

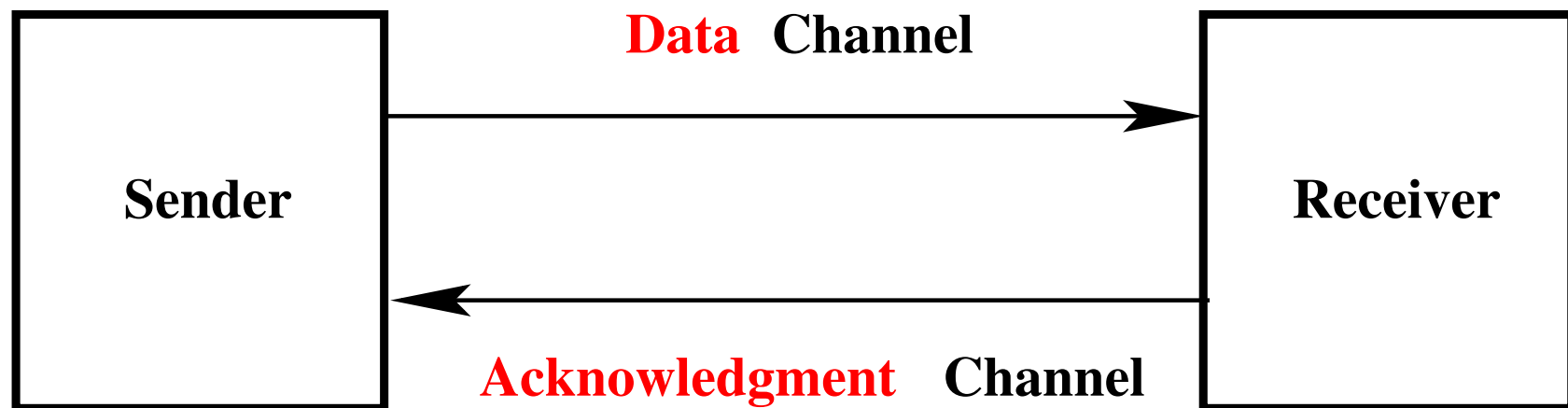
4. The Bounded Re-transmission Protocol

Jean-Raymond Abrial

July 2007

- The Bounded Re-transmission Protocol is a **file transfer protocol**
- This is a problem dealing with **fault tolerance**
- We suppose that the transfer channels are **unreliable**
- We present classical solutions to handle that problem: **timers**.
- We would like to see how we can **formalize such timers**

- A **sequential file** is transmitted from a **Sender** to a **Receiver**
- The file is transmitted **piece by piece** through a **Data Channel**
- After receiving some data, the Receiver sends an **acknowledgment**
- After receiving it, the Sender sends the **next piece of data**, etc.



- **Messages can be lost** in the Data or Acknowledgment channels

The goal of the BRP is to **totally** or **partially** transfer a certain non-empty original sequential file from one site to another.

FUN-1

A **total transfer** means that the transmitted file is a **copy** of the original one.

FUN-2

A **partial transfer** means that the transmitted file is a **genuine prefix** of the original one.

FUN-3

- Messages can be lost in the Data or Acknowledgment channels
- The Sender starts a timer before sending a piece of data
- The timer wakes up the Sender after a delay d
- This occurs if the Sender has not received an acknowledgment in the meantime

- Delay d is guaranteed to be **greater than twice the transmission time**
- When waken up, the Sender is then **sure** that the **data** or the **acknowledgment** has been **lost**
- When waken up, the Sender **re-transmits the previous data**
- The senders send an **alternating bit** together with a **new data**
- This ensures that the Receiver **does not confuse** (?) a **new data** with a **retransmitted** one.

- The Sender can re-transmit the same data **at most M times**
- After this, the Sender **decides to abort**
- **How does the Receiver know** that the Sender aborted?

- Each time the Receiver receives a **new** piece of data, it **starts a timer**
- The timer **wakes up** the Receiver after a delay $(M + 1) \times d$
- This occurs if the Sender **has not received a new data** in the meantime.
- After this delay, the Receiver is certain that **the Sender has aborted**
- Then the **Receiver aborts** too.

- At the end of the protocol, we might be in one of the **three situations**:
 - (1) The file **has been transmitted** entirely and the Sender **has received** the last acknowledgment
 - (2) The file **has been transmitted** entirely but the Sender **has not received** the last acknowledgment
 - (3) The file **has not been transmitted** entirely

Each site may end up in any of the **two situations**:

- either **it believes** that the protocol has **terminated successfully**,
- or **it believes** that the protocol has **aborted**

FUN-4

When the **Sender** believes that the protocol has **terminated successfully** then the **Receiver** believes so too.

FUN-5

However, it is possible for the **Sender** to believe that the protocol has **aborted** while the **Receiver** believes that it has **terminated successfully**.

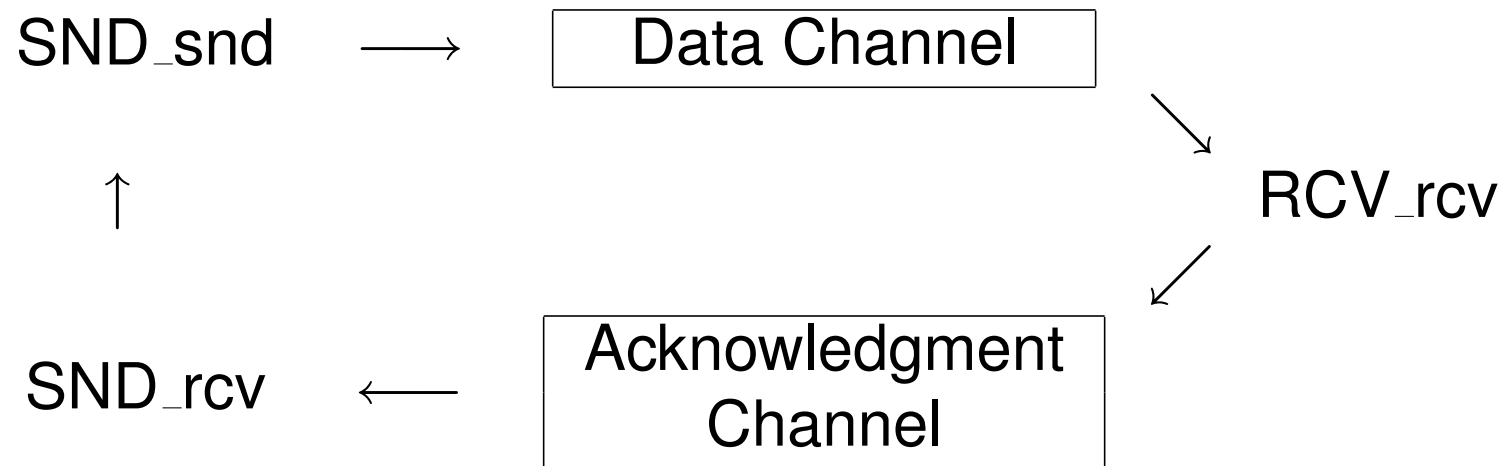
FUN-6

When the **Receiver** believes that the protocol has **terminated successfully**, this is because the original file has been **entirely copied** on the Receiver's site.

