
ca = cr	r=0	ca = cr
	ca=cr+1	

-
- **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

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

Buttons and Controller are weakly synchronized	FUN_1
--	-------

Controller are Equipment are strongly synchronized	FUN_2
--	-------

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

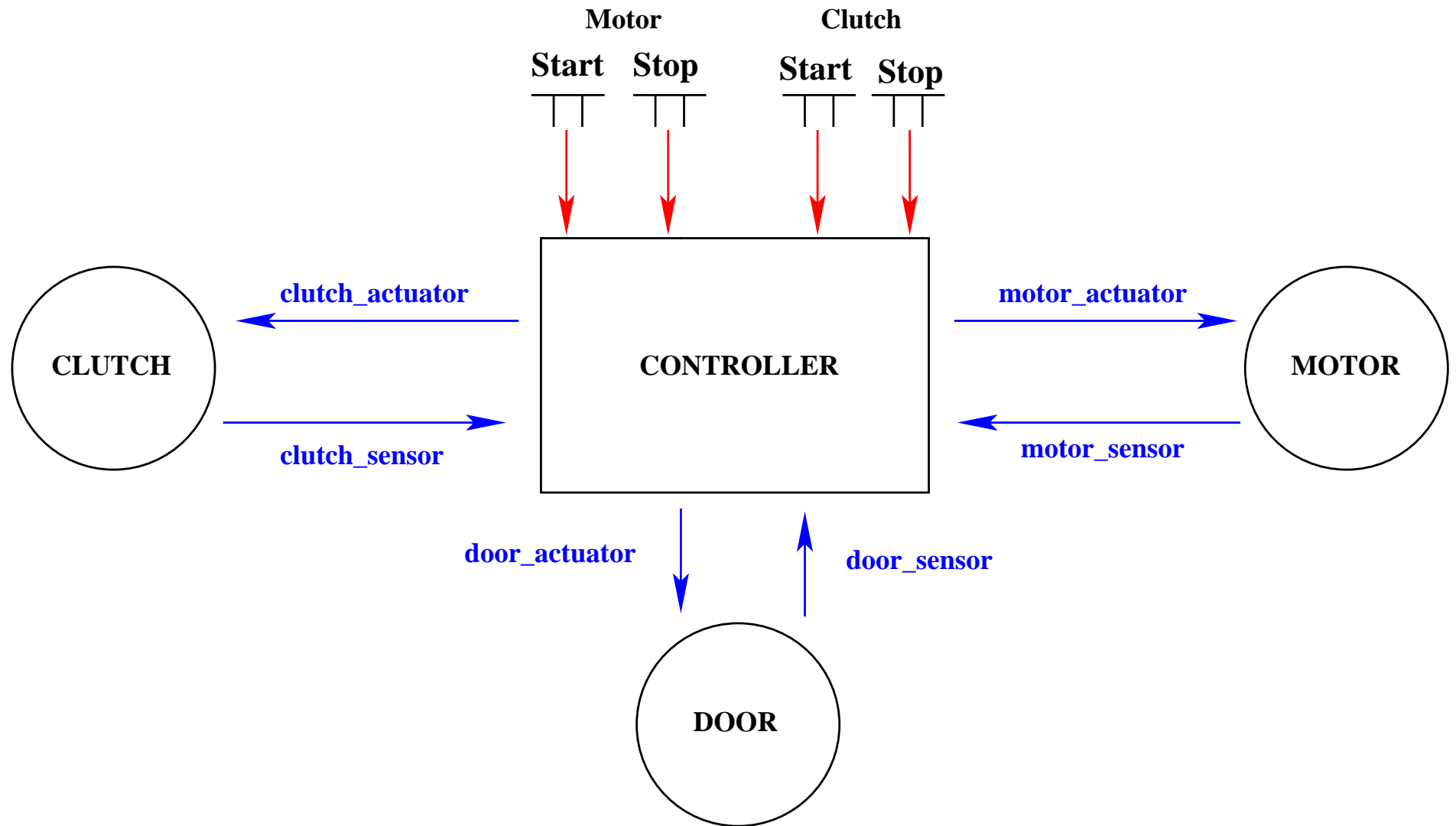
FUN_3

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

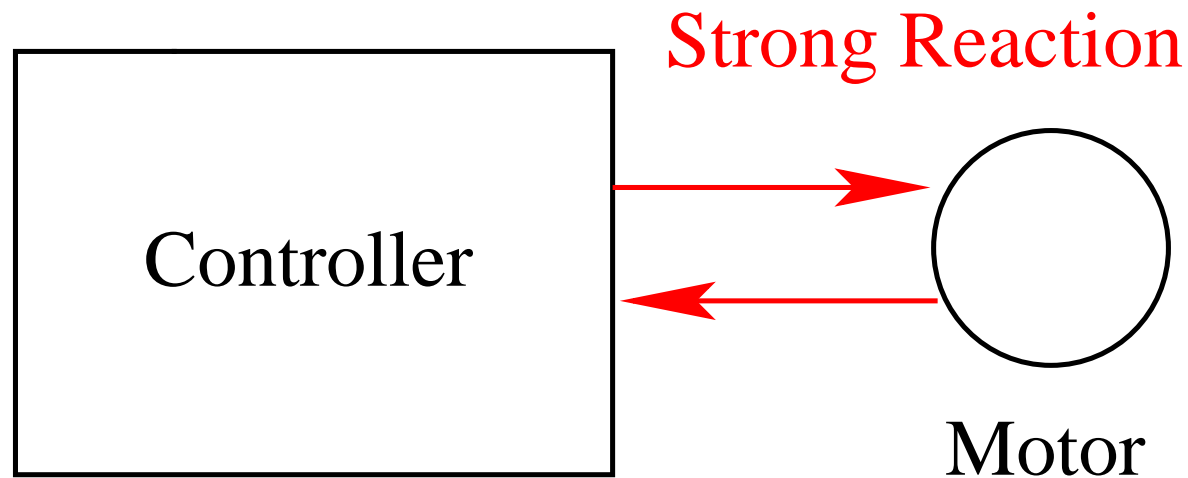


4. Proposing a Refinement Strategy

- 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**

- 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

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

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

```
a_off
  when
     $a = 1$ 
     $r = 1$ 
  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
```

set: *STATUS*

constants: *stopped*
 working

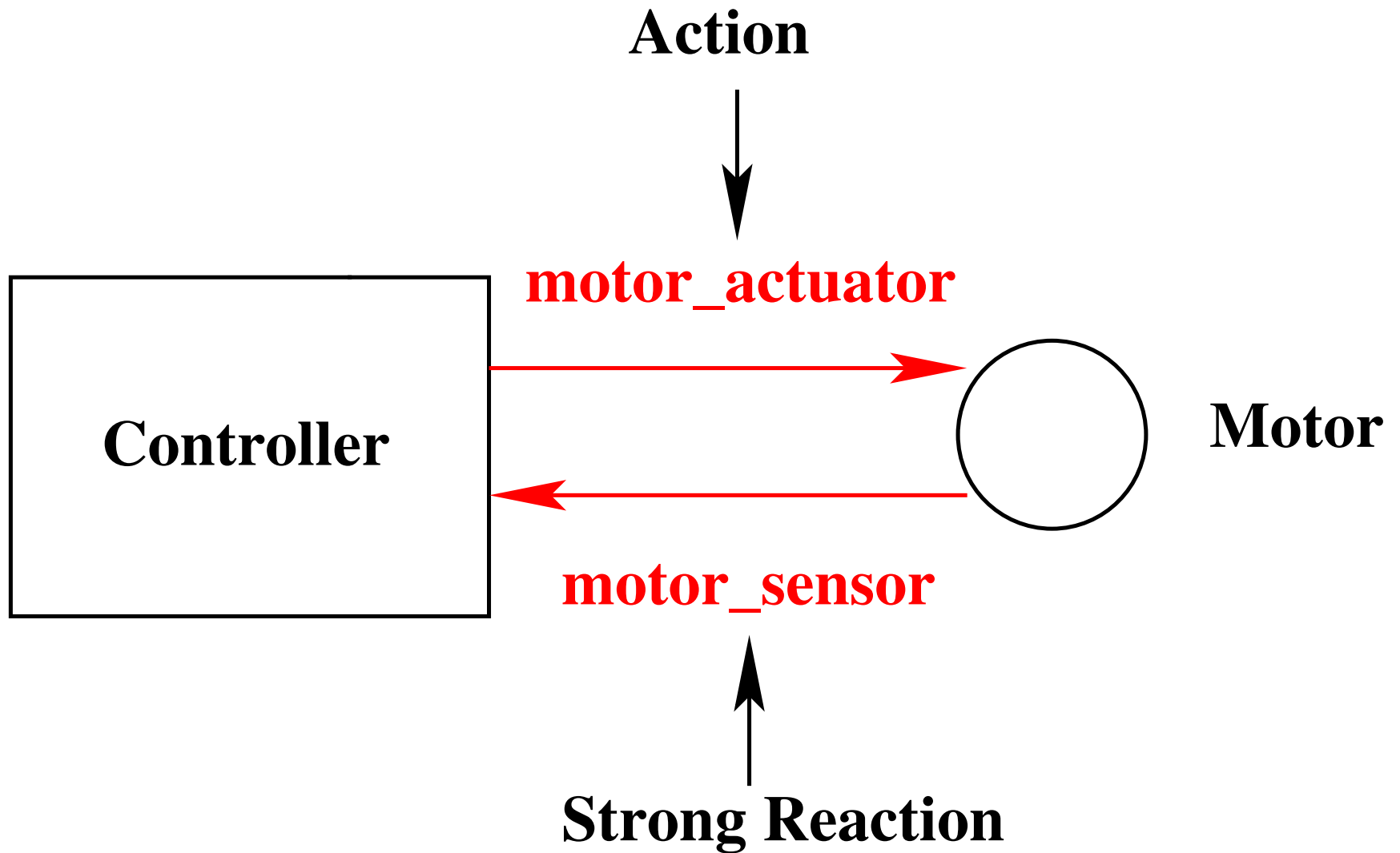
axm0_1: *STATUS = {stopped, working}*

axm0_2: *stopped ≠ working*

variables: *motor_actuator*
 motor_sensor

inv0_1: *motor_sensor* \in *STATUS*

inv0_2: *motor_actuator* \in *STATUS*



- We **instantiate the weak pattern** as follows:

<i>a</i>	\rightsquigarrow	<i>motor_actuator</i>
<i>r</i>	\rightsquigarrow	<i>motor_sensor</i>
0	\rightsquigarrow	<i>stopped</i>
1	\rightsquigarrow	<i>working</i>

a_on	\rightsquigarrow	treat_start_motor
a_off	\rightsquigarrow	treat_stop_motor
r_on	\rightsquigarrow	Motor_start
r_off	\rightsquigarrow	Motor_stop

- Convention: **Controller events** start with "**treat_**"

init

$a := 0$

$r := 0$

init

motor_actuator := stopped

motor_sensor := stopped

```
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

Motor_start

when

motor_sensor = stopped

motor_actuator = working

then

motor_sensor := working

end

```
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
```

r_off

when

$r = 1$

$a = 0$

then

$r := 0$

end

Motor_stop

when

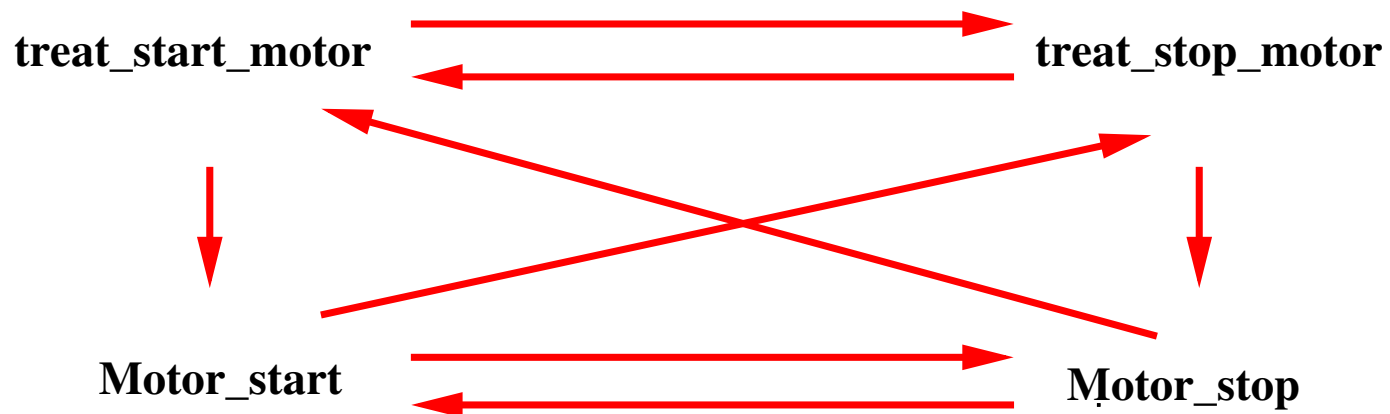
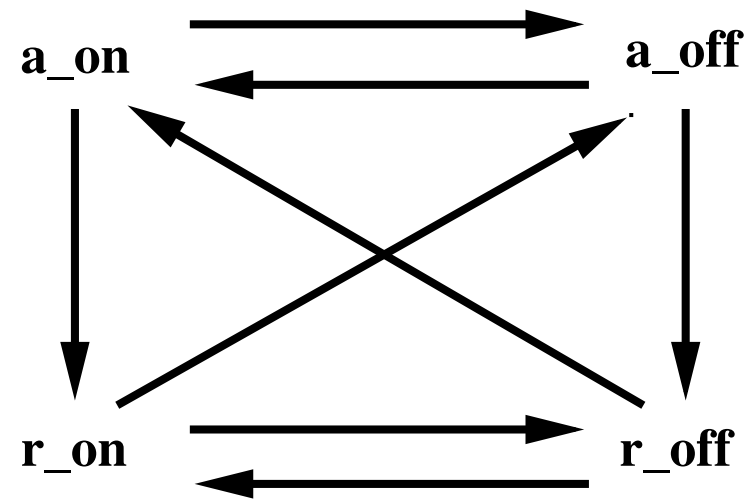
motor_sensor = working

motor_actuator = stopped

then

motor_sensor := stopped

end



- Environment

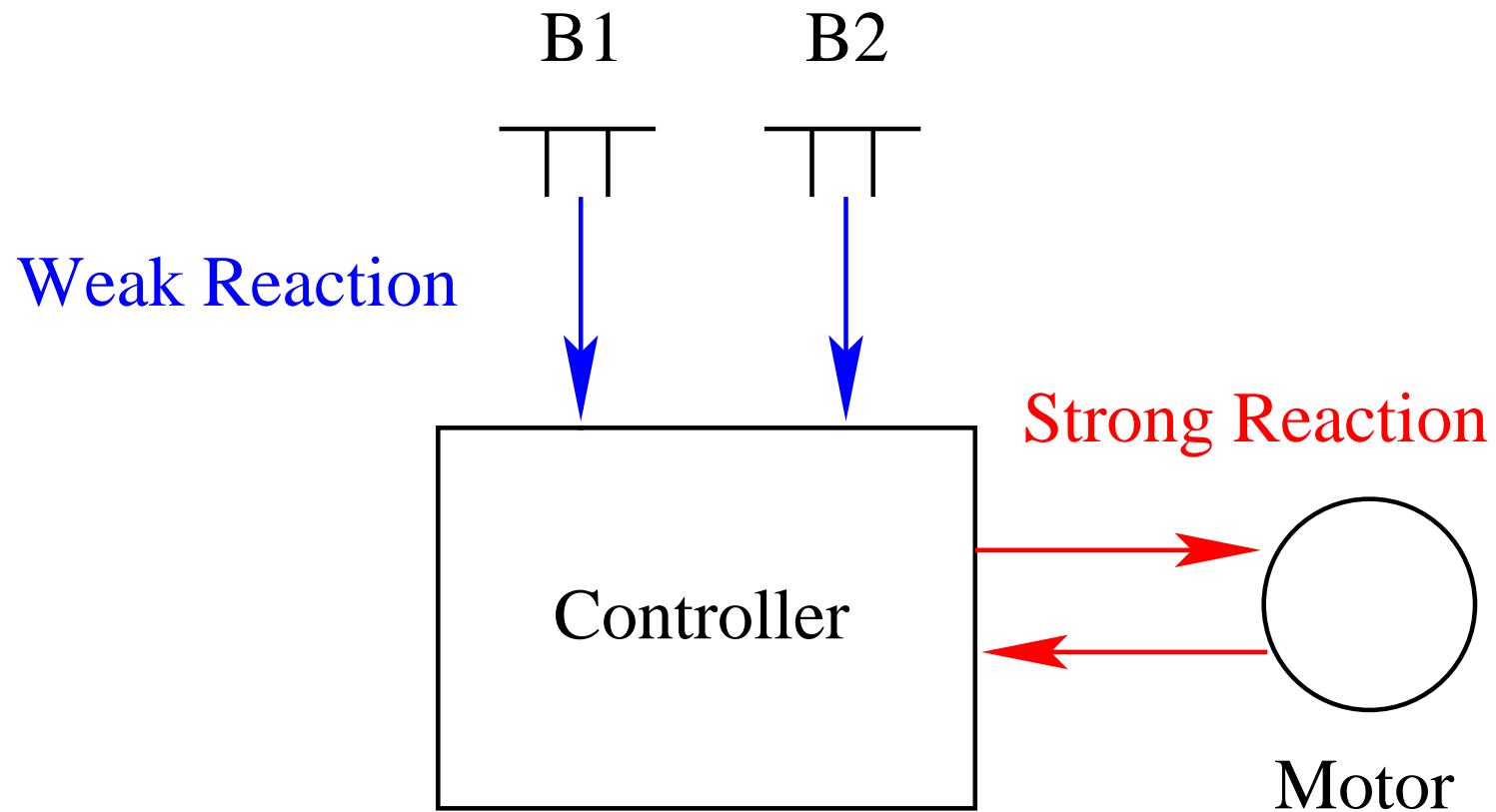
 - motor_start

 - motor_stop

- Controller

 - treat_start_motor

 - treat_stop_motor



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
    
```

```

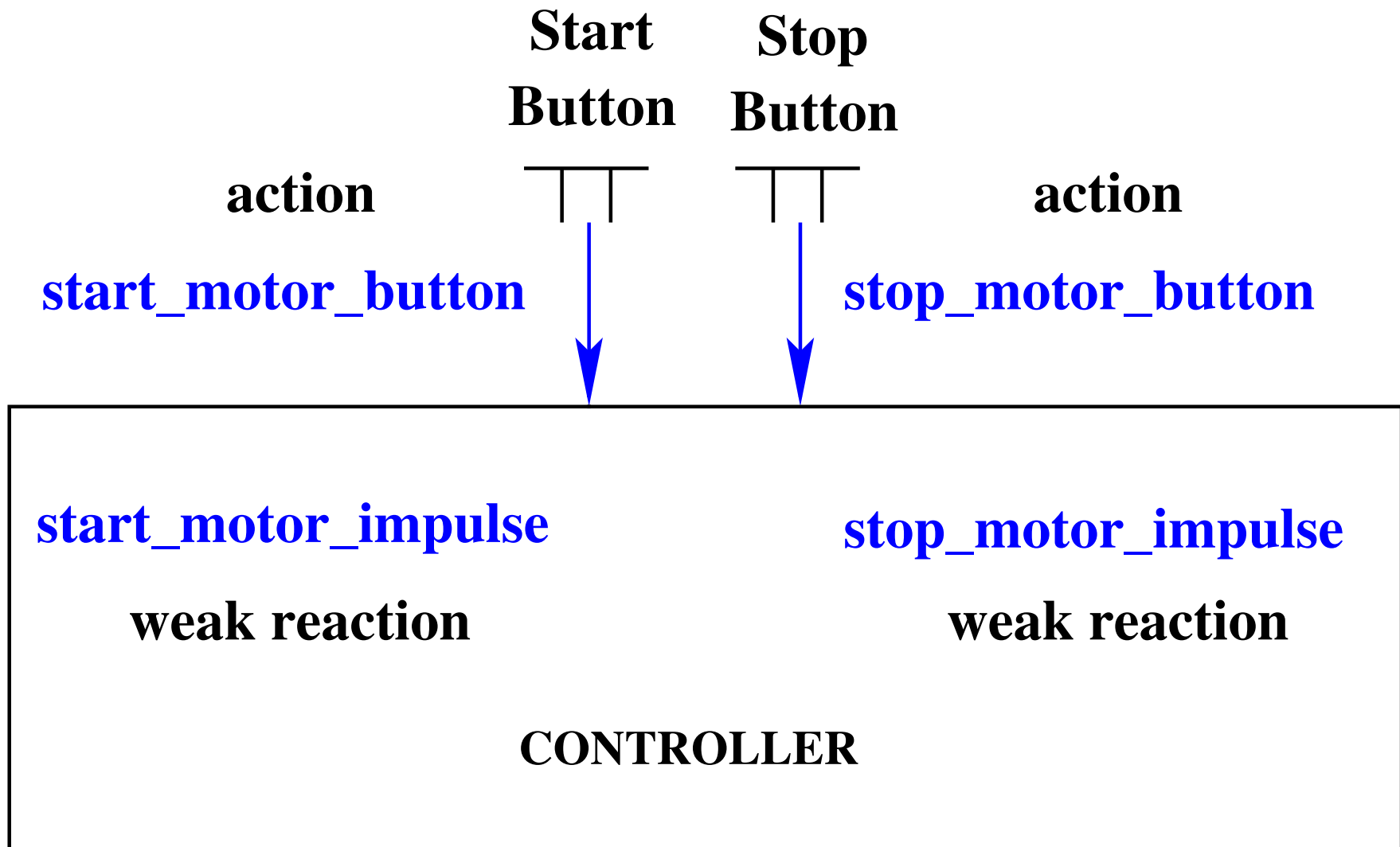
r_on
  when
    r = 0
    a = 1
  then
    r := 1
  end
    
```

```

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

variables: ...
 start_motor_button
 stop_motor_button
 start_motor_impulse
 stop_motor_impulse

inv1_1: *stop_motor_button* ∈ BOOL
inv1_2: *start_motor_button* ∈ BOOL
inv1_3: *stop_motor_impulse* ∈ BOOL
inv1_4: *start_motor_impulse* ∈ BOOL



- We **instantiate the pattern** as follows:

<i>a</i>	↪	<i>start_motor_button</i>
<i>r</i>	↪	<i>start_motor_impulse</i>
0	↪	FALSE
1	↪	TRUE

a_on	↪	push_start_motor_button
a_off	↪	release_stop_motor_button
r_on	↪	treat_push_start_motor_button
r_off	↪	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 := FALSE

start_motor_impulse := FALSE

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

```
push_start_motor_button  
  when  
     $start\_motor\_button = FALSE$   
  then  
     $start\_motor\_button := TRUE$   
  end
```

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

```
release_start_motor_button  
  when  
     $start\_motor\_button = TRUE$   
  then  
     $start\_motor\_button := FALSE$   
  end
```

```
r_on
```

```
  when
```

```
     $r = 0$ 
```

```
     $a = 1$ 
```

```
  then
```

```
     $r := 1$ 
```

```
end
```

```
treat_push_start_motor_button
```

```
  refines
```

```
    treat_start_motor
```

```
  when
```

```
     $start\_motor\_impulse = FALSE$ 
```

```
     $start\_motor\_button = TRUE$ 
```

```
     $motor\_actuator = stopped$ 
```

```
     $motor\_sensor = stopped$ 
```

```
  then
```

```
     $start\_motor\_impulse := TRUE$ 
```

```
     $motor\_actuator := working$ 
```

```
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 = FALSE

start_motor_button = TRUE

motor_actuator = stopped

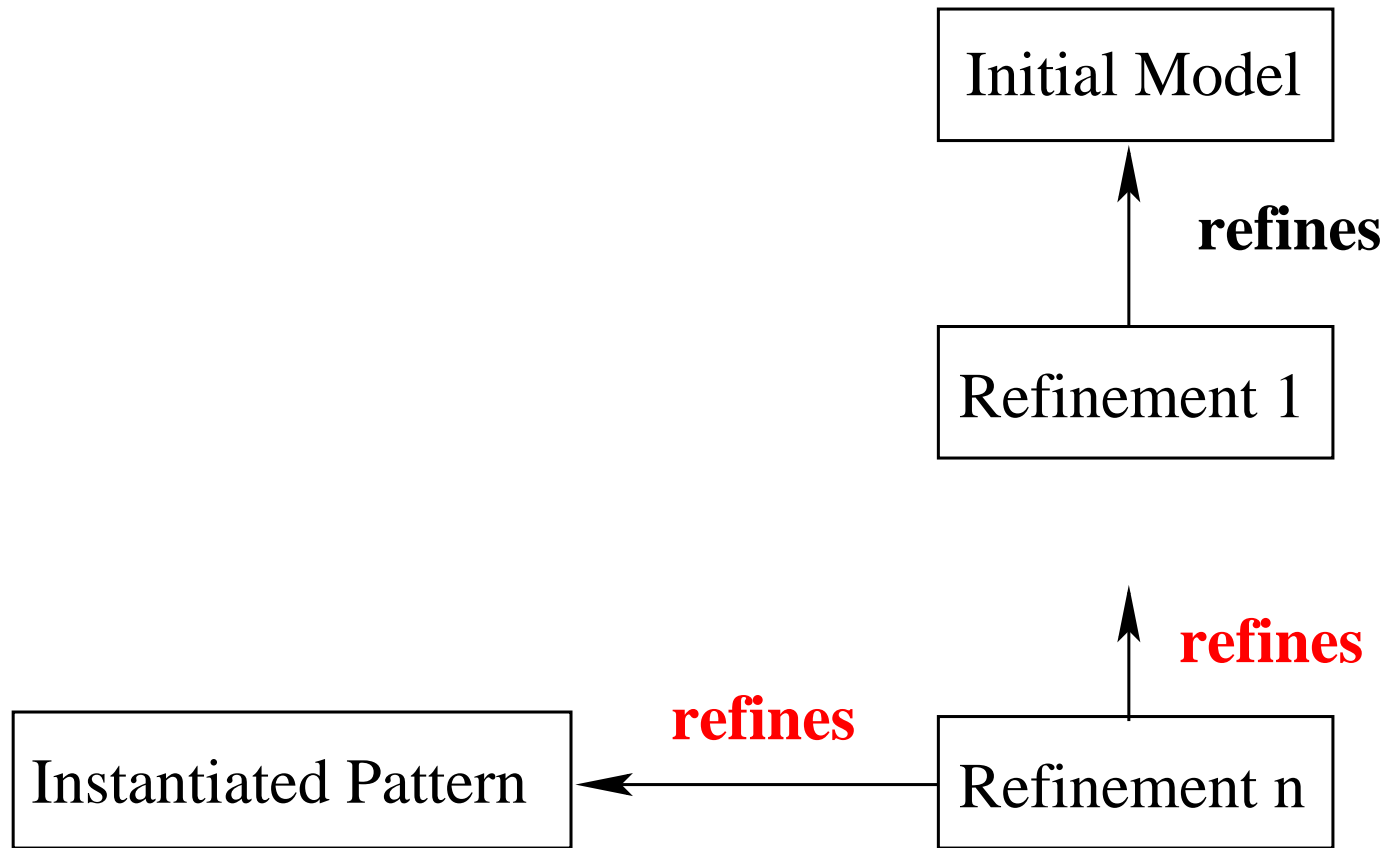
motor_sensor = stopped

then

start_motor_impulse := TRUE

motor_actuator := working

end



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

```
treat_release_start_motor_button  
  when  
     $start\_motor\_impulse = TRUE$   
     $start\_motor\_button = FALSE$   
  then  
     $start\_motor\_impulse := FALSE$   
  end
```

- We **instantiate the pattern** as follows:

<i>a</i>	~→	<i>stop_motor_button</i>
<i>r</i>	~→	<i>stop_motor_impulse</i>
0	~→	FALSE
1	~→	TRUE

a_on	~→	push_stop_motor_button
a_off	~→	release_stop_motor_button
r_on	~→	treat_push_stop_motor_button
r_off	~→	treat_release_stop_motor_button

init

$a := 0$

$r := 0$

init

motor_actuator := stopped

motor_sensor := stopped

start_motor_button := FALSE

start_motor_impulse := FALSE

stop_motor_button := FALSE

stop_motor_impulse := FALSE

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

```
push_stop_motor_button
  when
    stop_motor_button = FALSE
  then
    stop_motor_button := TRUE
  end
```

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

```
release_stop_motor_button
  when
    stop_motor_button = TRUE
  then
    stop_motor_button := FALSE
  end
```

```
r_on
```

```
  when
```

```
     $r = 0$ 
```

```
     $a = 1$ 
```

```
  then
```

```
     $r := 1$ 
```

```
end
```

```
treat_push_stop_motor_button
```

```
  refines
```

```
    treat_stop_motor
```

```
  when
```

```
     $stop\_motor\_impulse = FALSE$ 
```

```
     $stop\_motor\_button = TRUE$ 
```

```
     $motor\_sensor = working$ 
```

```
     $motor\_actuator = working$ 
```

```
  then
```

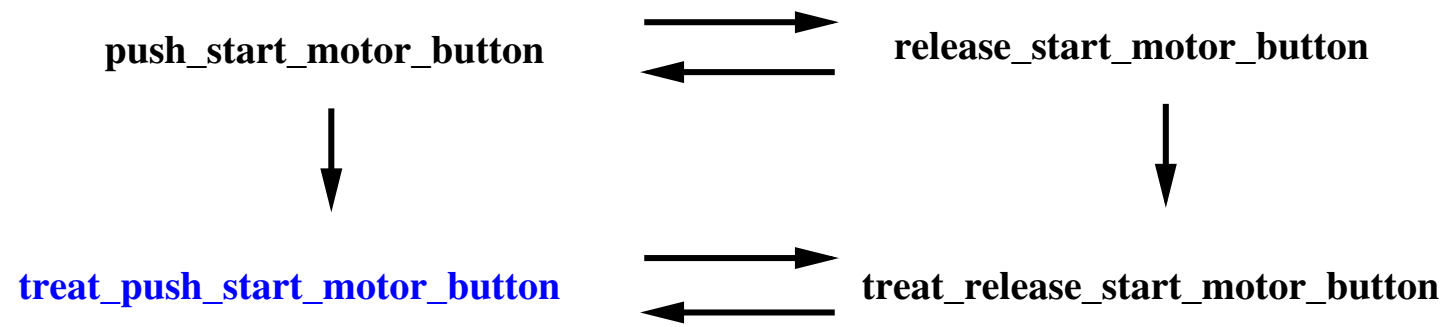
```
     $stop\_motor\_impulse := TRUE$ 
```

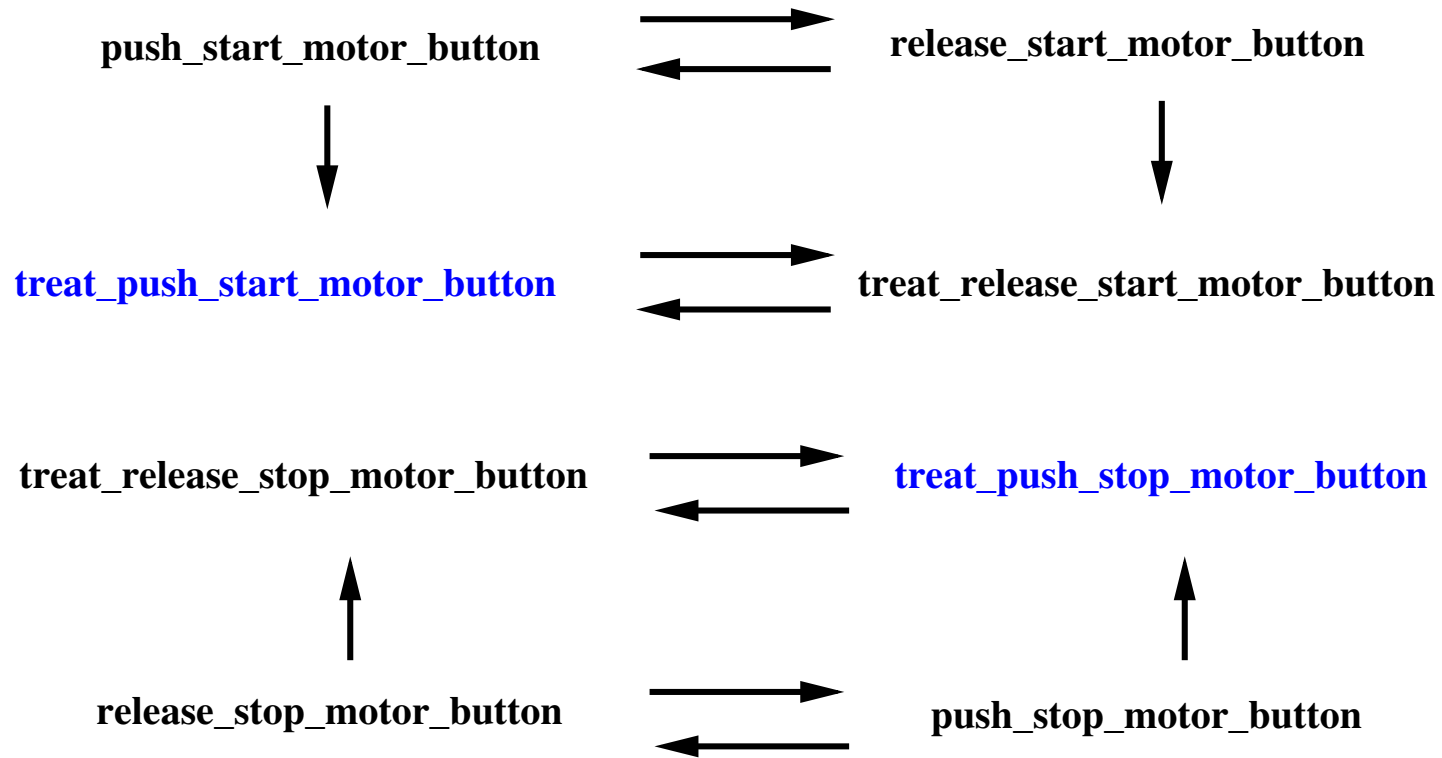
```
     $motor\_actuator := stopped$ 
```

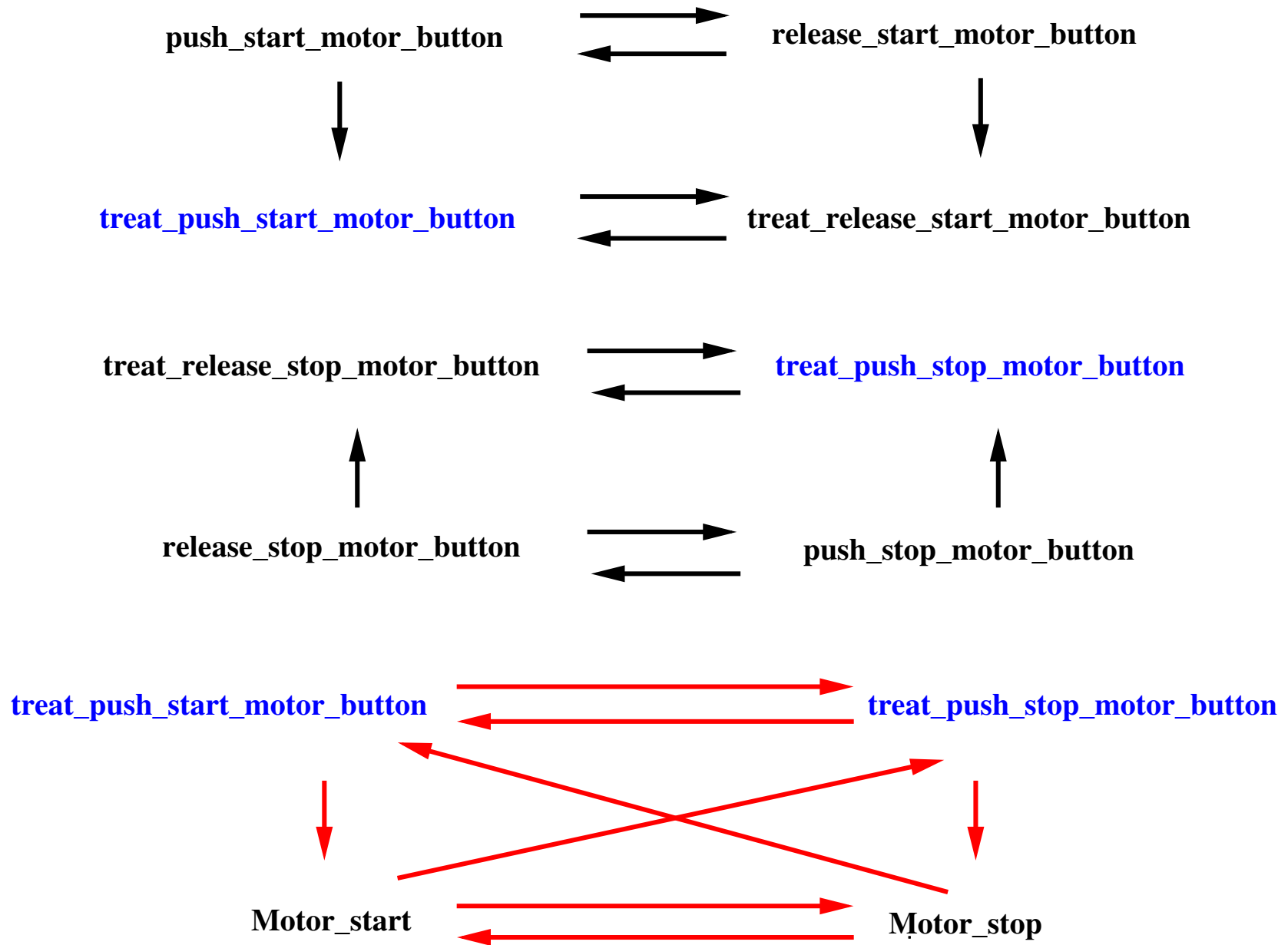
```
end
```

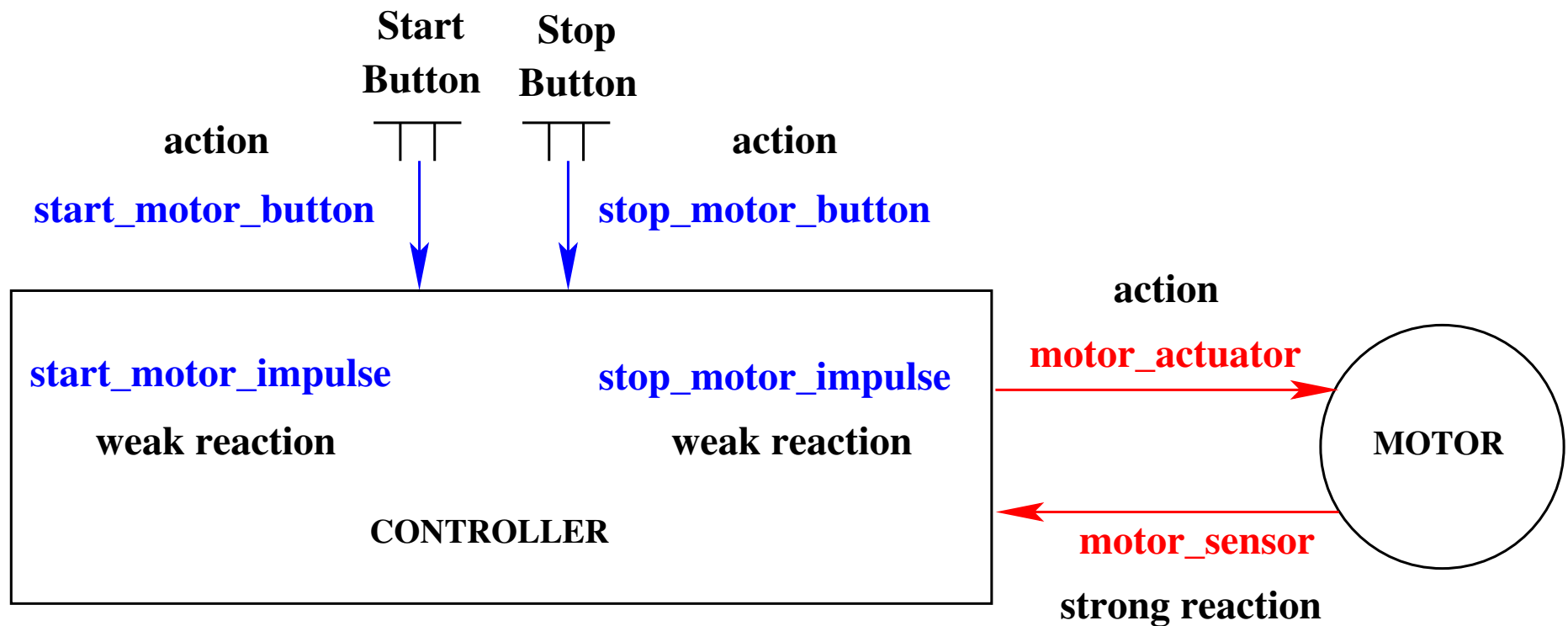
```
r_off
  when
     $r = 1$ 
     $a = 0$ 
  then
     $r := 0$ 
  end
```