FUN-7

When the **Receiver** believes that the protocol has **aborted**, this is because the original file has **not been copied entirely** on the Receiver's site.

FUN-8



SND_snd

when

SND_snd is waken up

then

Acquire data from Sender's file;

Store acquired data on Data Channel;

Store Sender's bit on Data Channel;

Start Sender's timer;

Activate Data Channel

end

```
RCV_rcv
  when
    Data Channel interrupt occurs
  then
    Acquire Sender's bit from Data Channel;
    if Sender's bit=Receiver's bit then
      Acquire Data from Data Channel;
      Store data on Receiver's file;
      Modify Receiver's bit;
      if data is not the last one then
        Start Receiver's timer
      end
    end
  end
  Reset Data Channel Interrupt;
  Activate Acknowledgement Channel
end
```

```
SND_rcv
```

```
  when
```

```
    Acknowledgment Channel interrupt occurs
```

```
  then
```

```
    Remove Data from sender's file;
```

```
    Reset retry counter;
```

```
    Modify sender's bit;
```

```
    Wake up event SND_snd;
```

```
    Reset Acknowledgment Channel interrupt;
```

```
    if Sender's file is not empty then
```

```
      Wake up event SND_snd
```

```
    end
```

```
  end
```

```
SND_timer
  when
    Sender's timer interrupt occurs
  then
    if retry counter is equal to M+1 then
      Abort protocol on Sender's site;
    else
      Increment retry counter;
      Wake up event SND_snd
    end
  end
end
```

RCV_timer

when

Receiver's timer interrupt occurs

then

Abort protocol on Receiver's site

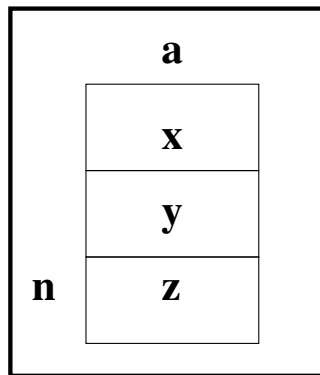
end

-
- Quite often, protocols are "specified" by such pseudo-codes
 - In fact, such a pseudo-code raises a number of questions:
 - Are we sure that this description is correct?
 - Are we sure that this protocol terminates?
 - What kinds of properties should this protocol maintain?
 - Hence the formal development which is presented now

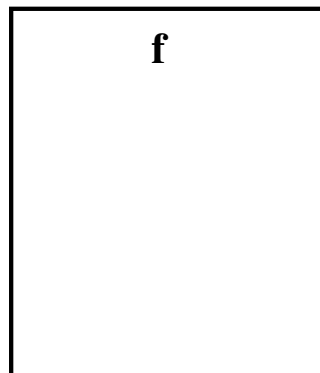
- FUN_1, FUN_2, and FUN_3: **partial transmission of the file.**
- FUN_4 to FUN_8. But each participant can have a **direct access** to the other
- Introducing **unreliable channels** and **timers.**

INITIAL SITUATION

SENDER

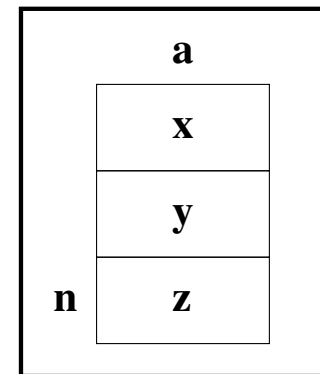


RECEIVER

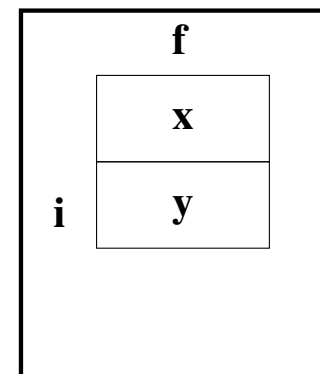


FINAL SITUATION

SENDER



RECEIVER



- Constant n denotes the **size** of file a
- Constant a denotes the **original file**,

carrier sets: D

constants: n, a

prp0_1: $n \in \mathbb{N}$

prp0_2: $0 < n$

prp0_3: $a \in 1 .. n \rightarrow D$

- Variable i denotes the **size** of file f
- Variable f denotes the **transmitted file**