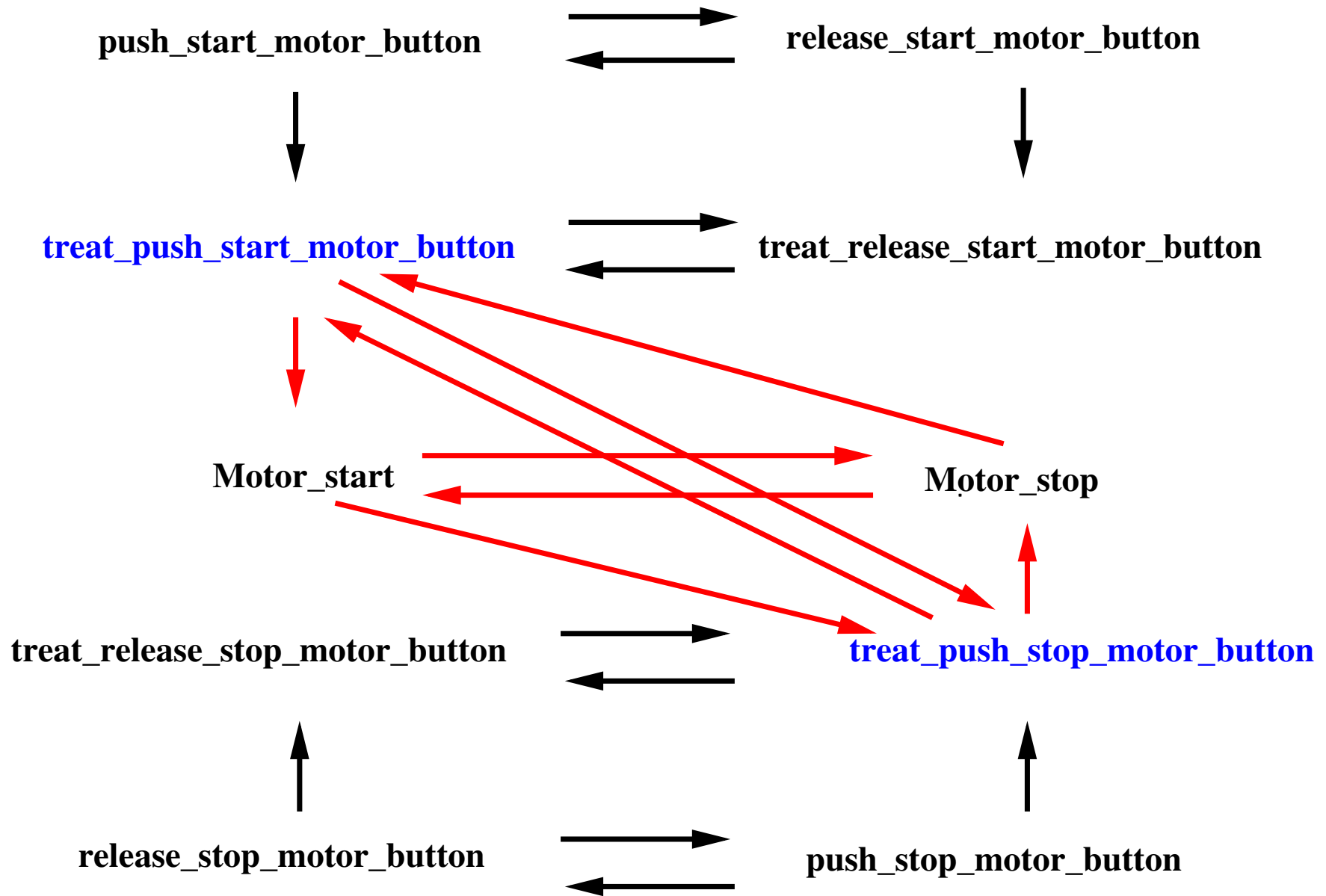
```
treat_release_stop_motor_button
  when
     $stop\_motor\_impulse = TRUE$ 
     $stop\_motor\_button = FALSE$ 
  then
     $stop\_motor\_impulse := FALSE$ 
  end
```











```
treat_push_start_motor_button
  refines
    treat_start_motor
  when
    start_motor_impulse = FALSE
    start_motor_button = TRUE
    motor_actuator = stopped
    motor_sensor = stopped
  then
    start_motor_impulse := TRUE
    motor_actuator := working
  end
```

- What happens when the following hold

$\neg (motor_actuator = stopped \wedge motor_sensor = stopped)$

- We need another event

```
treat_push_start_motor_button
  refines
    treat_start_motor
  when
    start_motor_impulse = FALSE
    start_motor_button = TRUE
    motor_actuator = stopped
    motor_sensor = stopped
  then
    start_motor_impulse := TRUE
    motor_actuator := working
  end
```

```
treat_push_start_motor_button_false

  when
    start_motor_impulse = FALSE
    start_motor_button = TRUE
     $\neg$  (motor_actuator = stopped  $\wedge$ 
        motor_sensor = stopped)
  then
    start_motor_impulse := 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 := TRUE

```
treat_push_stop_motor_button
  refines
    treat_stop_motor
  when
    stop_motor_impulse = FALSE
    stop_motor_button = TRUE
    motor_sensor = working
    motor_actuator = working
  then
    stop_motor_impulse := TRUE
    motor_actuator := stopped
  end
```

```
treat_push_stop_motor_button_false

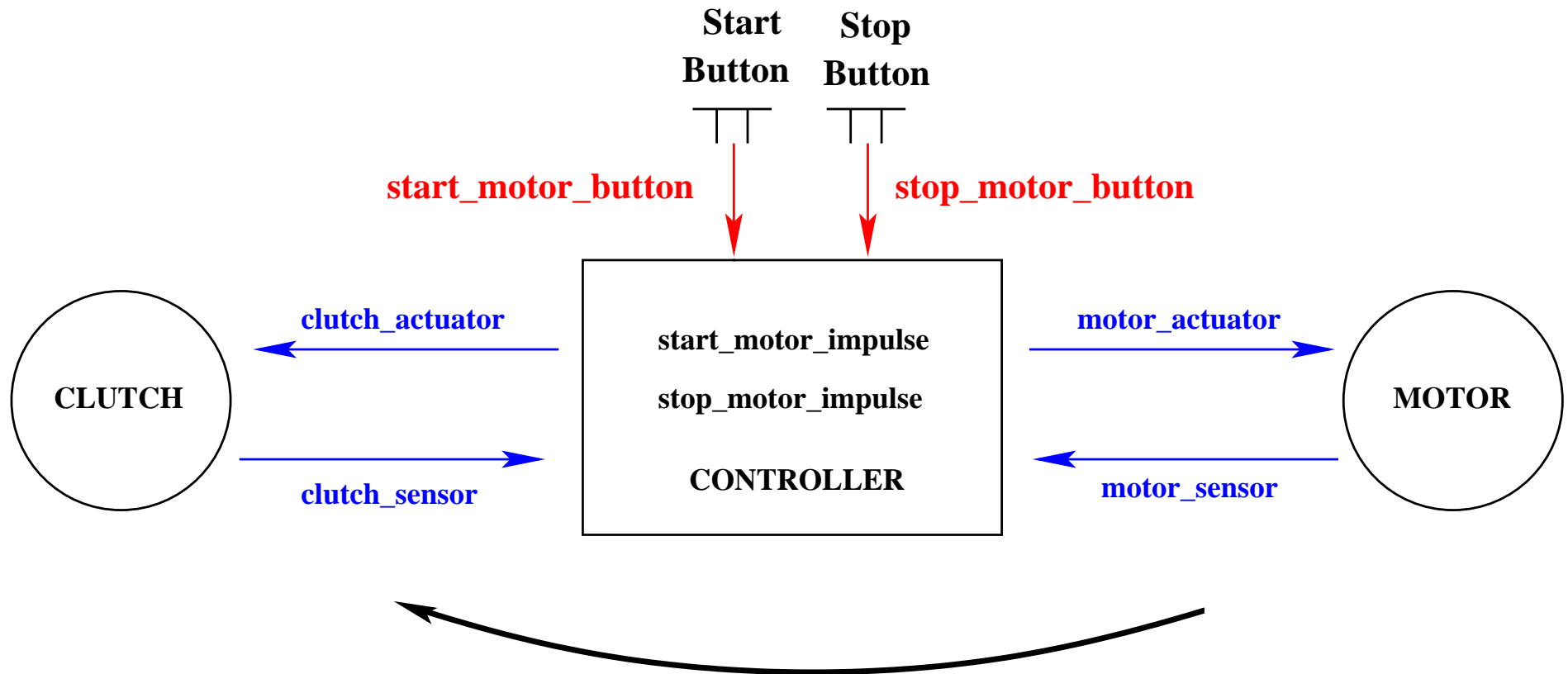
  when
    stop_motor_impulse = FALSE
    stop_motor_button = TRUE
     $\neg$  (motor_sensor = working  $\wedge$ 
        motor_actuator = working)
  then
    stop_motor_impulse := TRUE
  end
```

- 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:

stop_motor_impulse := TRUE

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

- 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**



- We introduce the set in a new context:

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

- We copy the initial model where we instantiate:

motor \rightsquigarrow *clutch*

STATUS \rightsquigarrow *CLUTCH*

working \rightsquigarrow *engaged*

stopped \rightsquigarrow *disengaged*

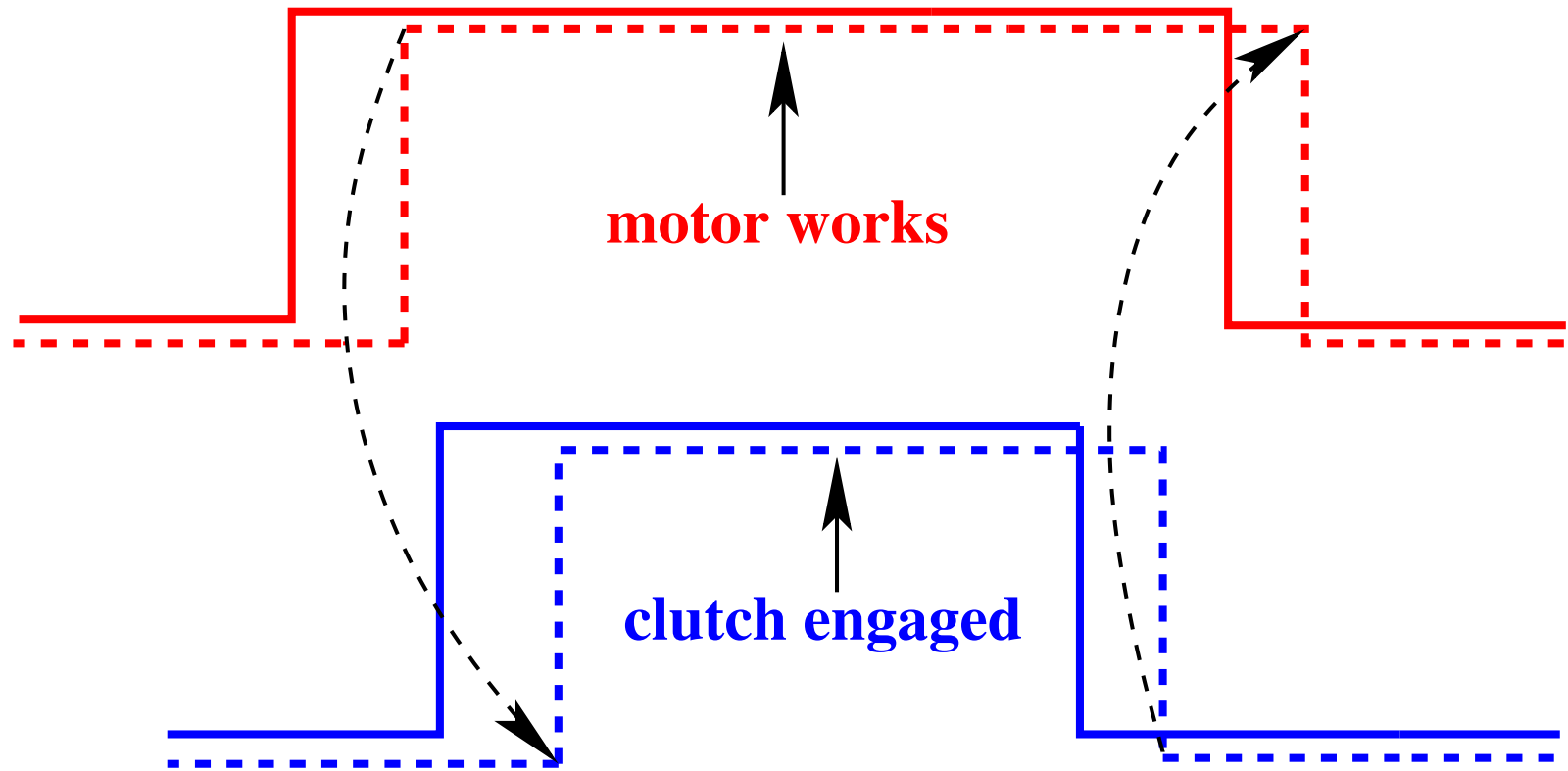
- Environment
 - motor_start
 - motor_stop
 - clutch_start
 - clutch_stop
 - push_start_motor_button
 - release_start_motor_button
 - push_stop_motor_button
 - release_stop_motor_button

- 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
---	-------

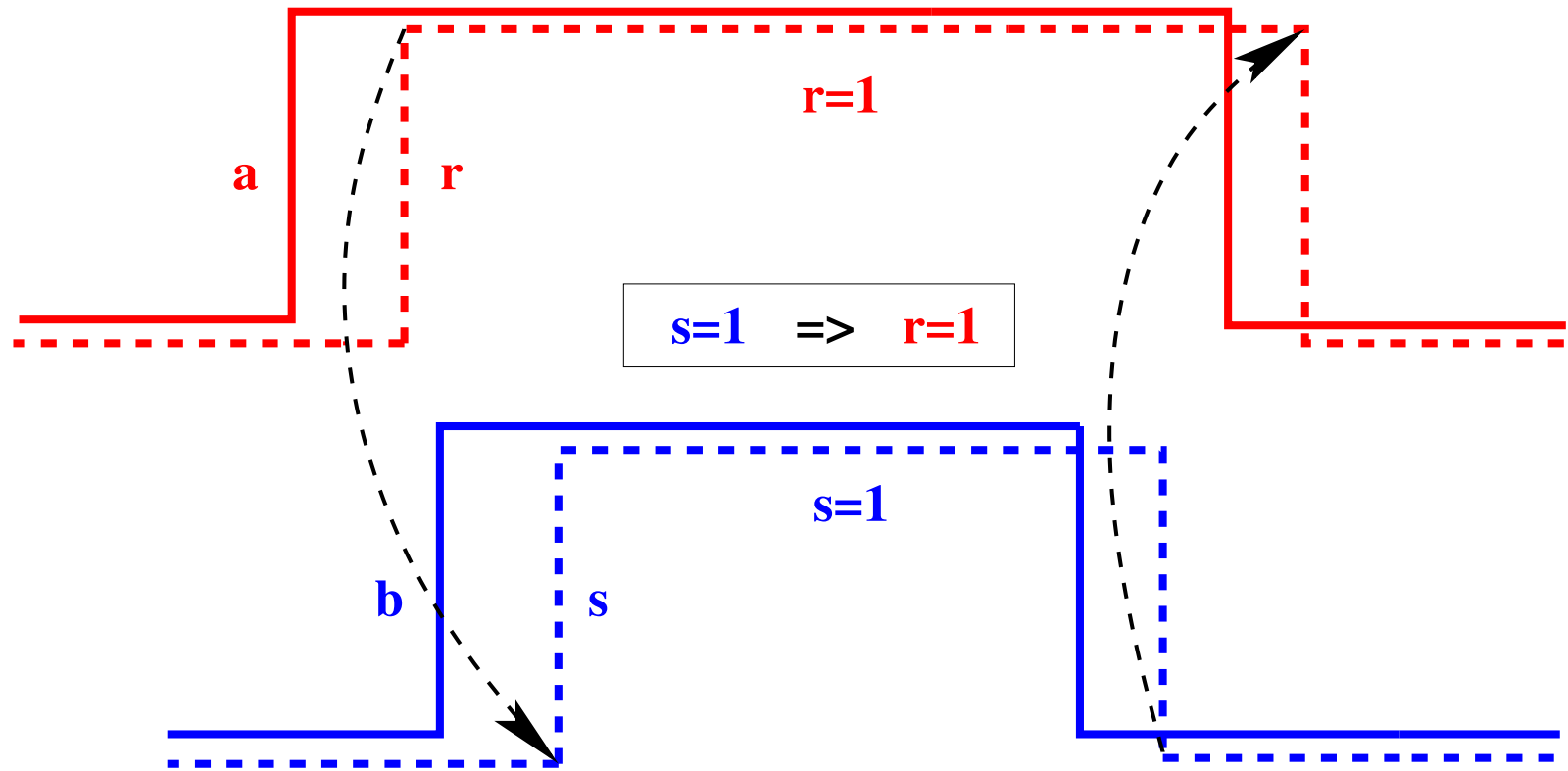
- For this we develop **ANOTHER DESIGN PATTERN**
- It is called: **Weak synchronization of two Strong Reactions**



When the clutch is engaged

then

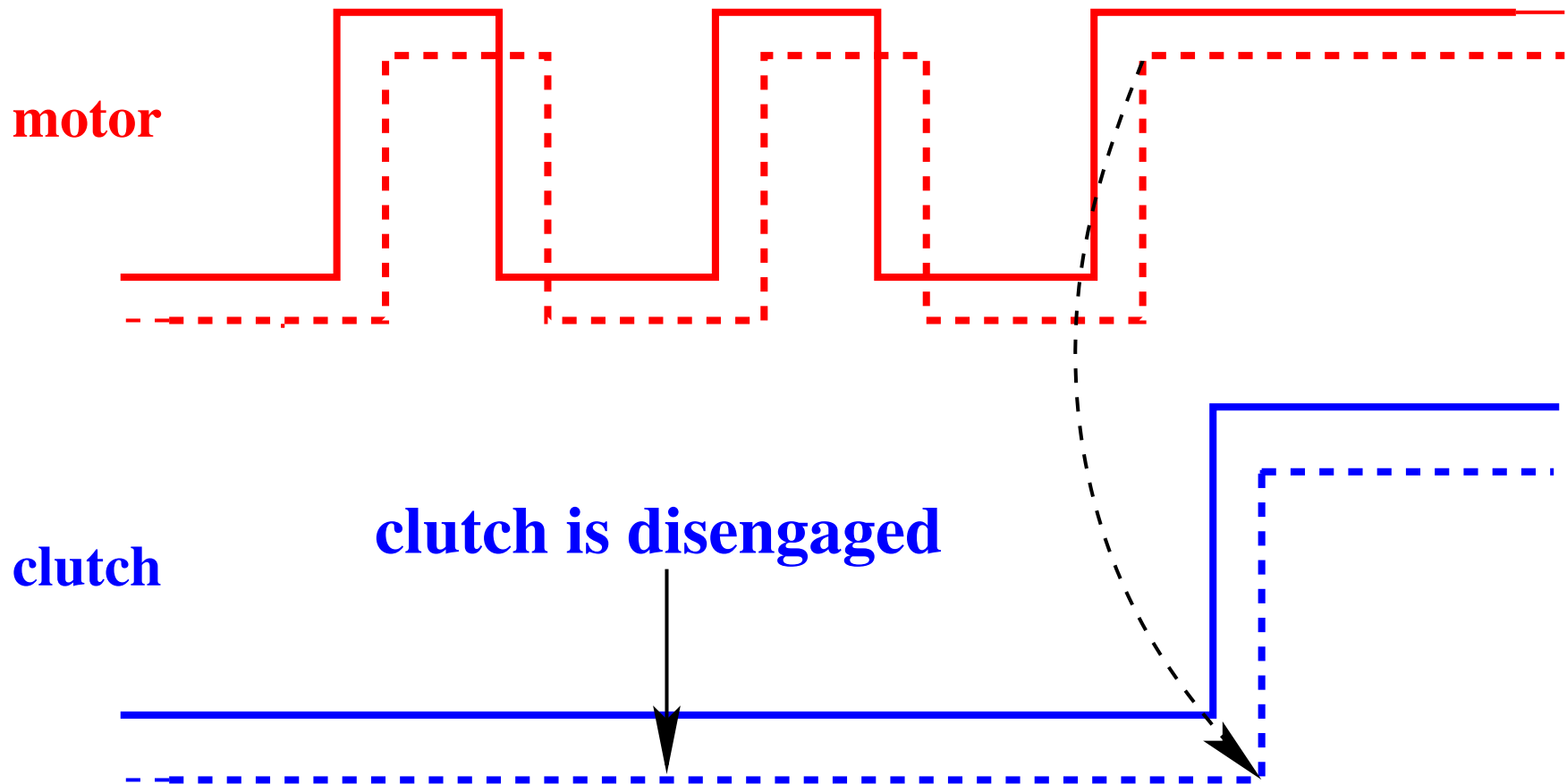
the motor must work



When the clutch is engaged

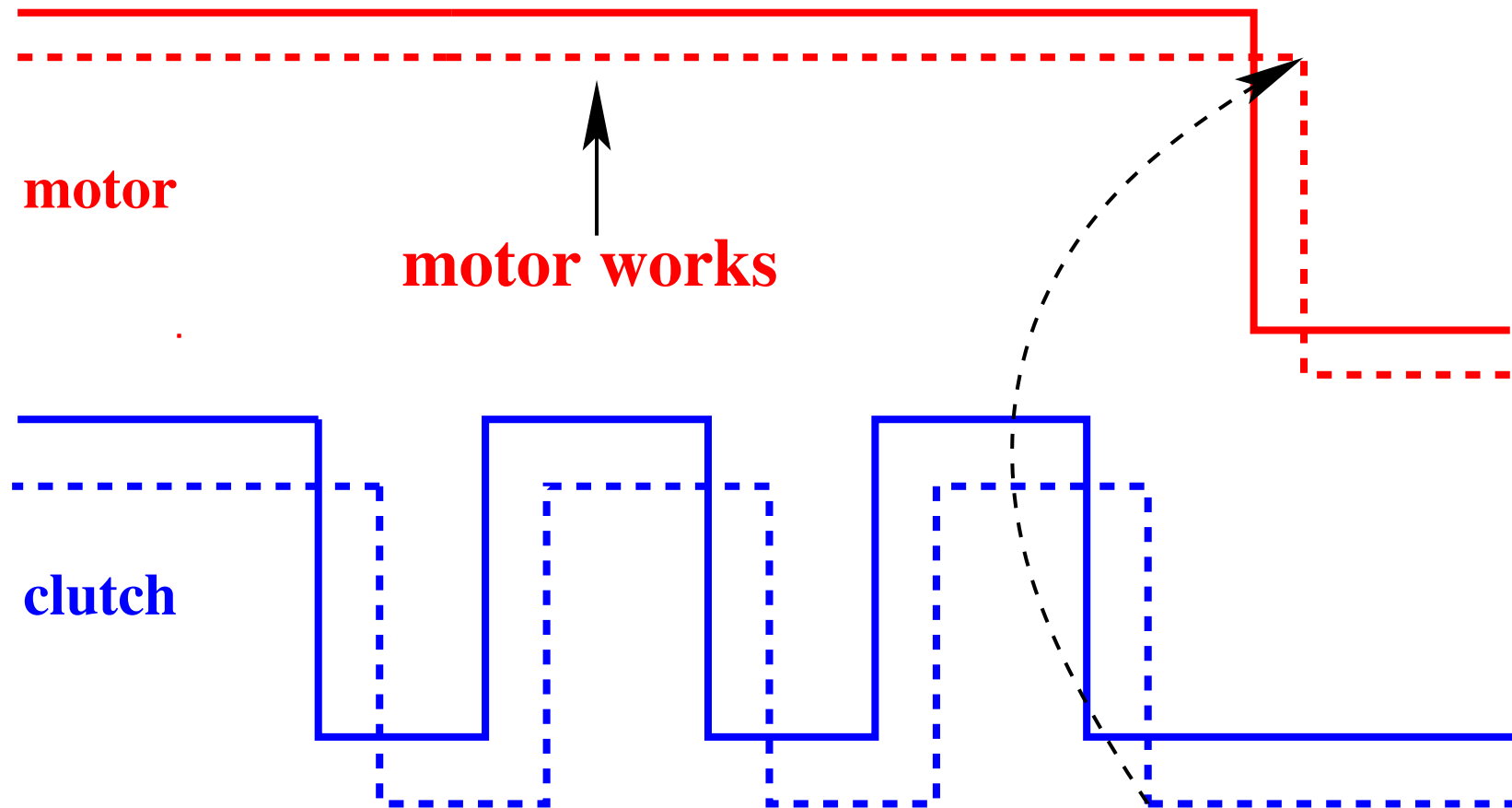
then

the motor must work



When the clutch is disengaged,
then

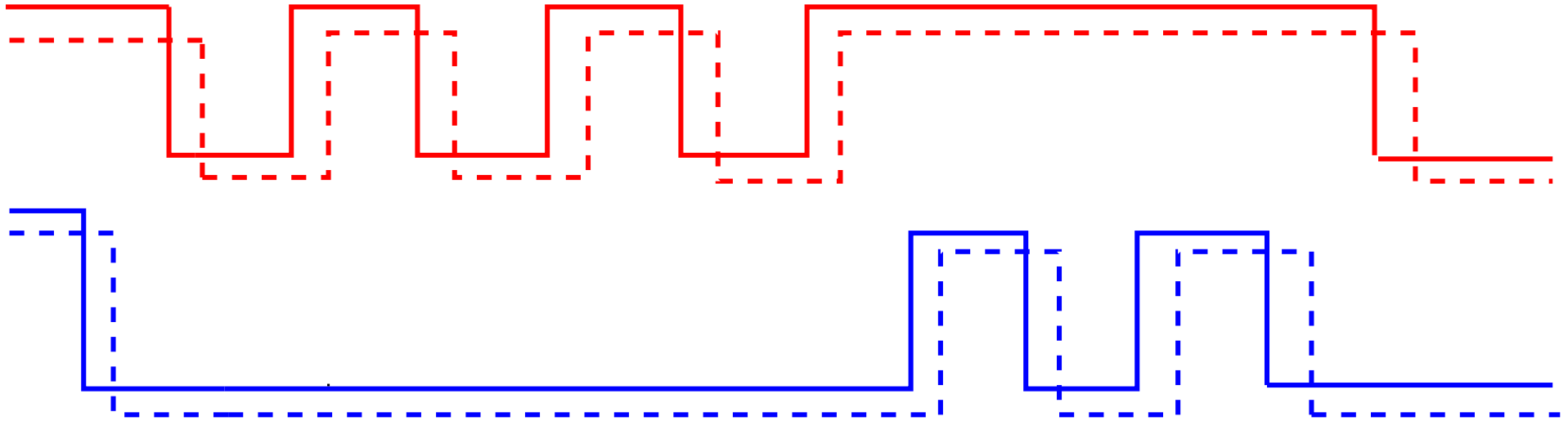
the motor can be started and stopped several times

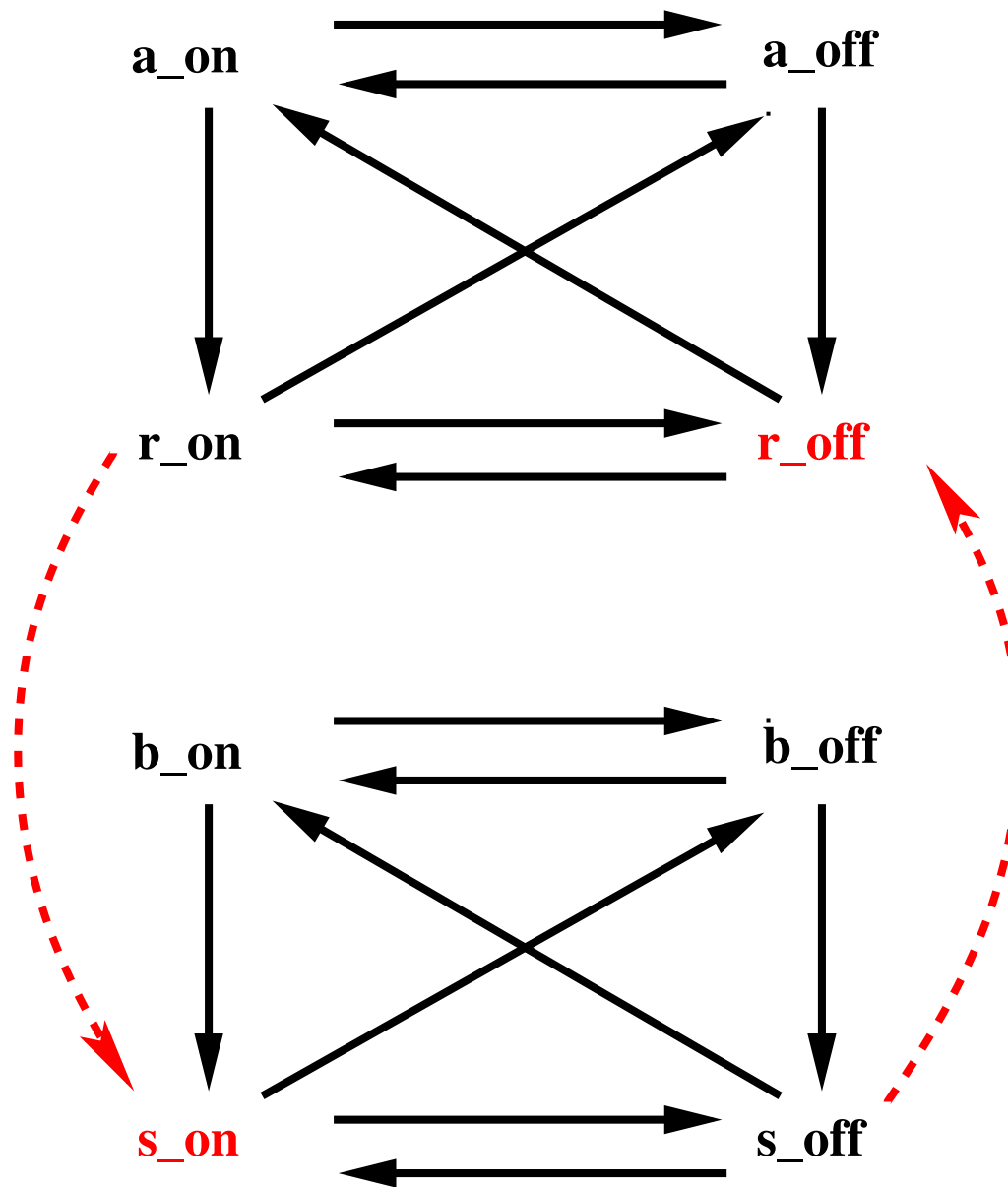


When the motor works,

then

the clutch can be engaged and disengaged several times





$$\text{dbl0_1: } a \in \{0, 1\}$$

$$\text{dbl0_2: } r \in \{0, 1\}$$

$$\text{dbl0_3: } ca \in \mathbb{N}$$

$$\text{dbl0_4: } cr \in \mathbb{N}$$

$$\text{dbl0_5: } a = 1 \wedge r = 0 \Rightarrow ca = cr + 1$$

$$\text{dbl0_6: } a = 0 \vee r = 1 \Rightarrow ca = cr$$

$$\text{dbl0_7: } b \in \{0, 1\}$$

$$\text{dbl0_8: } s \in \{0, 1\}$$

$$\text{dbl0_9: } cb \in \mathbb{N}$$

$$\text{dbl0_10: } cs \in \mathbb{N}$$

$$\text{dbl0_11: } b = 1 \wedge s = 0 \Rightarrow cb = cs + 1$$

$$\text{dbl0_12: } b = 0 \vee s = 1 \Rightarrow cb = cs$$