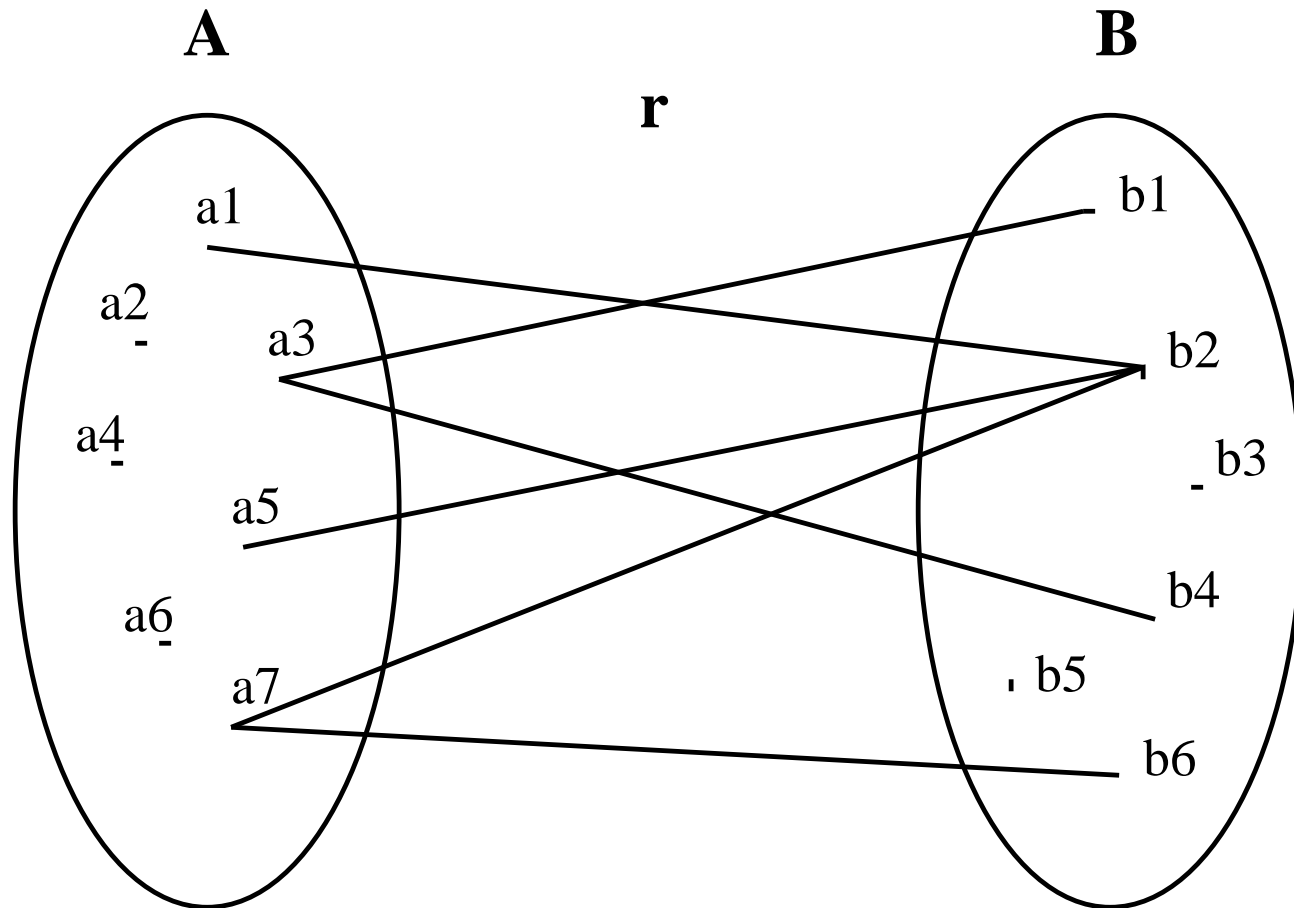
variables: i, f

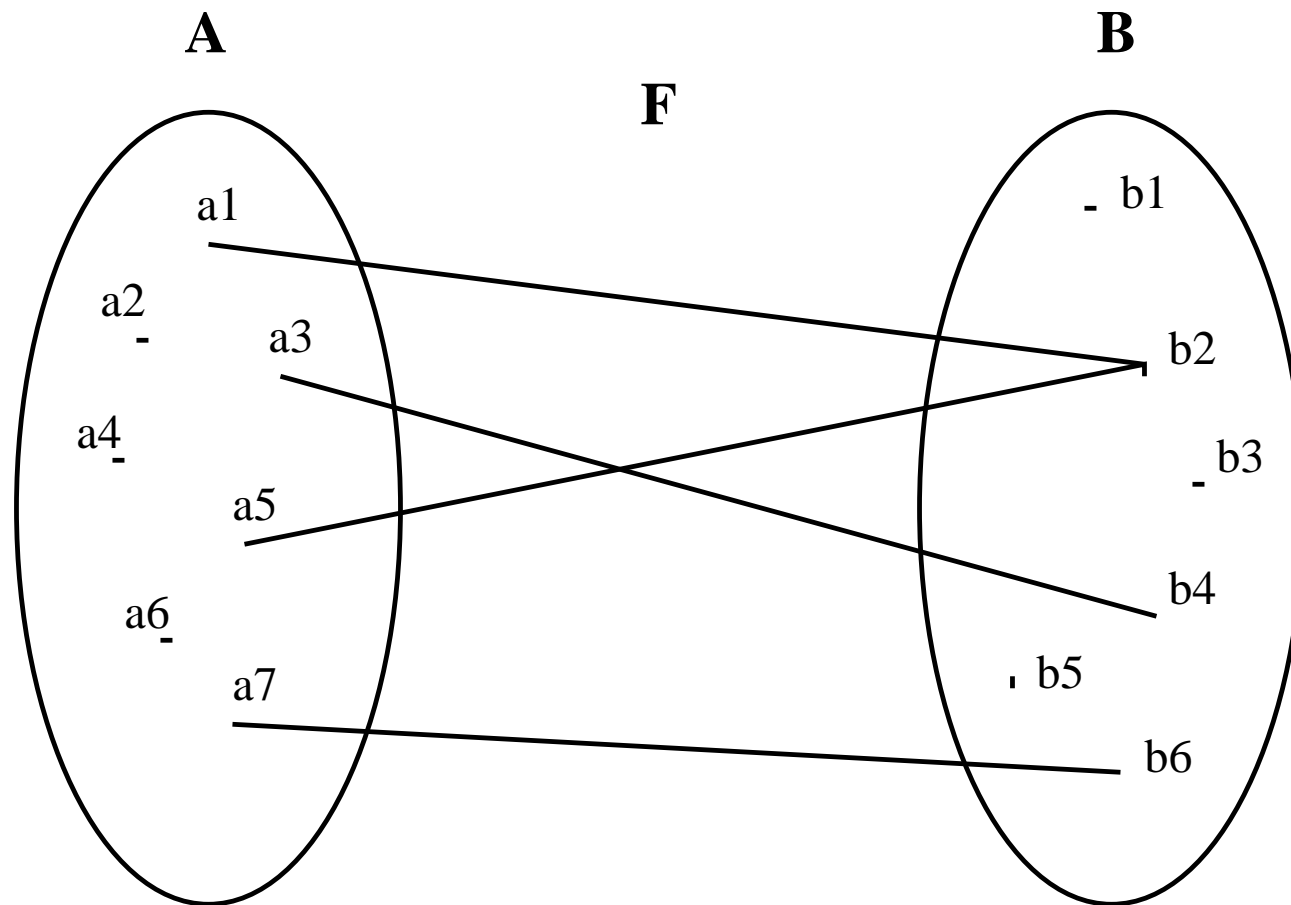
inv0_1: $i \in 0 .. n$

inv0_2: $f \in 1 .. i \rightarrow D$

$x \in S$	set membership operator
\mathbb{N}	set of natural numbers: $\{0, 1, 2, 3, \dots\}$
$a .. b$	interval from a to b : $\{a, a + 1, \dots, b\}$ (empty when $b < a$)
$a \mapsto b$	pair constructing operator
$S \times T$	Cartesian product operator
$S \subseteq T$	set inclusion operator
$\mathbb{P}(S)$	power set operator

$S \leftrightarrow T$	set of binary relations from S to T
$S \rightarrow T$	set of total functions from S to T
$S \twoheadrightarrow T$	set of partial functions from S to T
$\text{dom}(r)$	domain of a relation r
$\text{ran}(r)$	range of a relation r

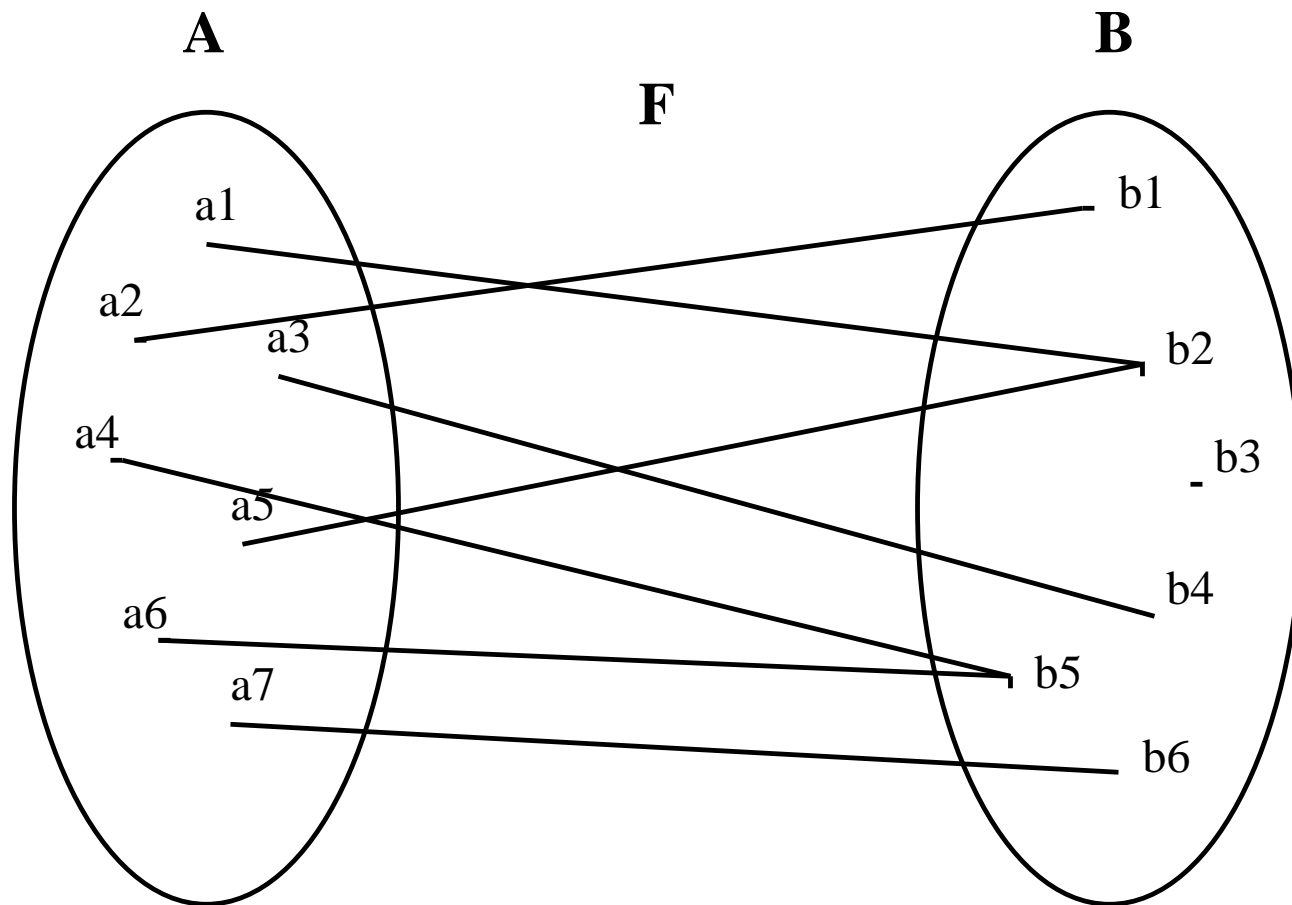




$$F = \{a1 \mapsto b2, a3 \mapsto b4, a5 \mapsto b2, a7 \mapsto b6\}$$

$$\text{dom}(F) = \{a1, a3, a5, a7\}$$

$$\text{ran}(F) = \{b2, b4, b6\}$$



$$\text{dom}(F) = A$$

- Event **brp** describes the situation at the **end of the protocol**
- It only says that the file might be **partially transmitted**
- It is made of a **non-deterministic assignment**