```
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
```

```
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
```

$$\text{dbl1_1: } s = 1 \Rightarrow r = 1$$

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

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
```

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

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: $a = 0 \Rightarrow b = 0$

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

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

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

\rightsquigarrow

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

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

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
    
```

\rightsquigarrow

```

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**

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

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

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

\rightsquigarrow

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

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

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
```

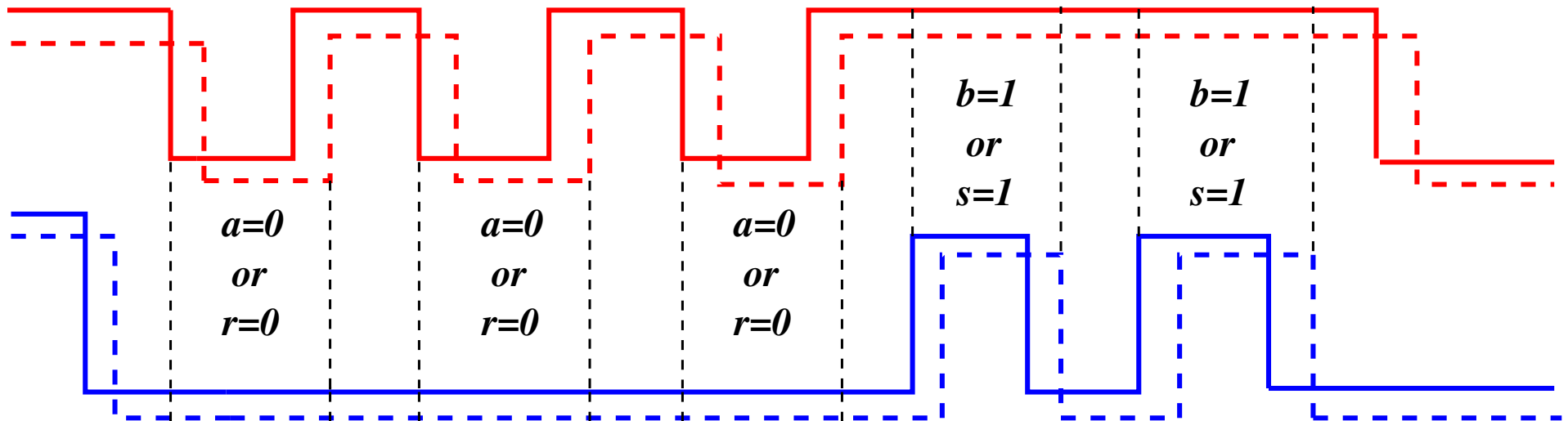
$$\begin{array}{l} \mathbf{dbl1_1:} \quad s = 1 \Rightarrow r = 1 \\ \mathbf{dbl1_2:} \quad b = 1 \Rightarrow r = 1 \\ \mathbf{dbl1_3:} \quad a = 0 \Rightarrow s = 0 \\ \mathbf{dbl1_4:} \quad a = 0 \Rightarrow b = 0 \end{array} \quad \begin{array}{l} (s = 1 \Rightarrow a = 1) \\ (b = 1 \Rightarrow a = 1) \end{array}$$

This can be put into a single invariant

$$\mathbf{dbl1_5:} \quad b = 1 \vee s = 1 \Rightarrow a = 1 \wedge r = 1$$

with the following contraposed form

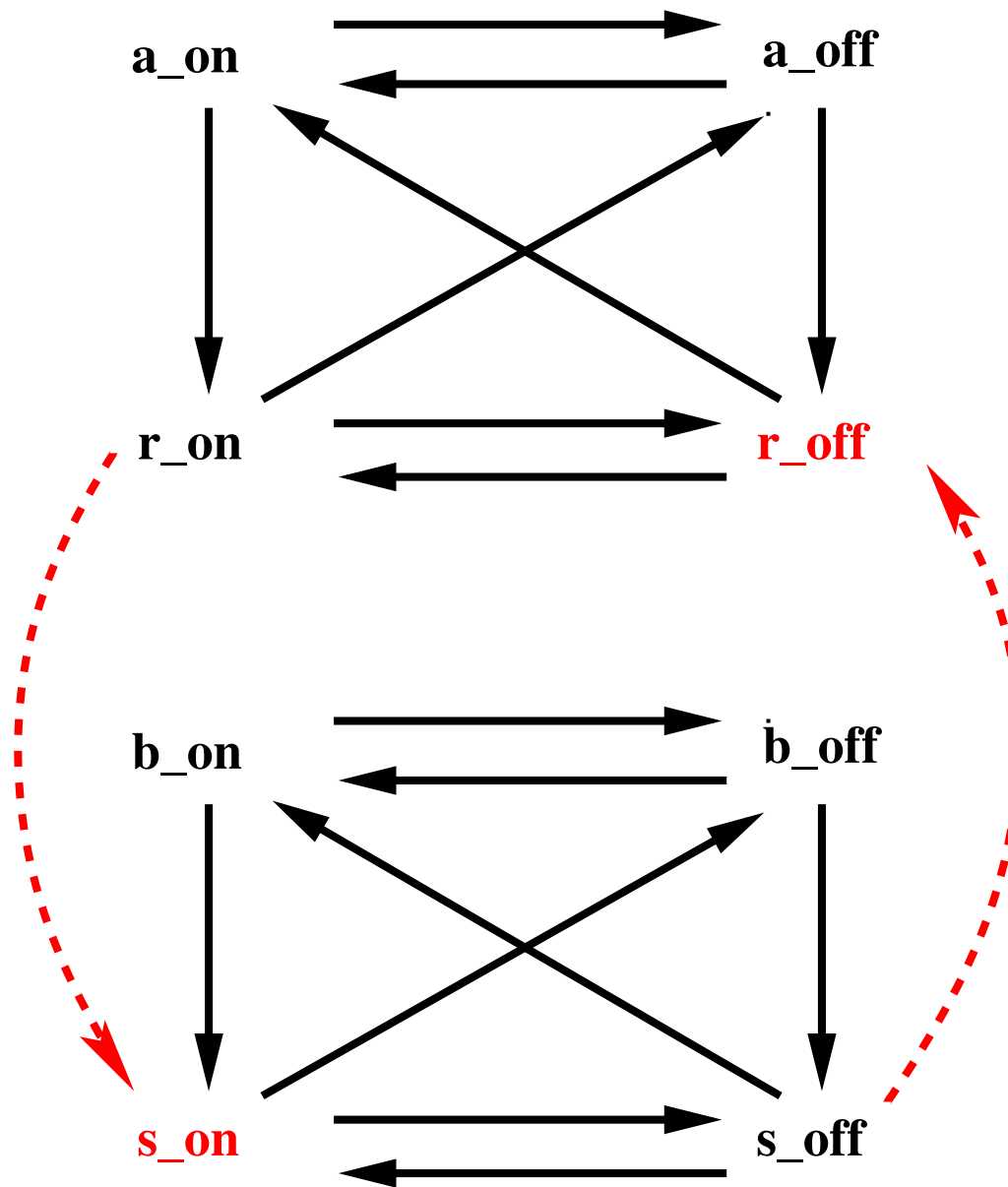
$$\mathbf{dbl1_6:} \quad a = 0 \vee r = 0 \Rightarrow b = 0 \wedge s = 0$$

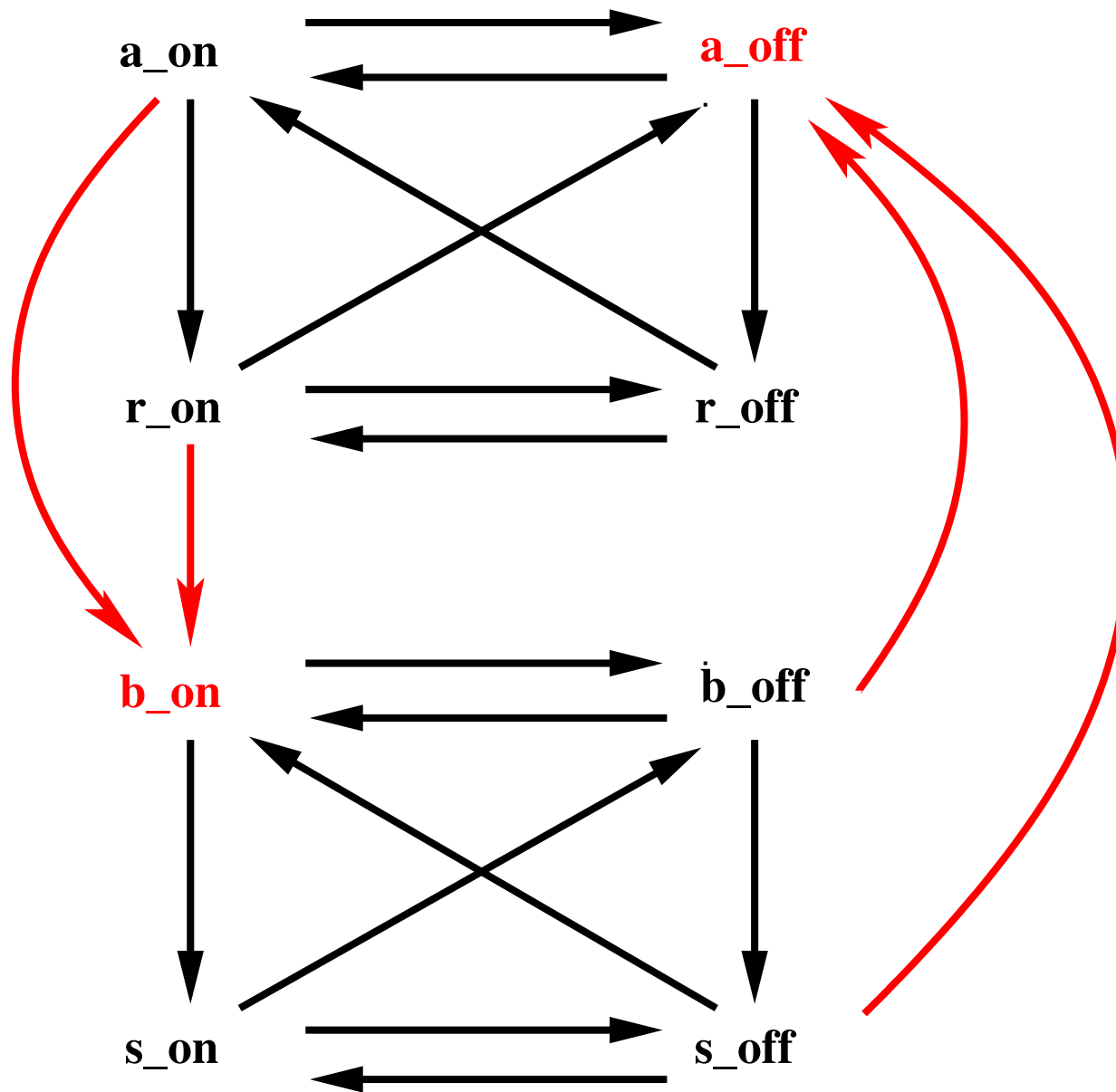


Reminder: - - - is the **motor** and - - - is the **clutch**

$$\mathbf{dbl1_5:} \quad b = 1 \vee s = 1 \Rightarrow a = 1 \wedge r = 1$$

$$\mathbf{dbl1_6:} \quad a = 0 \vee r = 0 \Rightarrow b = 0 \wedge s = 0$$





When the clutch is engaged, the motor must work	SAF_1
---	-------

inv3_1: <i>clutch_sensor = engaged</i> \Rightarrow <i>motor_sensor = working</i>

- This is an instance of the previous design pattern

- We **instantiate the pattern** as follows:

<i>a</i>	↔	<i>motor_actuator</i>	<i>a_on</i>	↔	<i>treat_push_start_motor_button</i>
<i>r</i>	↔	<i>motor_sensor</i>	<i>a_off</i>	↔	<i>treat_push_stop_motor_button</i>
<i>0</i>	↔	<i>stopped</i>	<i>r_on</i>	↔	<i>Motor_start</i>
<i>1</i>	↔	<i>working</i>	<i>r_off</i>	↔	<i>Motor_stop</i>

<i>b</i>	↔	<i>clutch_actuator</i>	<i>b_on</i>	↔	<i>treat_start_clutch</i>
<i>s</i>	↔	<i>clutch_sensor</i>	<i>b_off</i>	↔	<i>treat_stop_clutch</i>
<i>0</i>	↔	<i>disengaged</i>	<i>s_on</i>	↔	<i>Clutch_start</i>
<i>1</i>	↔	<i>engaged</i>	<i>s_off</i>	↔	<i>Clutch_stop</i>

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

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

inv3_1: \Rightarrow
 $clutch_sensor = engaged$
 $motor_sensor = working$

inv3_2: \Rightarrow
 $clutch_actuator = engaged$
 $motor_sensor = working$

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

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

inv3_3: $\begin{array}{l} \text{motor_actuator} = \text{stopped} \\ \Rightarrow \\ \text{clutch_sensor} = \text{disengaged} \end{array}$

inv3_4: $\begin{array}{l} \text{motor_actuator} = \text{stopped} \\ \Rightarrow \\ \text{clutch_actuator} = \text{disengaged} \end{array}$

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

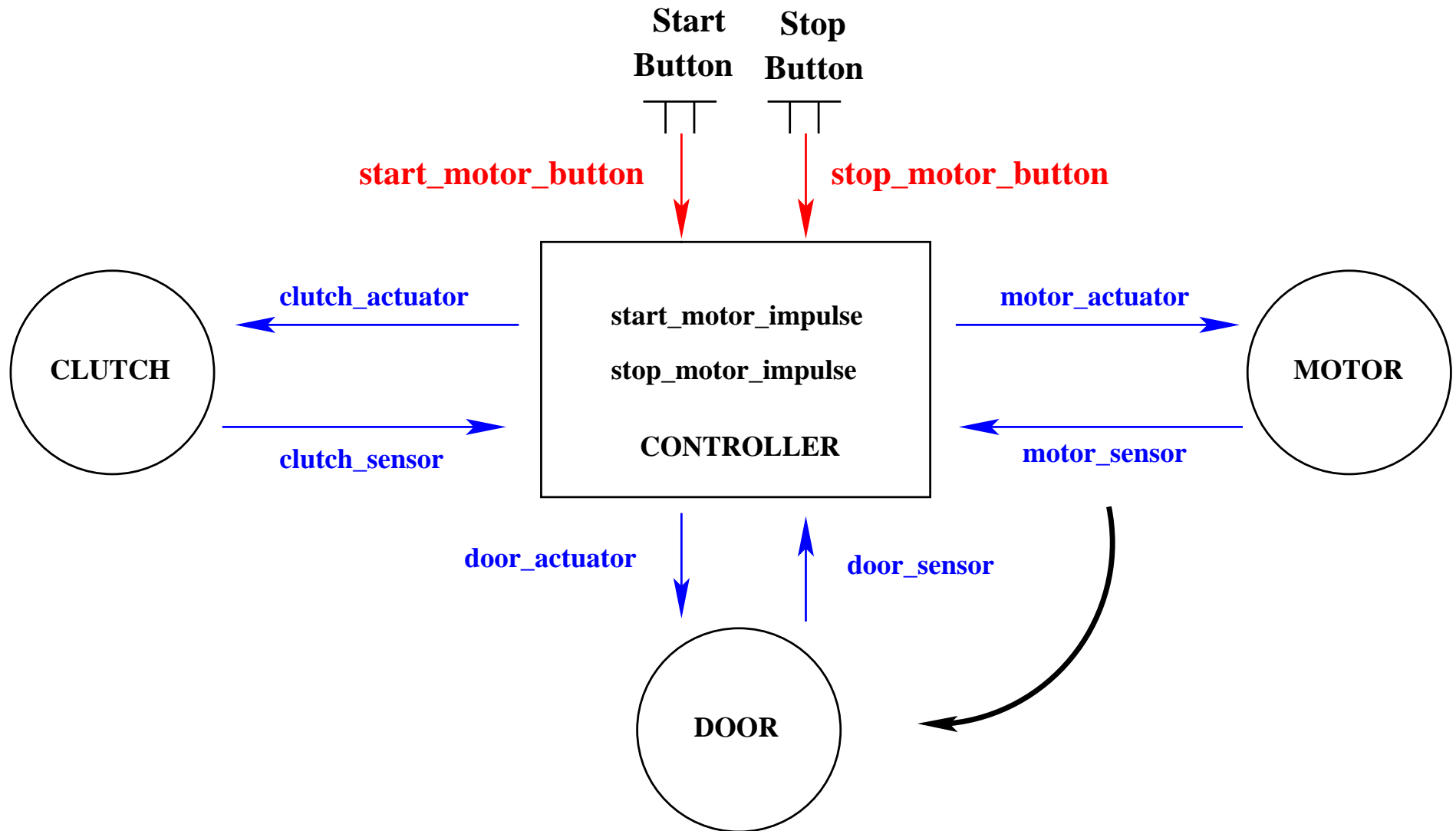
```
a_off
when

  a = 1
  r = 1
  s = 0
  b = 0
then
  a := 0
end
```

```
treat_push_stop_motor_button
when
  stop_motor_impulse = FALSE
  stop_motor_button = TRUE
  motor_actuator = working
  motor_sensor = working
  clutch_sensor = disengaged
  clutch_actuator = disengaged
then
  motor_actuator := stopped
  stop_motor_impulse := TRUE
end
```

- 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



- 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:

motor \rightsquigarrow *door*

STATUS \rightsquigarrow *DOOR*

working \rightsquigarrow *closed*

stopped \rightsquigarrow *open*

- 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

- 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?

- 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

- 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
--	-------

- 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
---	-------

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: $SAF_2 \wedge SAF_3' \Rightarrow SAF_1$

- 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

- 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

- 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

- 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

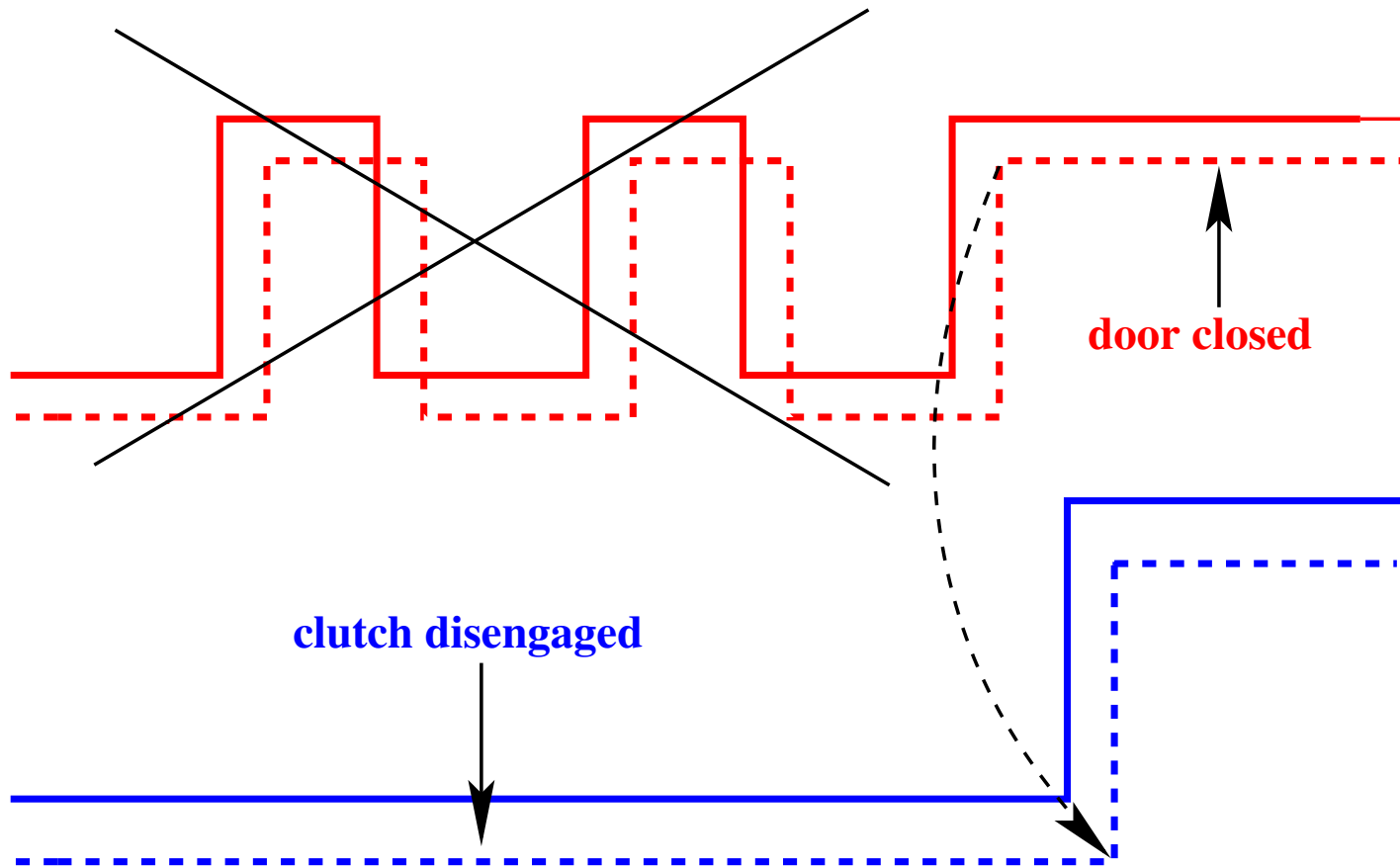
- Adding two **functional constraints**

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

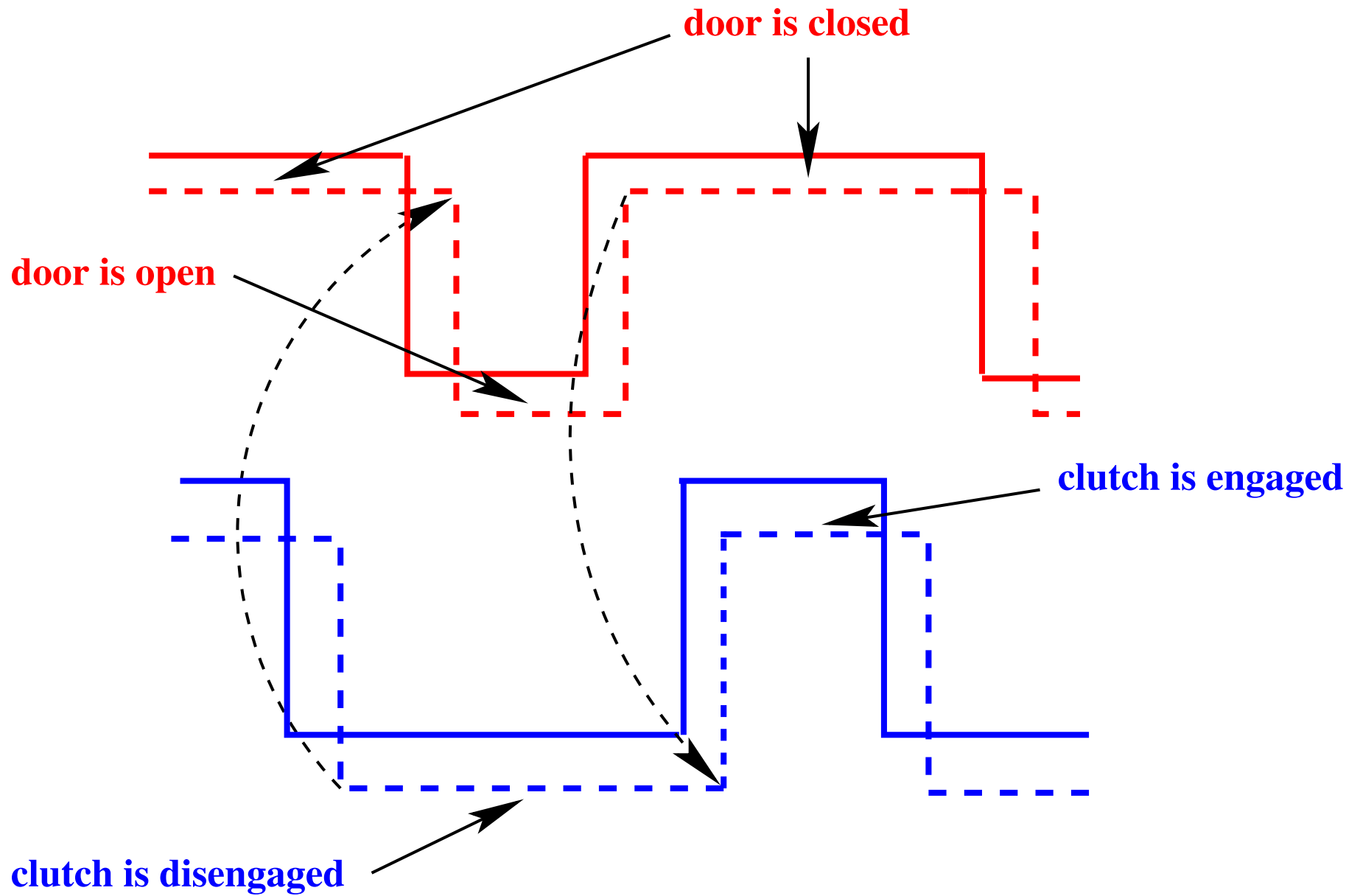
When the door is closed, the clutch cannot be disengaged several times, ONLY ONCE	FUN_4
---	-------

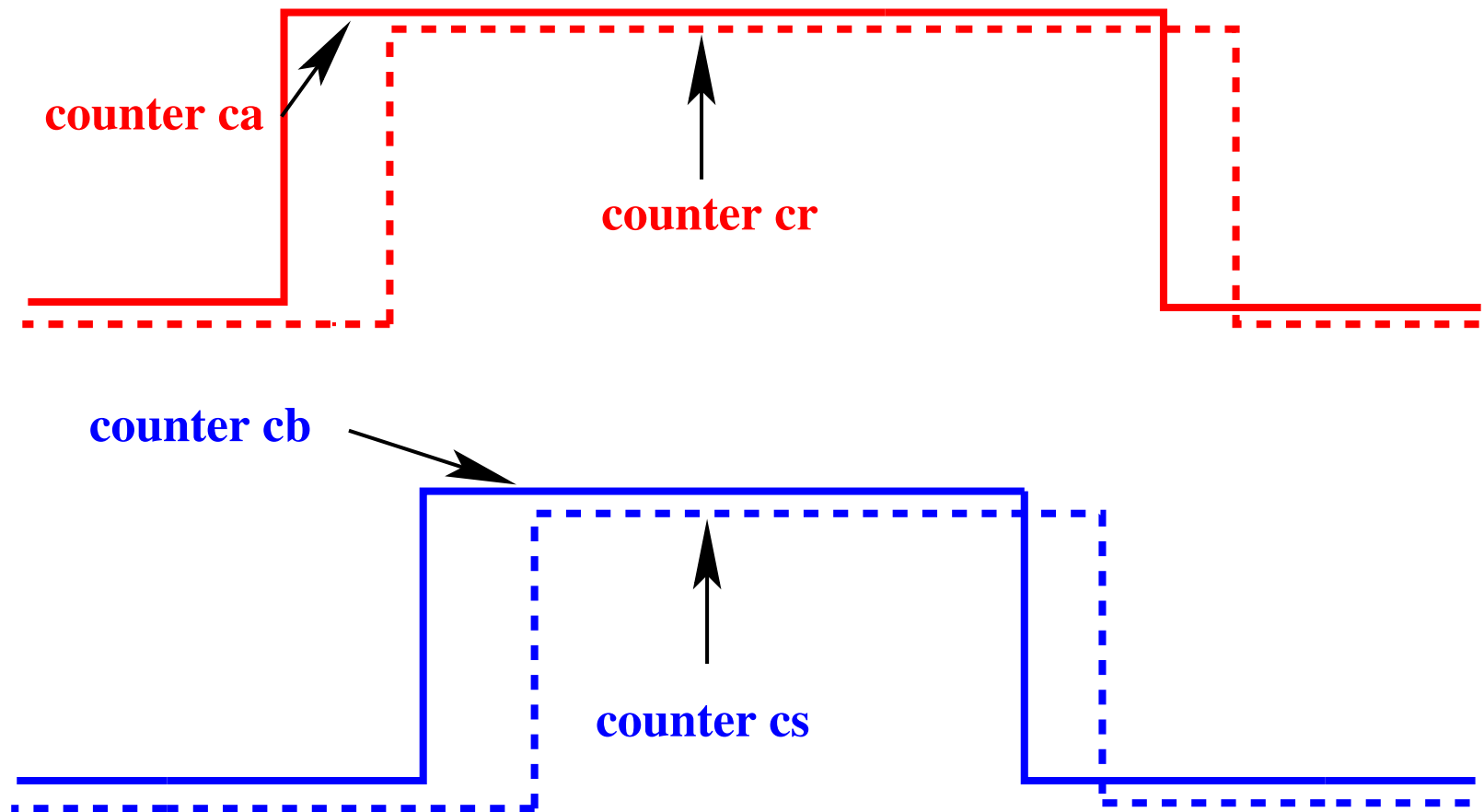