init

$i := 0$

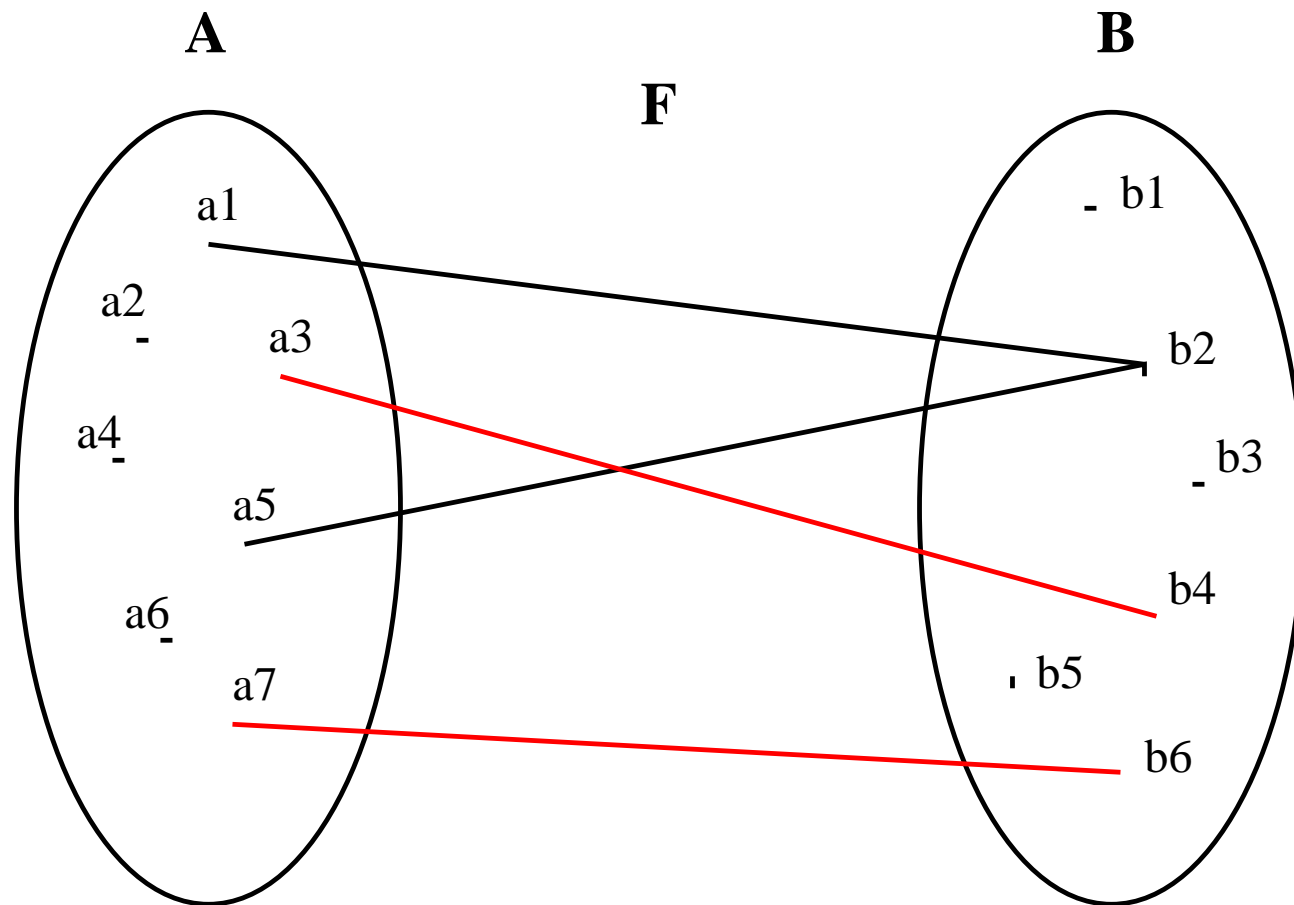
$f := \emptyset$

brp

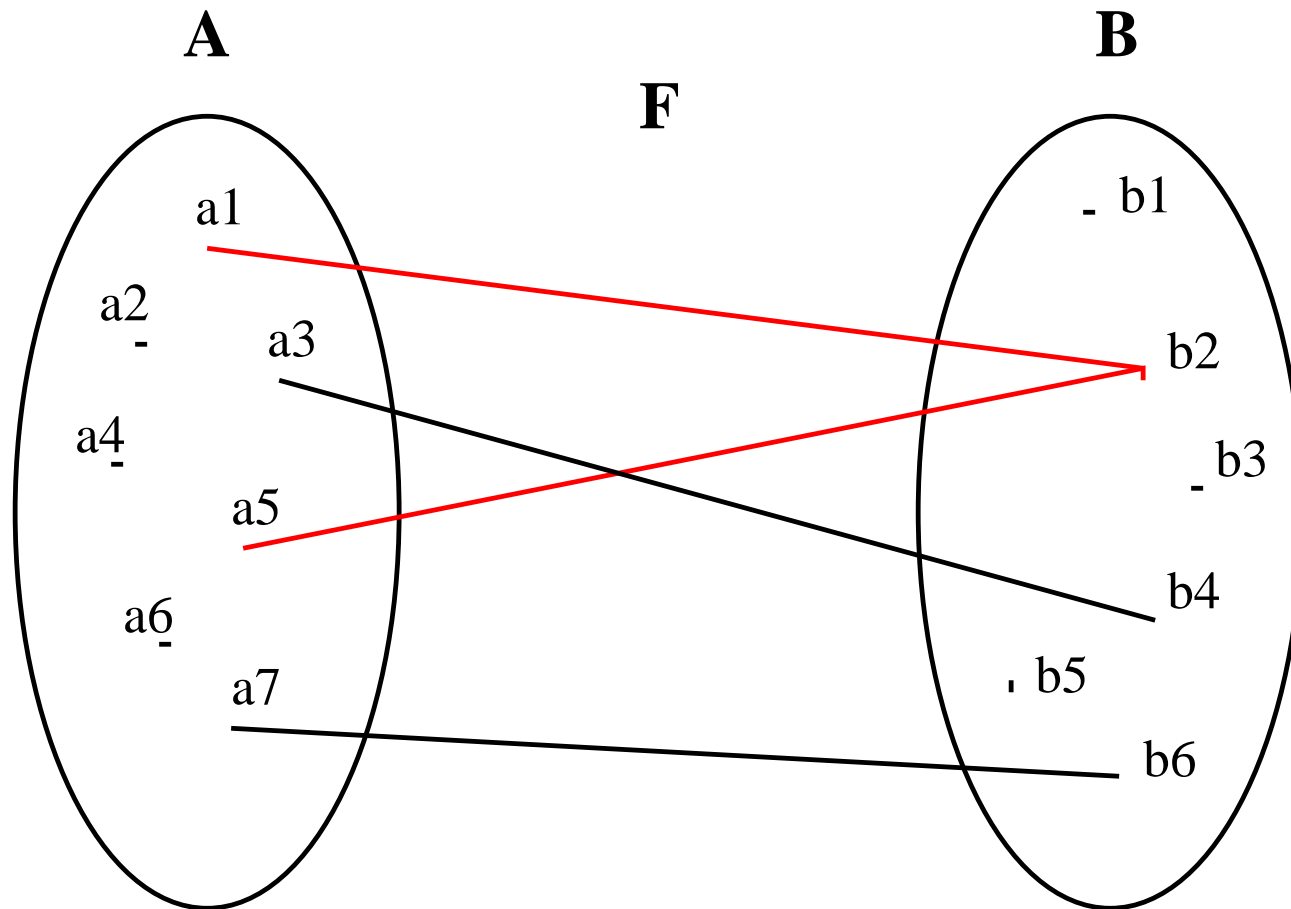
$i, f :| \left(\begin{array}{l} i' \in 0 .. n \\ f' = (1 .. i') \triangleleft a \end{array} \right)$

- Operator **:|** is to be read: "***i* and *f* become such that ...**"