Problem with the **Weak** Synchronization of **Strong** Reactions

249



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

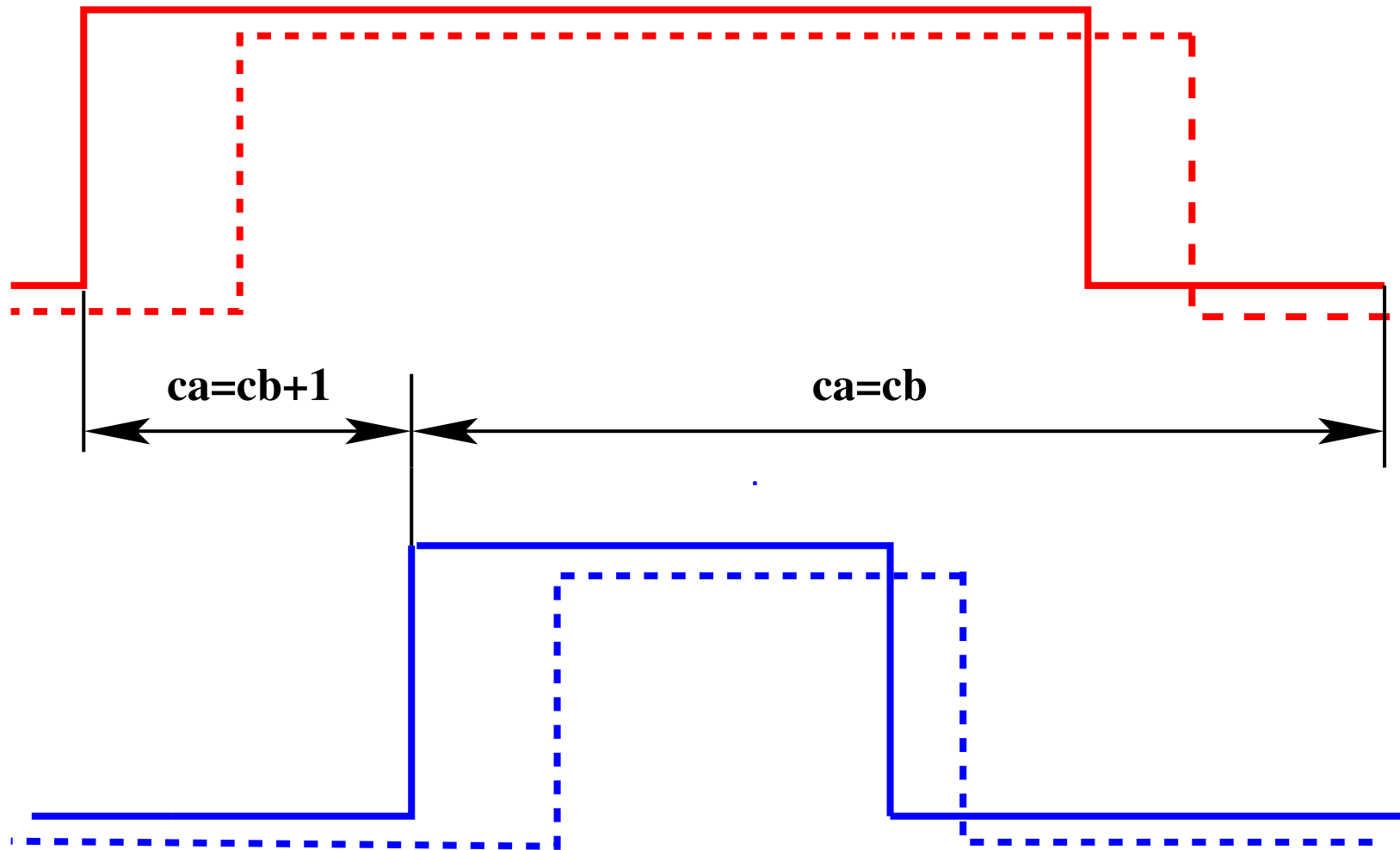


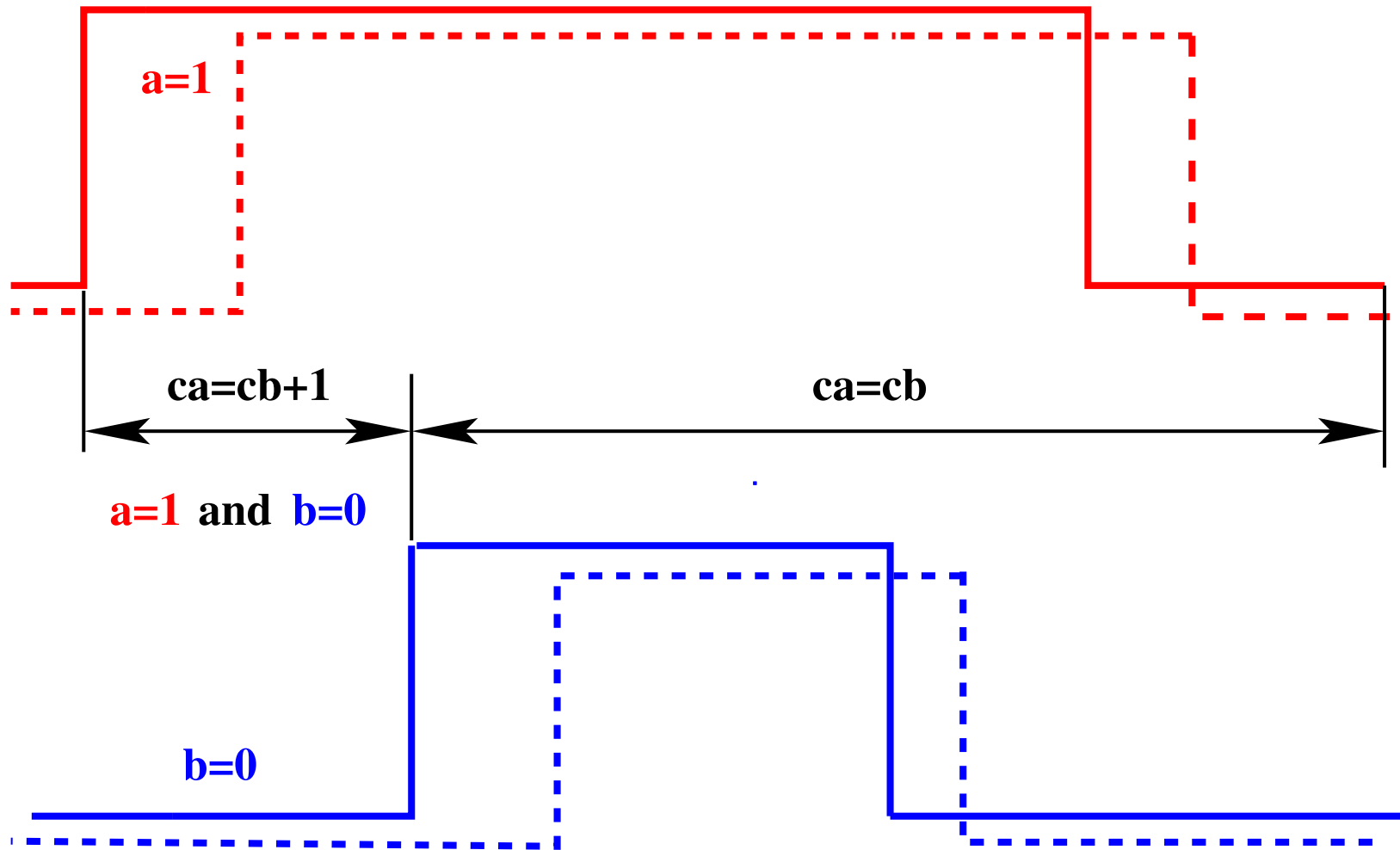


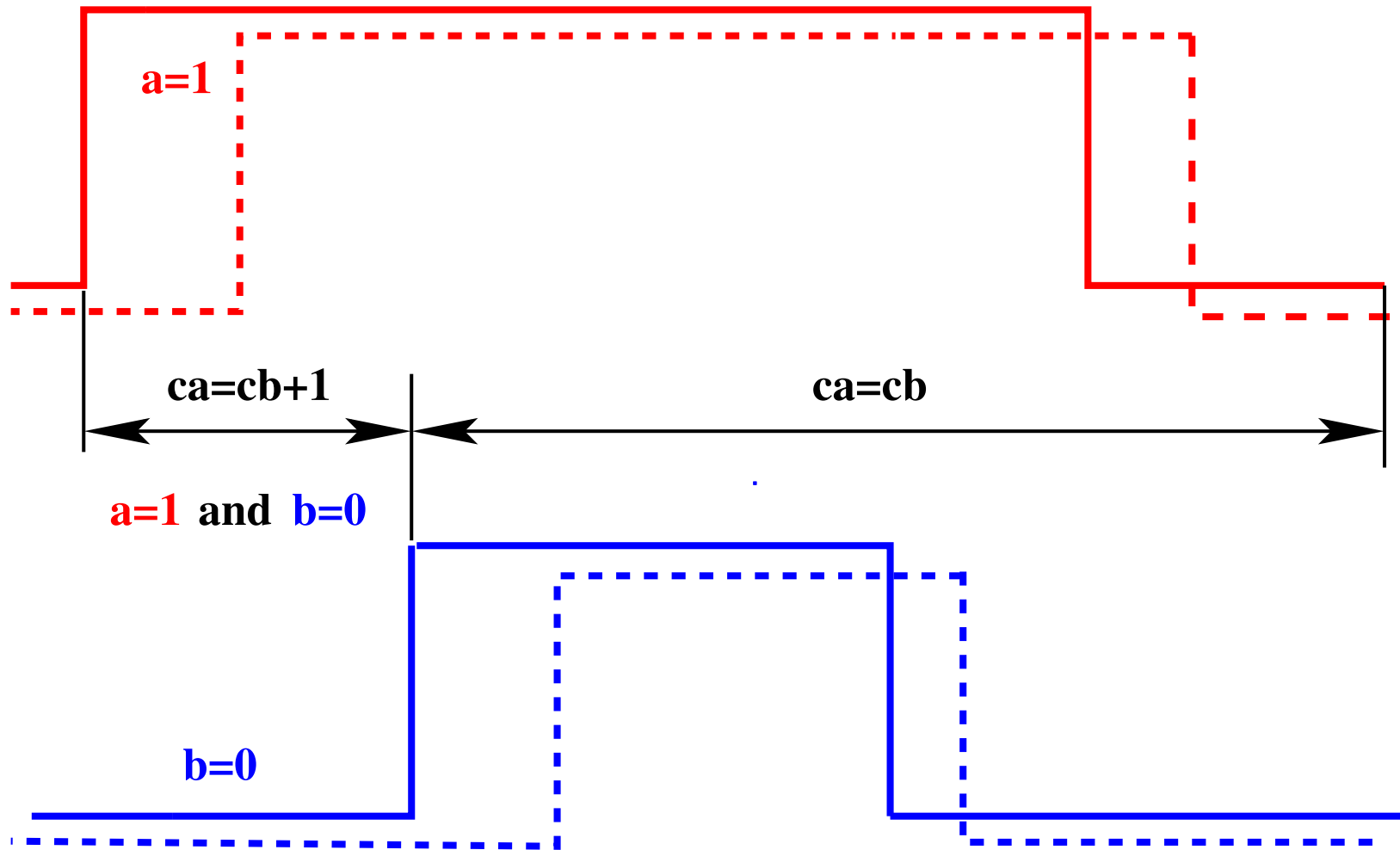
What we want:

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

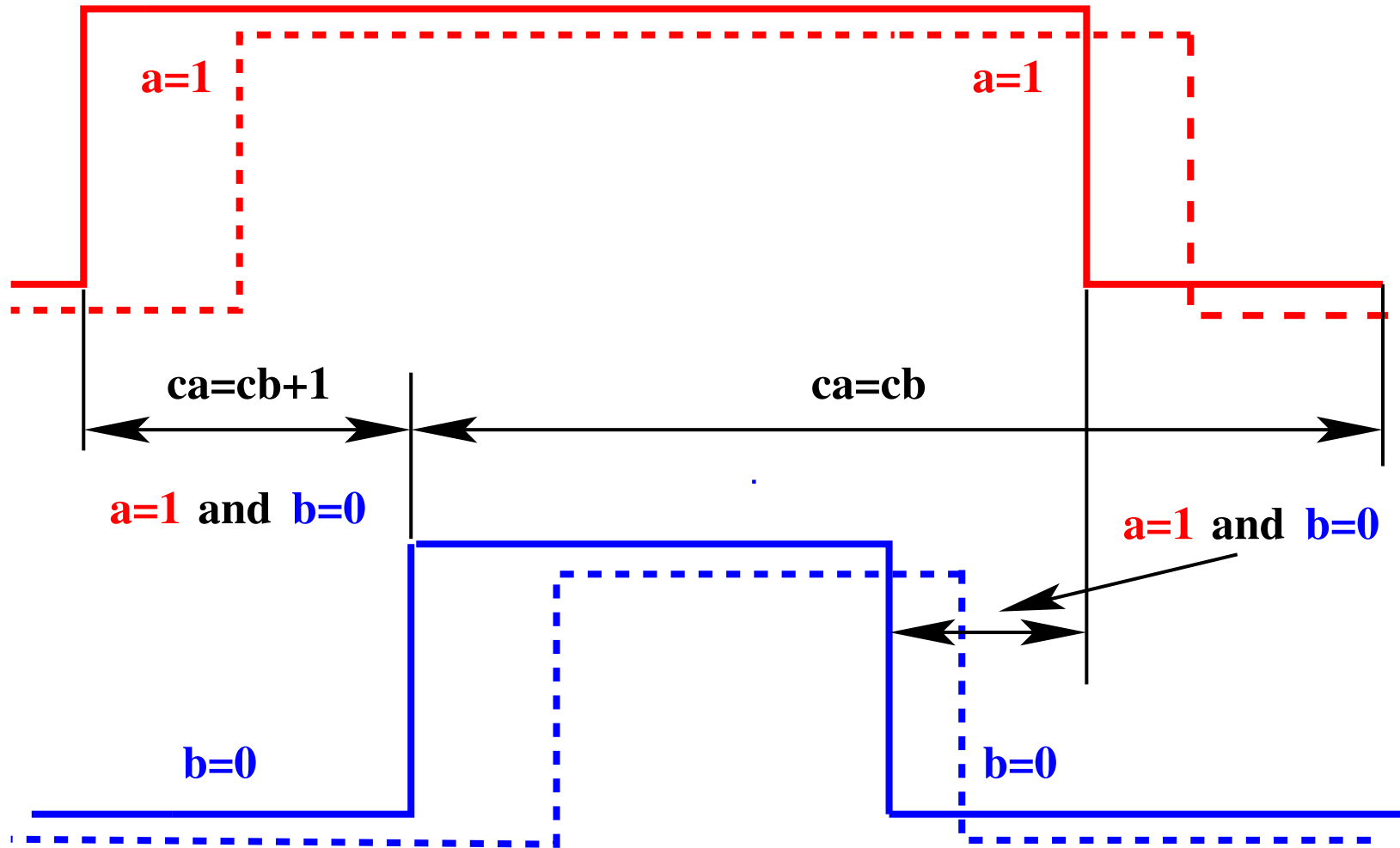
$$cr = cs \vee cr = cs + 1$$

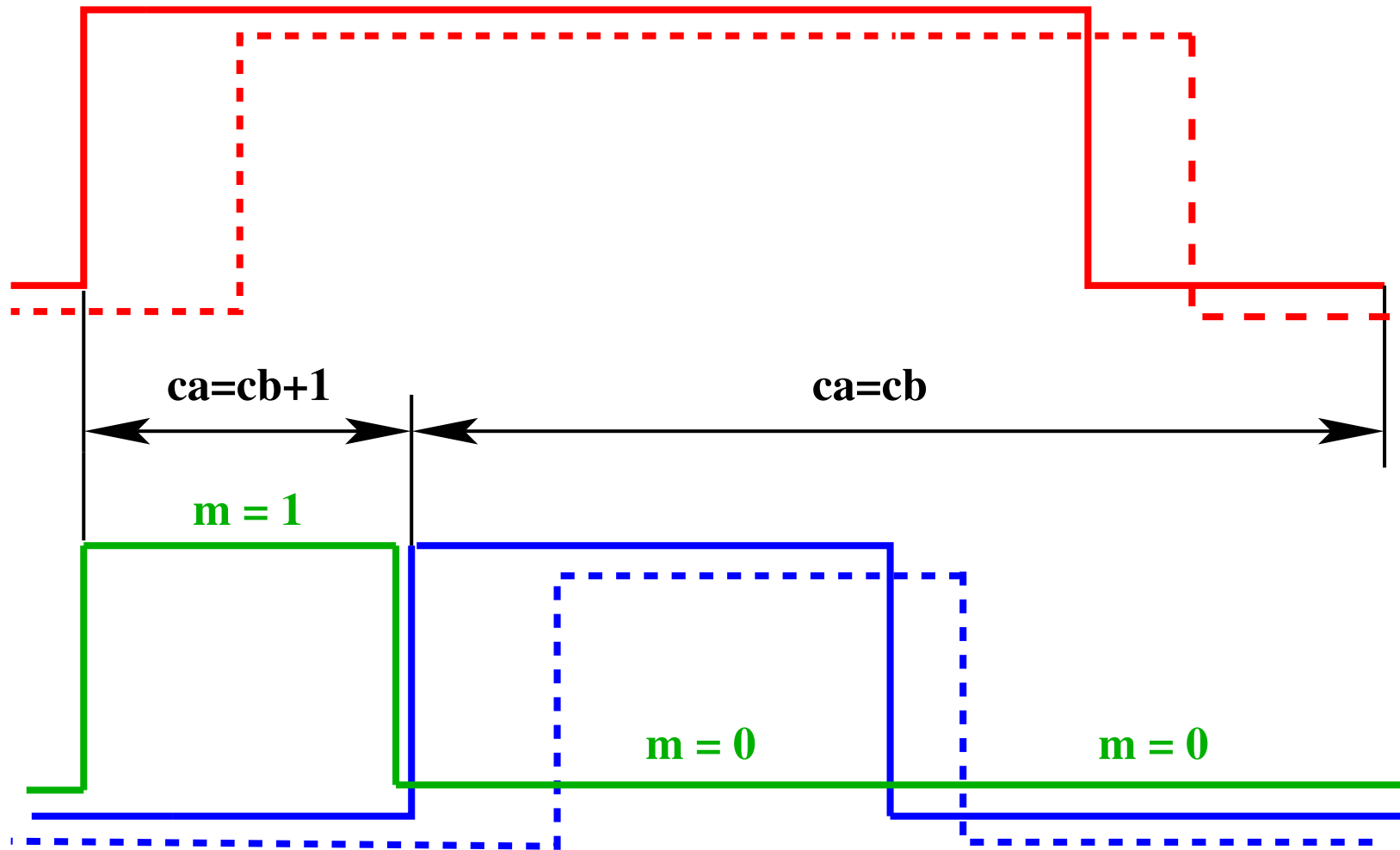




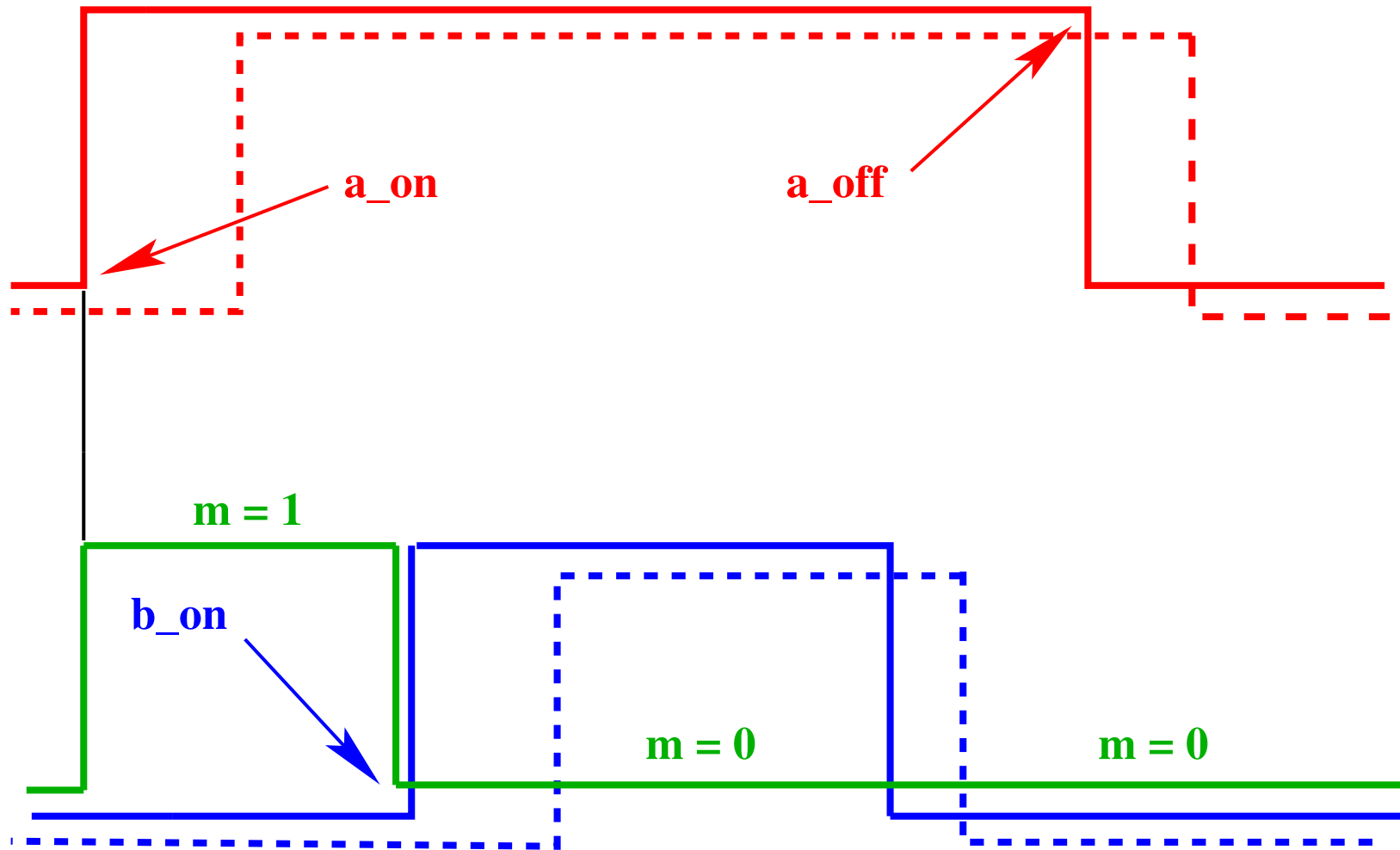


$$a = 1 \wedge b = 0 \Rightarrow ca = cb + 1 \quad ?$$





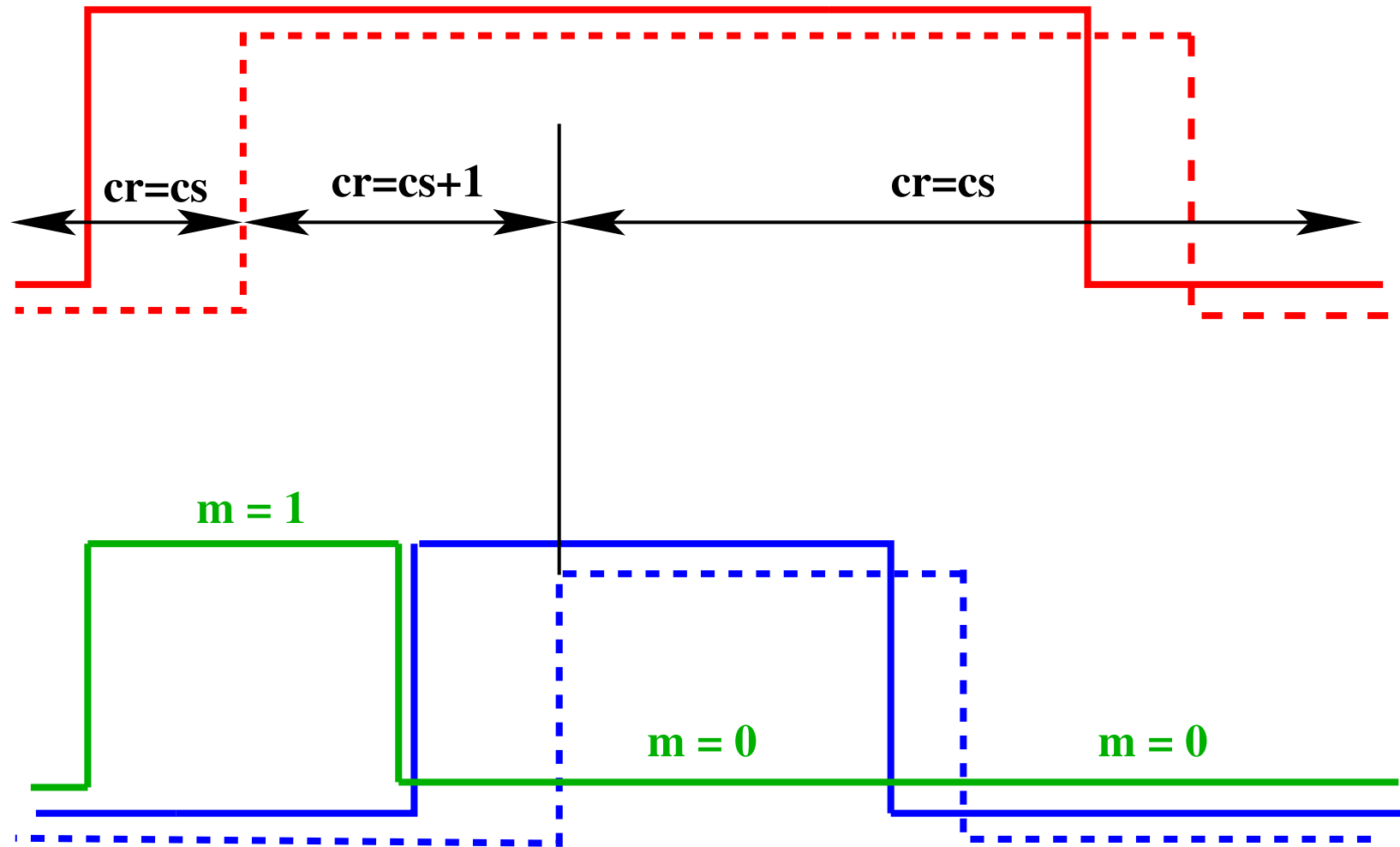
$$\begin{aligned}
 m = 1 &\Rightarrow ca = cb + 1 \\
 m = 0 &\Rightarrow ca = cb
 \end{aligned}$$

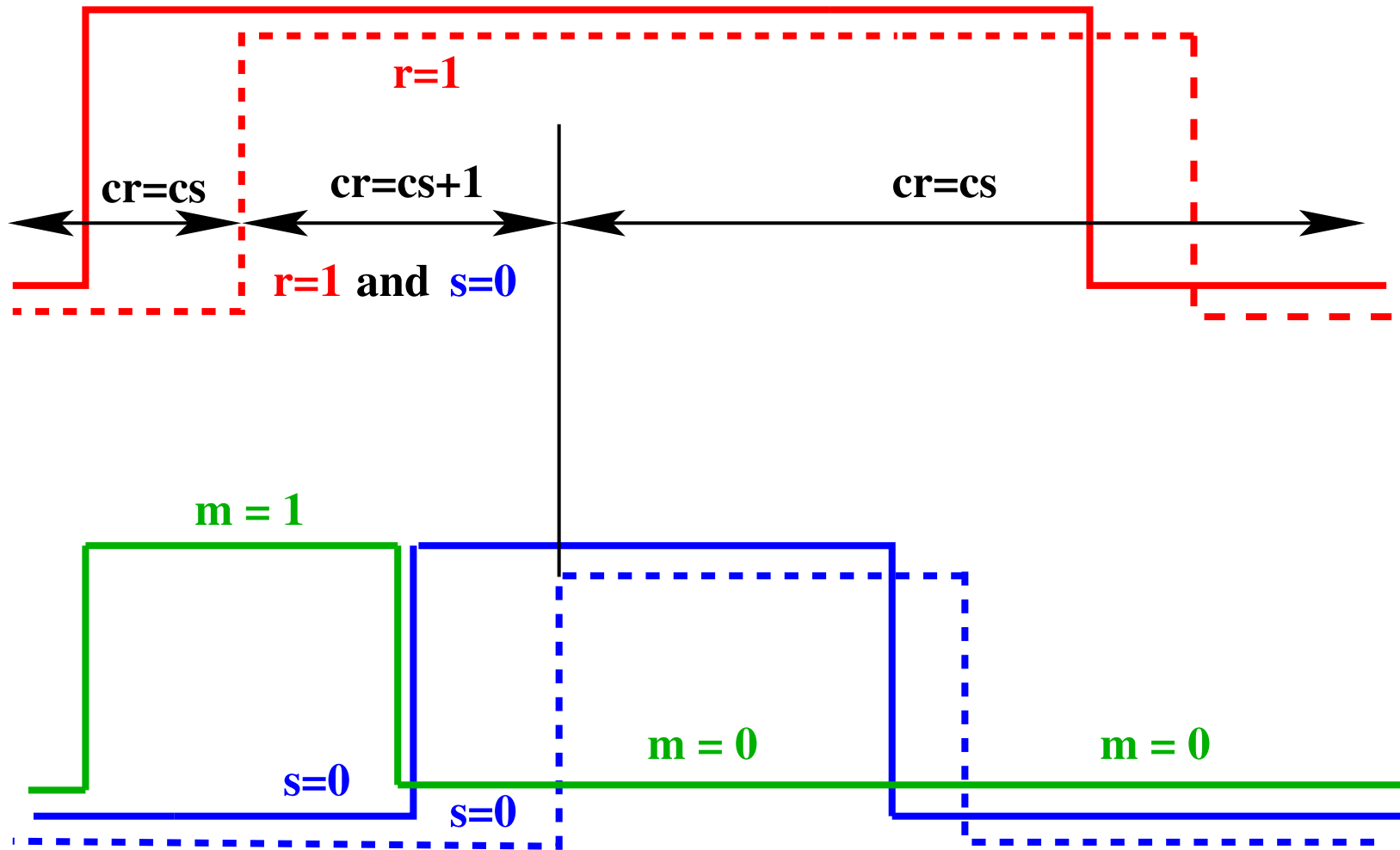


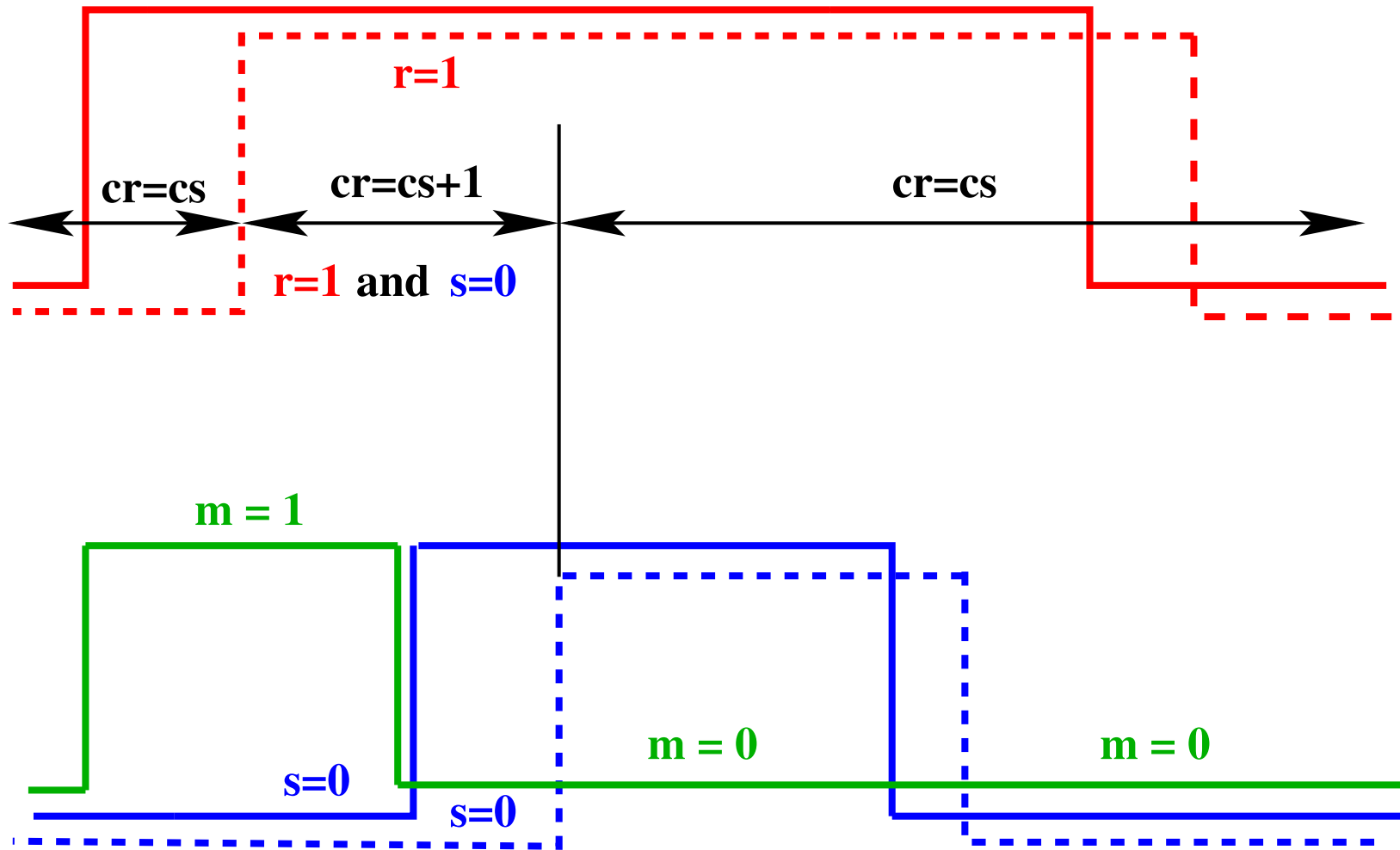
```
a_on
when
   $a = 0$ 
   $r = 0$ 
then
   $a := 1$ 
   $ca := ca + 1$ 
   $m := 1$ 
end
```

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

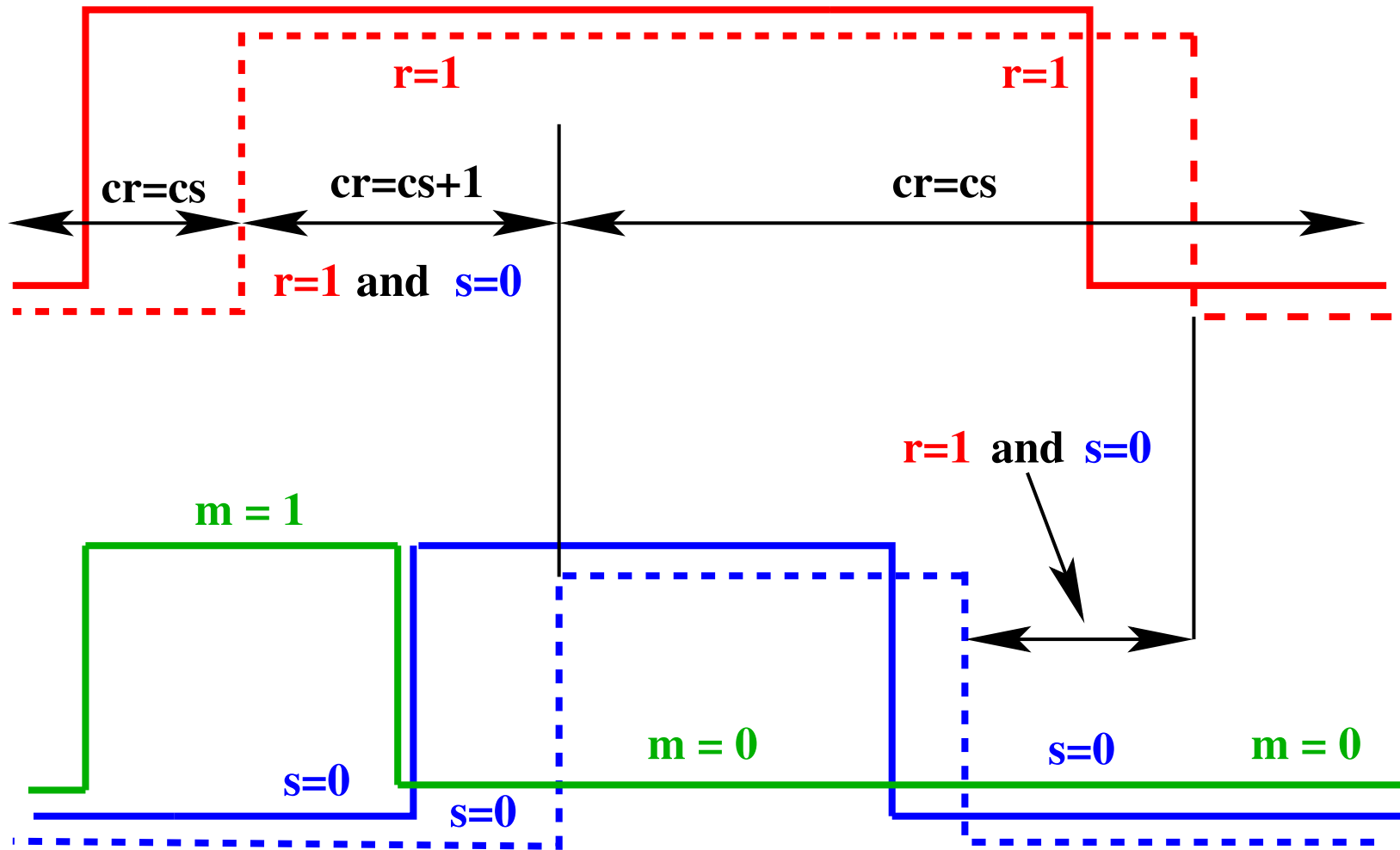
```
a_off
when
   $a = 1$ 
   $r = 1$ 
   $b = 0$ 
   $s = 0$ 
   $m = 0$ 
then
   $a := 0$ 
end
```

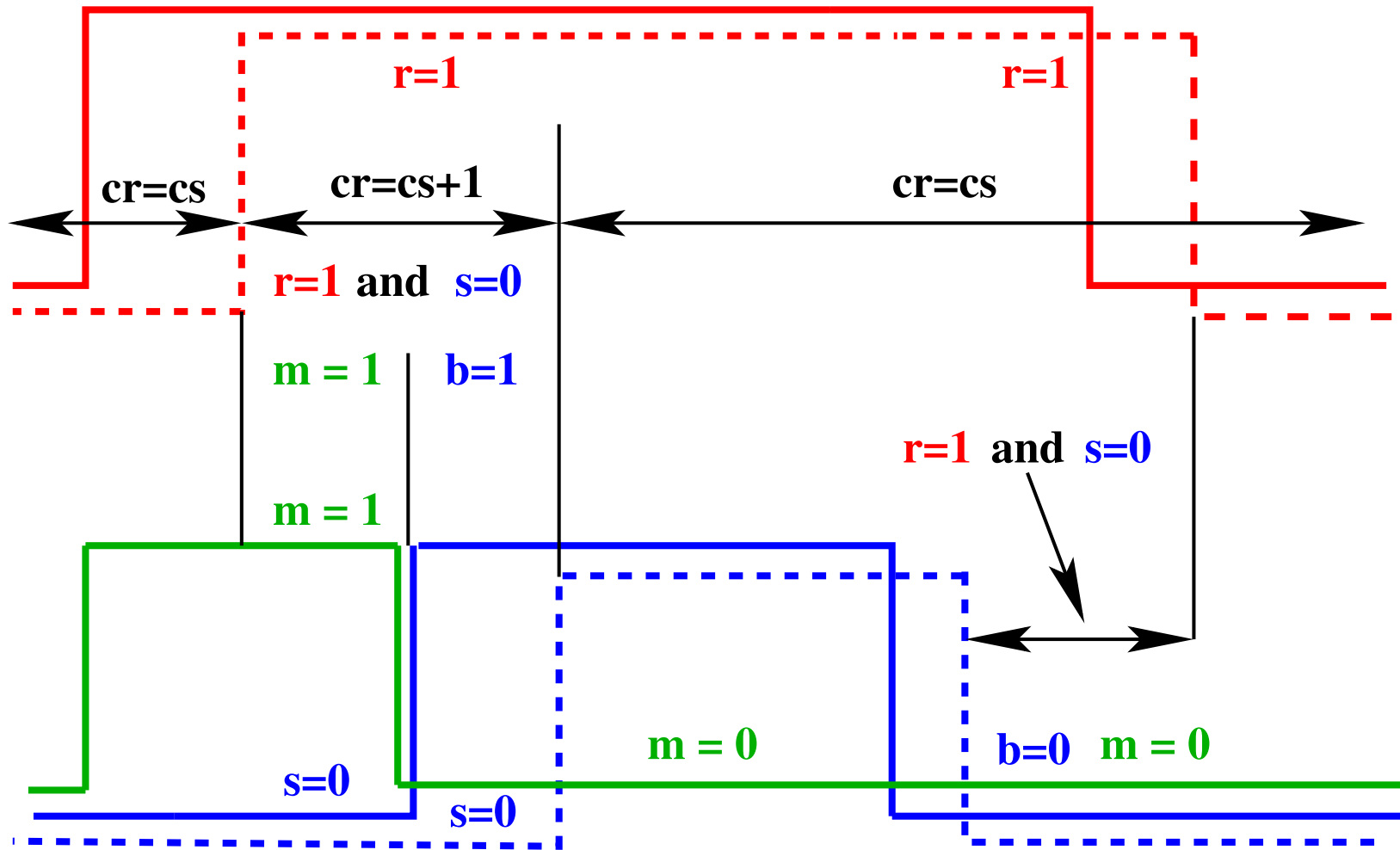


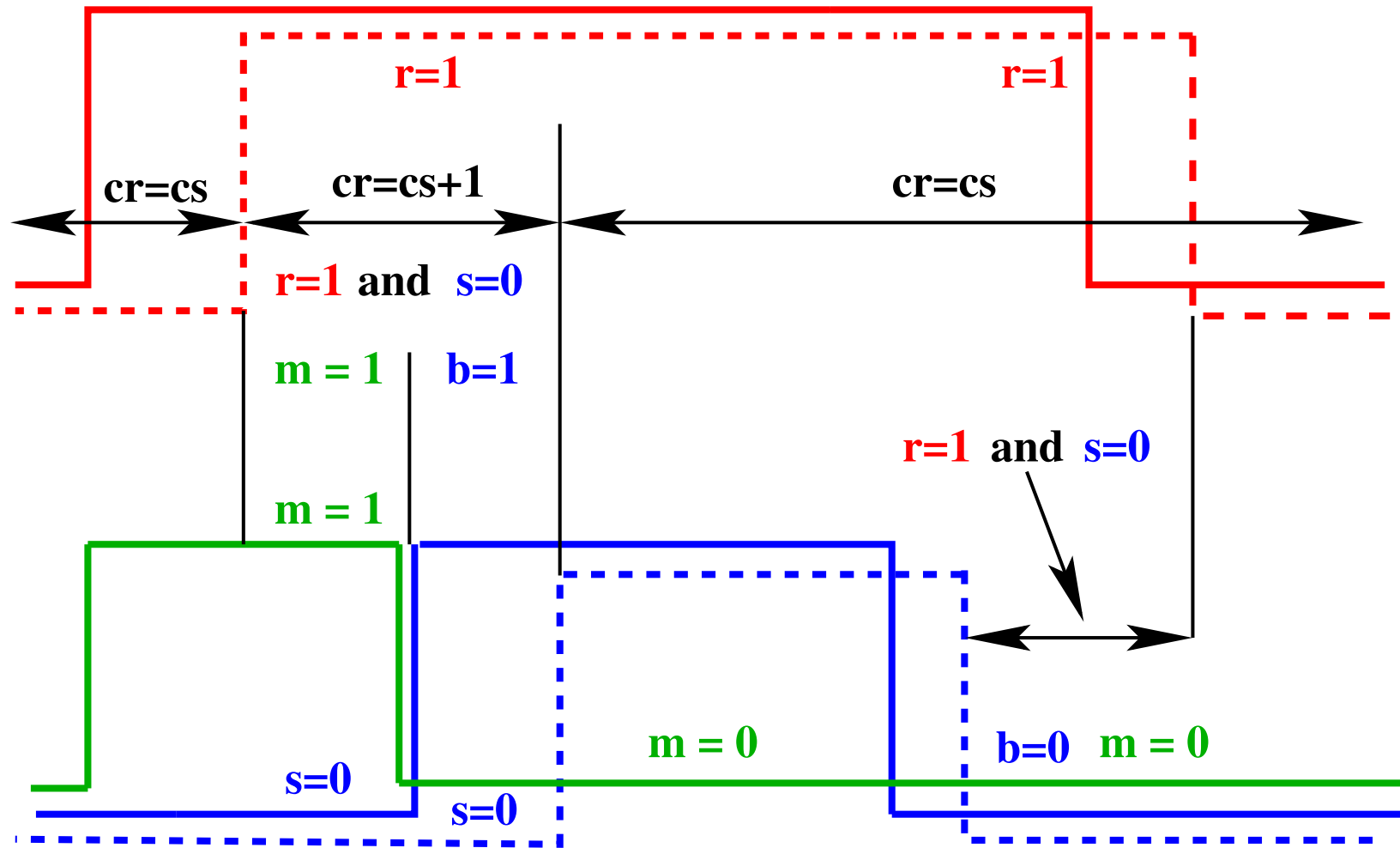




$$r = 1 \wedge s = 0 \Rightarrow cr = cs + 1 \quad ?$$







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

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

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

$$\mathbf{dbl2_2: } m = 1 \Rightarrow ca = cb + 1$$

$$\mathbf{dbl2_3: } m = 0 \Rightarrow ca = cb$$

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

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

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

$$\mathbf{dbl2_2:} \quad m = 1 \Rightarrow ca = cb + 1$$

$$\mathbf{dbl2_3:} \quad m = 0 \Rightarrow ca = cb$$

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

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

- The following theorems are easy to prove

$$\mathbf{thm2_1:} \quad ca = cb \vee ca = cb + 1$$

$$\mathbf{thm2_2:} \quad cr = cs \vee cr = cs + 1$$

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

$$\mathbf{dbl2_2:} \quad m = 1 \Rightarrow ca = cb + 1$$

$$\mathbf{dbl2_3:} \quad m = 0 \Rightarrow ca = cb$$

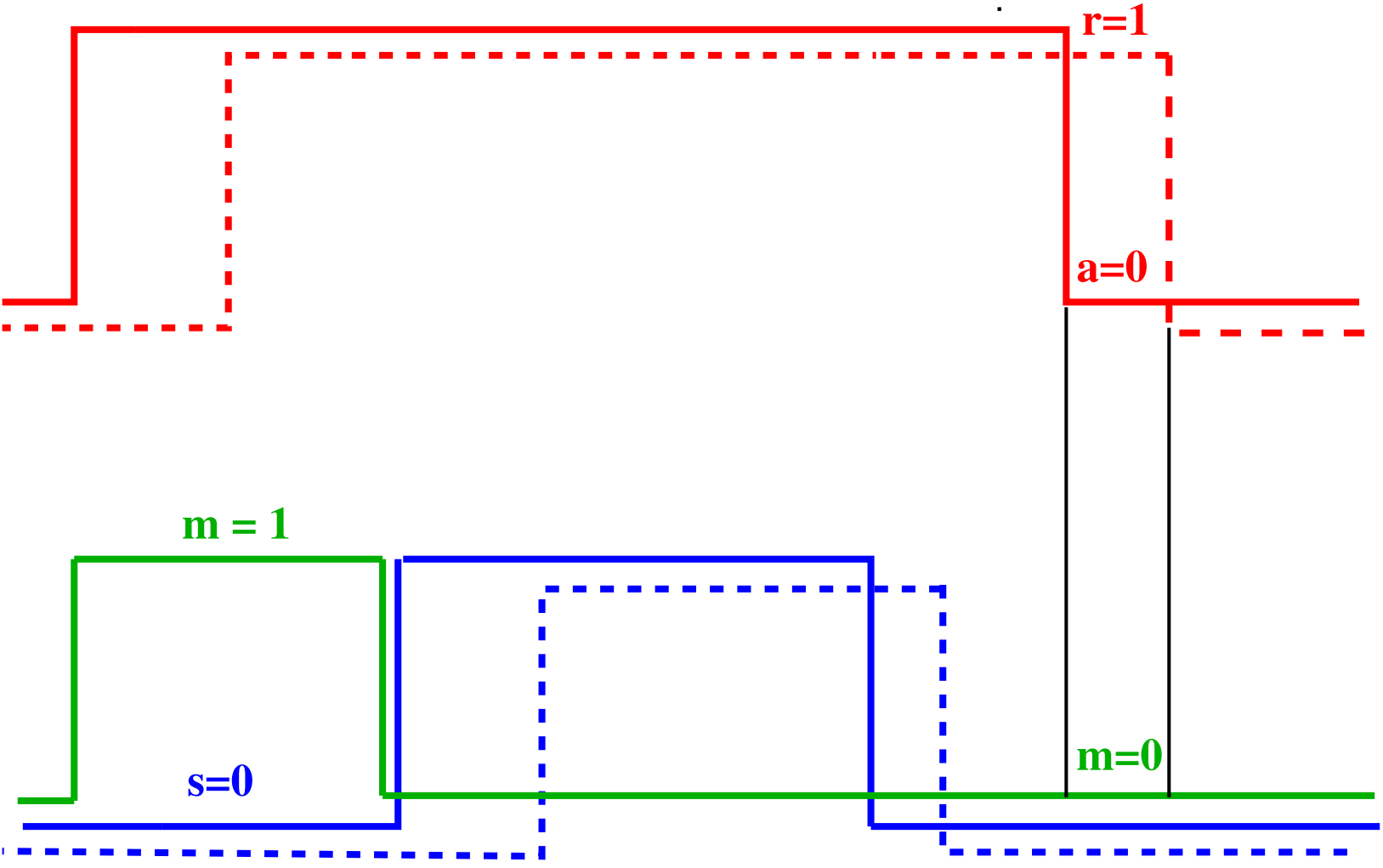
$$\mathbf{dbl2_4:} \quad r = 1 \wedge s = 0 \wedge (m = 1 \vee b = 1) \Rightarrow cr = cs + 1$$

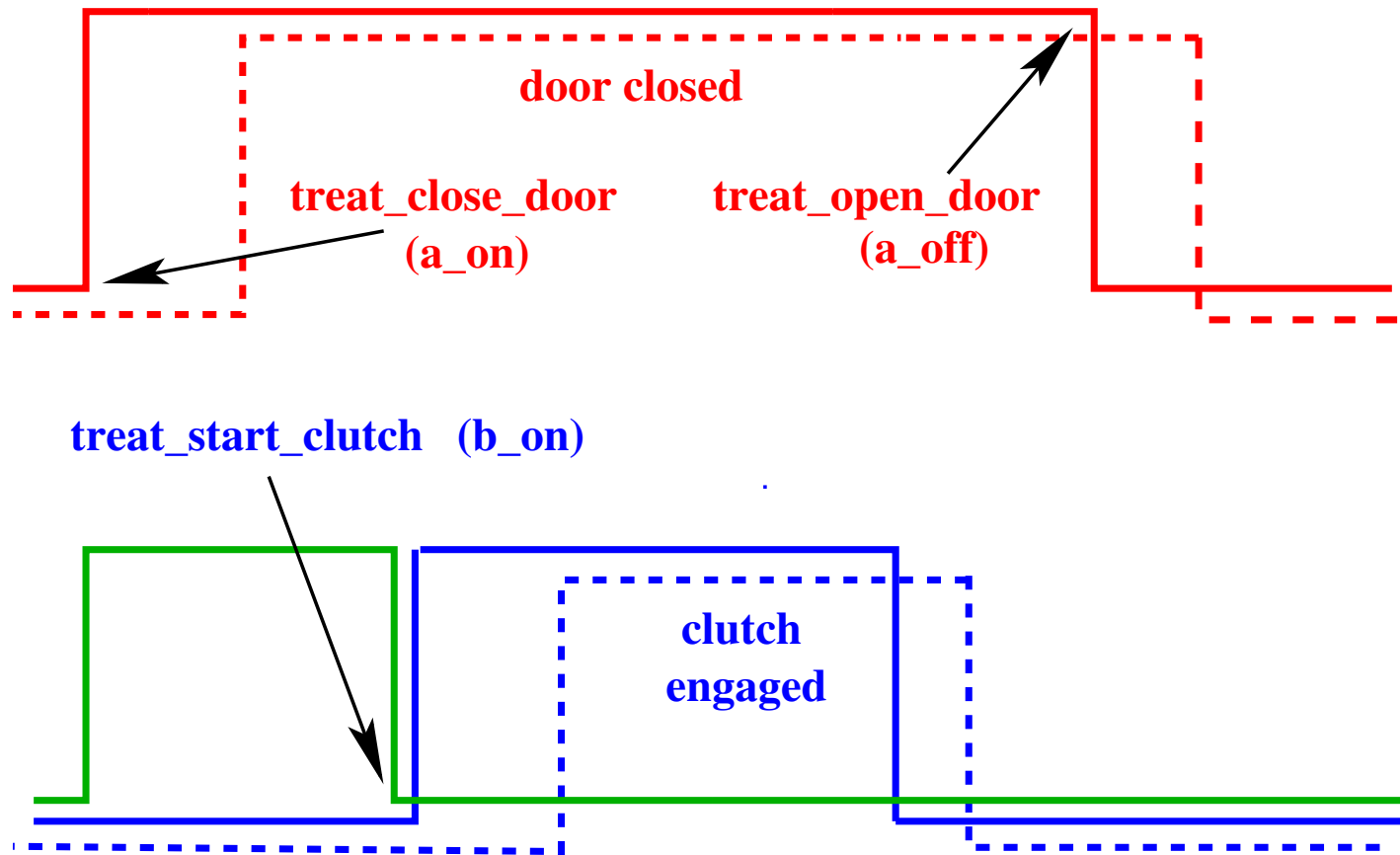
$$\mathbf{dbl2_5:} \quad r = 0 \vee s = 1 \vee (m = 0 \wedge b = 0) \Rightarrow cr = cs$$

$$\mathbf{dbl2_6:} \quad r = 1 \wedge a = 0 \Rightarrow m = 0$$

$$\mathbf{dbl2_7:} \quad m = 1 \Rightarrow s = 0$$

- The two **new invariants** were discovered **while doing the proof**
- The proofs are now **completely automatic**





- We **instantiate the pattern** as follows:

<i>a</i>	\rightsquigarrow	<i>door_actuator</i>	<i>b</i>	\rightsquigarrow	<i>clutch_actuator</i>
<i>r</i>	\rightsquigarrow	<i>door_sensor</i>	<i>s</i>	\rightsquigarrow	<i>clutch_sensor</i>
<i>0</i>	\rightsquigarrow	<i>open</i>	<i>0</i>	\rightsquigarrow	<i>disengaged</i>
<i>1</i>	\rightsquigarrow	<i>closed</i>	<i>1</i>	\rightsquigarrow	<i>engaged</i>

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