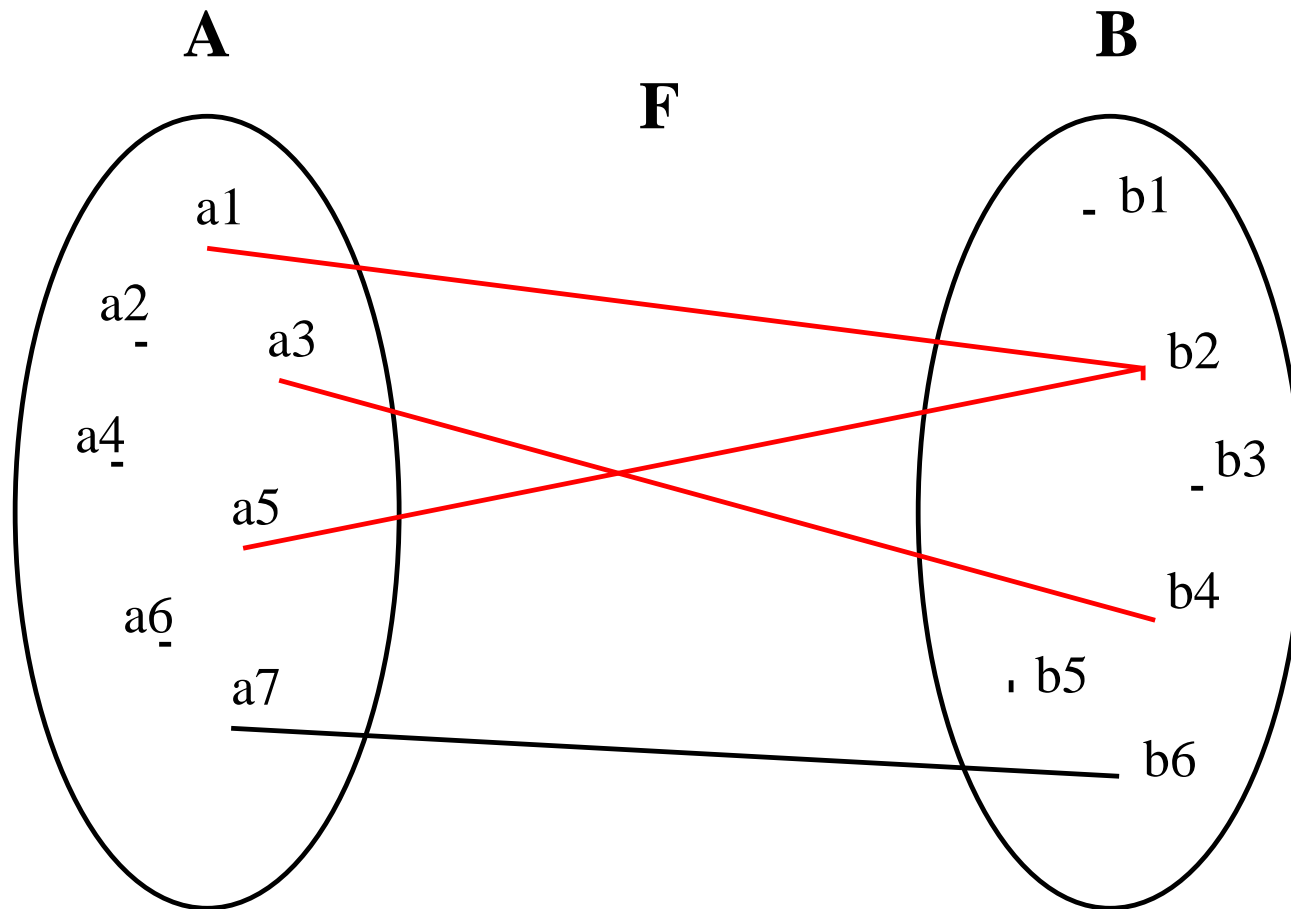
$s \triangleleft r$	domain restriction operator
$s \triangleleft r$	domain subtraction operator
$r \triangleright t$	range restriction operator
$r \triangleright t$	range subtraction operator



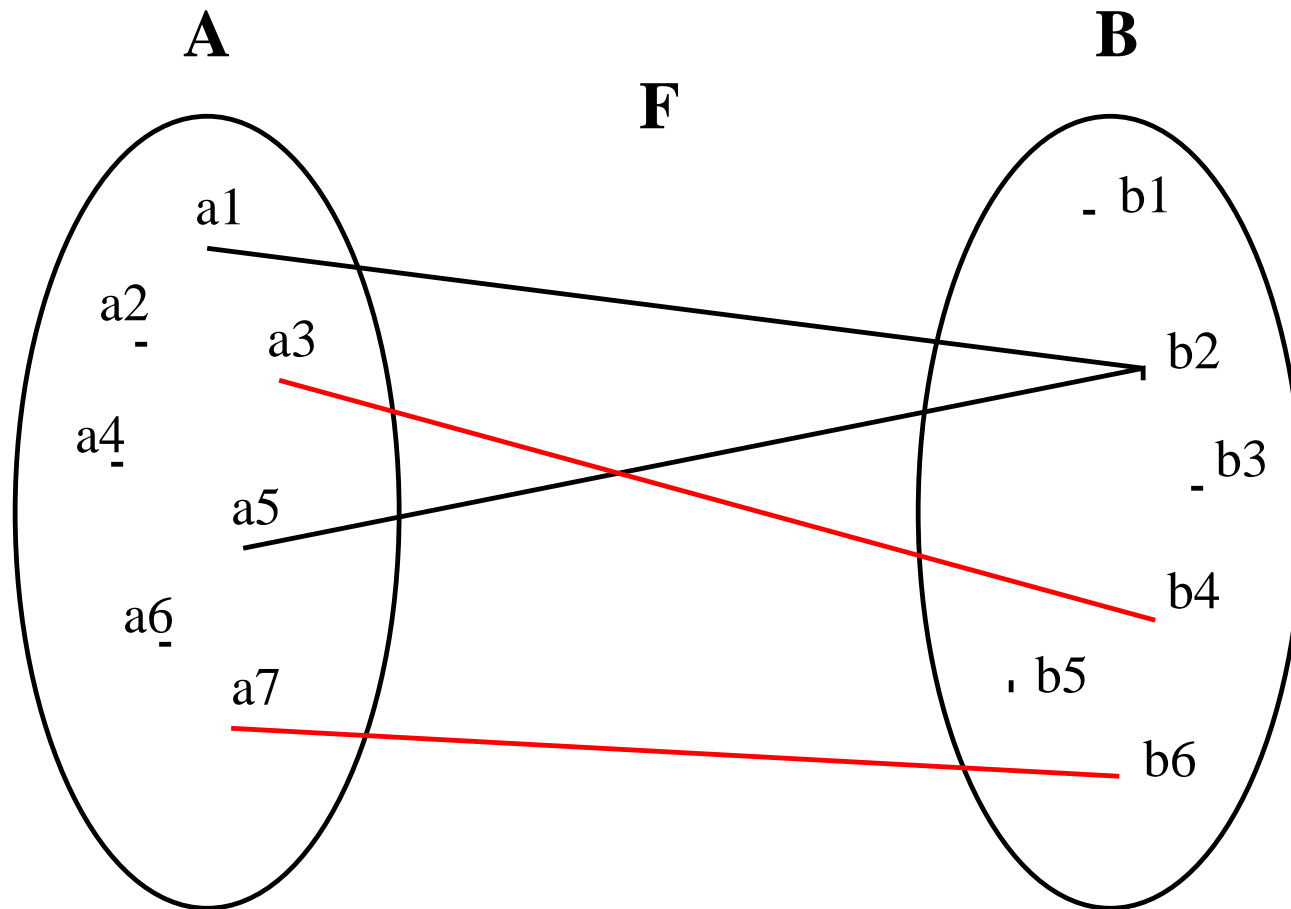
$$\{a_3, a_7\} \triangleleft F$$



$$\{a_3, a_7\} \triangleleft F$$



$$F \triangleright \{b2, b4\}$$



$$F \triangleright \{b_2\}$$

brp

$$i, f : | \left(\begin{array}{l} i' \in 0 .. n \\ f' = (1 .. i') \triangleleft a \end{array} \right)$$

i and f are assigned **any values i' and f'** such that the following holds:

$$i' \in 0 .. n \wedge f' = (1 .. i') \triangleleft a$$

```
when
  conditions
then
  assignments
end
```

- where *assignments* was defined according to the syntax

variables := expressions

- This syntax is now **extended** to

variables :| predicates

variables :| *predicates*

- *variables* denotes the **concerned state variables**
- *predicates* directly denotes the **before-after predicate**
- It is possible to **mix determinacy and non-determinacy**

- To each event can be associated a **before-after predicate**
- It denotes the **relationship** holding between the **values** of the variable *just before* and *just after* the event occurrence
- The **before-value** is denoted by the **variable names**
- The **after-value** is denoted by the **variable names *primed***

r_on

when

$r = 0$

$a = 1$

then

$r := 1$

$cr := cr + 1$

end

Variables a and ca are unchanged

r_on

when

$r = 0$

$a = 1$

then

$r := 1$

$cr := cr + 1$

end

Variables a and ca are unchanged

The corresponding **before-after** predicate is

$$r' = 1 \quad \wedge \quad cr' = cr + 1$$

r_on

when

$r = 0$

$a = 1$

then

$r := 1$

$cr := cr + 1$

end

Variables a and ca are unchanged

The corresponding **before-after** predicate is

$$r' = 1 \wedge cr' = cr + 1 \wedge a' = a \wedge ca' = ca$$

- The before-after predicate we have shown is **very simple**

$$r' = 1 \quad \wedge \quad cr' = cr + 1 \quad \wedge \quad a' = a \quad \wedge \quad ca' = ca$$

- The primed after-values are **functions** of the before-values
- This is because the corresponding events are **deterministic**
- We might have **non-deterministic** before-after predicates

when

$G(c, v)$

then

$v := E(c, v)$

end

$P(c)$

$I(c, v)$

$G(c, v)$

⊢

$I(c, E(c, v))$

INV

when

$G(c, v)$

then

$v :| Q(c, v, v')$

end

$P(c)$

$I(c, v)$

$G(c, v)$

$Q(c, v, v')$

⊢

$I(c, v')$

INV

- For other events in the initial model
- **Deterministic Case**

Properties Invariants Guards of the event ┆ Modified Invariant	INV
--	-----

- For other events in the initial model

- **Non-deterministic Case**

Properties Invariants Guards of the event Before-after predicate of the event ┆ Modified Invariant	INV
---	-----

- "Before-after predicate" means the predicate associated with the action of the event

prp0_1	$n \in \mathbb{N}$
prp0_2	$0 < n$
prp0_3	$a \in 1 .. n \rightarrow D$
inv0_1	$i \in 0 .. n$
inv0_2	$f \in 1 .. i \rightarrow D$
before-after	$i' \in 0 .. n$
predicate of brp	$f' = (1 .. i') \triangleleft a$
┆	┆
Modified inv0_1	$i' \in 0 .. n$

prp0_1	$n \in \mathbb{N}$
prp0_2	$0 < n$
prp0_3	$a \in 1 .. n \rightarrow D$
inv0_1	$i \in 0 .. n$
inv0_2	$f \in 1 .. i \rightarrow D$
before-after	$i' \in 0 .. n$
predicate of brp	$f' = (1 .. i') \triangleleft a$
┆	┆
Modified inv0_2	$f' \in 1 .. i' \rightarrow D$

when

$G(c, v)$

then

$v :| Q(c, v, v')$

end

$P(c)$

$I(c, v)$

$G(c, v)$

\vdash

$\exists v' \cdot Q(c, v, v')$

FIS

- We have to prove that the before-after predicate **is not void**

- Non-deterministic Case

Properties Invariants Guards of the event ⊢ $\exists v' \cdot (\text{Before-after predicate of the event})$	FIS
---	-----

prp0_1

prp0_2

prp0_3

inv0_1

inv0_2

$n \in \mathbb{N}$

$0 < n$

$a \in 1 .. n \rightarrow D$

$f \in \mathbb{N} \leftrightarrow D$

$i \in \mathbb{N}$

⊢

$\exists i', f' \cdot \left(\begin{array}{l} \text{BA pred.} \\ \text{of brp} \end{array} \right)$

⊢

$\exists i', f' \cdot \left(\begin{array}{l} i' \in 0 .. n \\ f' = (1 .. i') \triangleleft a \end{array} \right)$