```
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

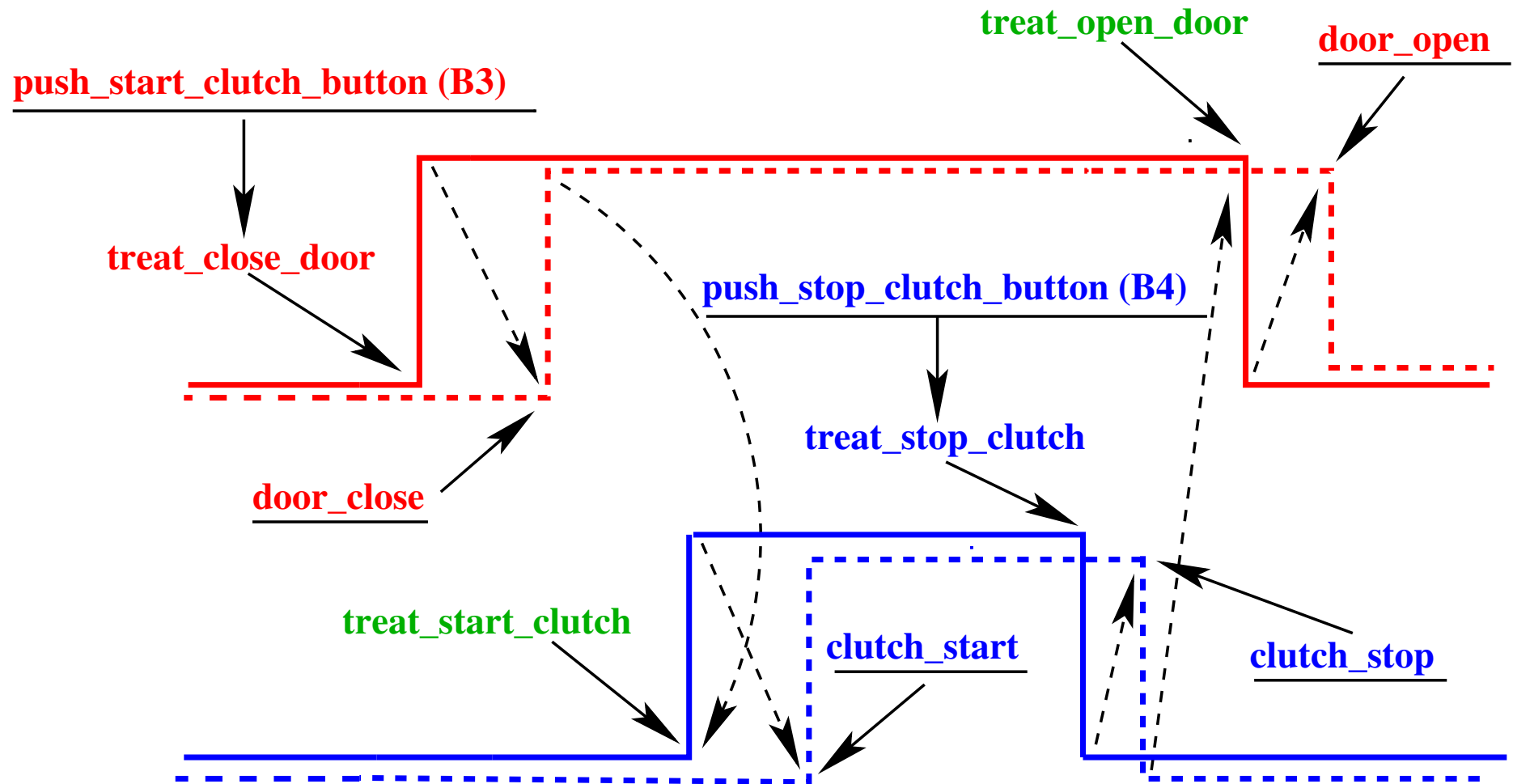
$clutch_actuator := engaged$

$m := 0$

end

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

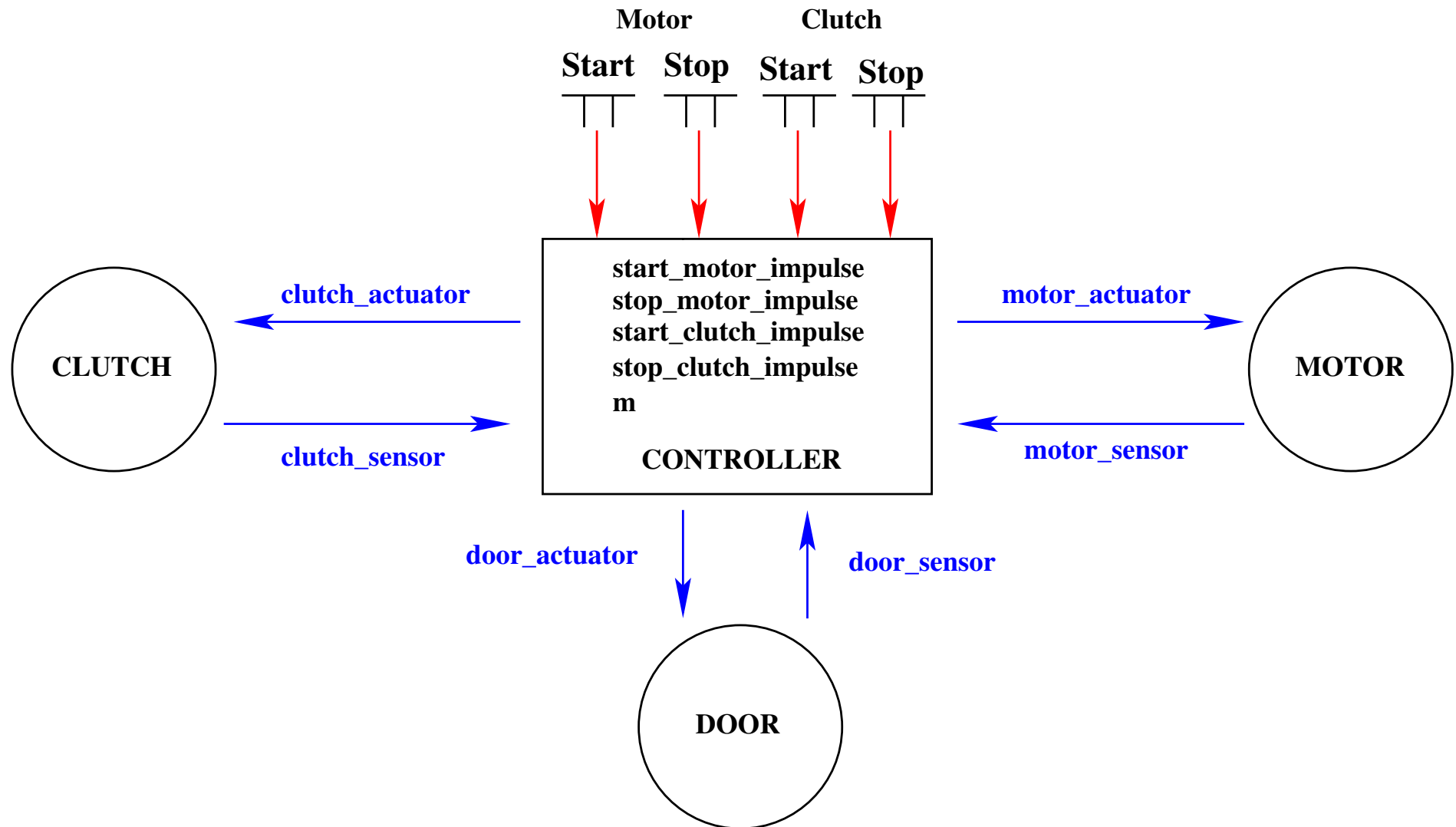
```
treat_open_door
when
   $door\_actuator = closed$ 
   $door\_sensor = closed$ 
   $clutch\_sensor = disengaged$ 
   $clutch\_actuator = disengaged$ 
   $m = 0$ 
then
   $door\_actuator := open$ 
end
```



- 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

- 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



- 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

-
- 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

-
- 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

- 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*

- 14 environment events,
- 14 controller events,
- 130 lines for environment events,
- 180 lines for controller events.

-
- 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

- 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

-
- 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: $4 \times 18 + 3 \times 12 + 3 \times 60 + 40 = 328$

- 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.

- 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**)