Proposing **witnesses** 0 and \emptyset

$n \in \mathbb{N}$

$0 < n$

$a \in 1 .. n \rightarrow D$

$f \in \mathbb{N} \leftrightarrow D$

$i \in \mathbb{N}$

⊢

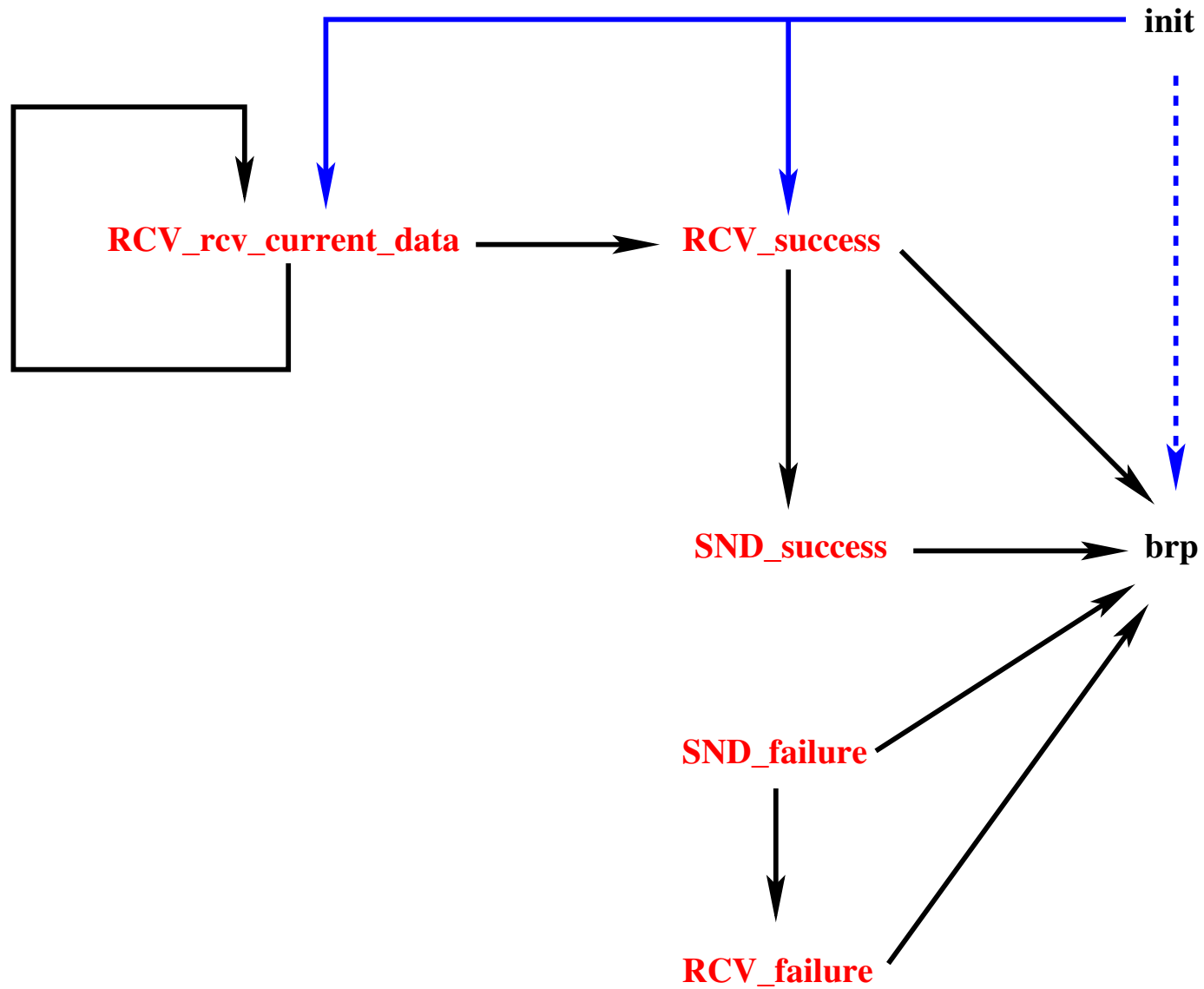
$0 \in 0 .. n$

$\emptyset = (1 .. 0) \triangleleft a$

init



brp



-
- They allow to **observe** the (future) system with a **finer time grain**
 - Analogies with a **microscope** or a **parachute**
 - They **refine** the (implicit) event **doing nothing** (skip)
 - They must **not take control for ever** (exhibiting a variant)

carrier sets: D, STATUS

constants: $a, n,$
 $\mathit{working},$
 $\mathit{success},$
 $\mathit{failure}$

prp1_1: $\mathit{STATUS} = \{\mathit{working}, \mathit{success}, \mathit{failure}\}$

prp1_2: $\mathit{working} \neq \mathit{success}$

prp1_3: $\mathit{working} \neq \mathit{failure}$

prp1_4: $\mathit{success} \neq \mathit{failure}$

- Variables f and i are replaced by variables b and m
- Variable m denotes the size of the transmitted file
- Variable b denotes the transmitted file
- The file b is the same as the file a restricted to the interval $1 .. m$

variables: m, b

inv1_1: $m \in 0 .. n$

inv1_2: $b = (1 .. m) \triangleleft a$

- Variables r and s denotes the status of both participants

variables: m, b, r, s

inv1_3: $r \in STATUS$

inv1_4: $s \in STATUS$

- Requirements FUN_4 to FUN_8

$$\mathbf{inv1_5:} \quad r = \mathit{success} \Leftrightarrow m = n$$

$$\mathbf{inv1_6:} \quad s = \mathit{success} \Rightarrow r = \mathit{success}$$

- When the **Receiver succeeds**, the file has been **entirely transfered** (**inv1_5**)
- When the **Receiver does not succeed** (working or failure), then the file has **not been entirely transfered** (**inv1_5**)
- When the **Sender succeeds**, the **Receiver succeeds** too (**inv1_6**)

init

$m := 0$

$b := \emptyset$

$r := \textit{working}$

$s := \textit{working}$

(concrete_)brp

when

$r \neq \textit{working}$

$s \neq \textit{working}$

then

$i, f := m, b$

end

(abstract_)brp

$i, f :| \left(\begin{array}{l} i' \in 0..n \\ f' = (1..i') \triangleleft a \end{array} \right)$

Rcv_rcv_current_data

when

$r = \textit{working}$

$m + 1 < n$

then

$m := m + 1$

$b := b \cup \{m + 1 \mapsto a(m + 1)\}$

end

- This new event **maintains invariant inv1_5** and it refines **skip**

inv1_5: $r = \textit{success} \Leftrightarrow m = n$

RCV_success

when

$r = \textit{working}$

$m + 1 = n$

then

$r := \textit{success}$

$m := m + 1$

$b := b \cup \{n \mapsto a(n)\}$

end

RCV_failure

when

$r = \textit{working}$

$s = \textit{failure}$

then

$r := \textit{failure}$

end

- These new events **maintain inv1_5** and **inv1_6** and they refine **skip**

inv1_5: $r = \textit{success} \Leftrightarrow m = n$

inv1_6: $s = \textit{success} \Rightarrow r = \textit{success}$

```
SND_success
when
  s = working
  r = success
then
  s := success
end
```

```
SND_failure
when
  s = working
then
  s := failure
end
```

- Event SND_success **maintains invariant inv1_6**

```
inv1_6: s = success  $\Rightarrow$  r = success
```

- These new events are still **very abstract**. They refine **skip**.

```

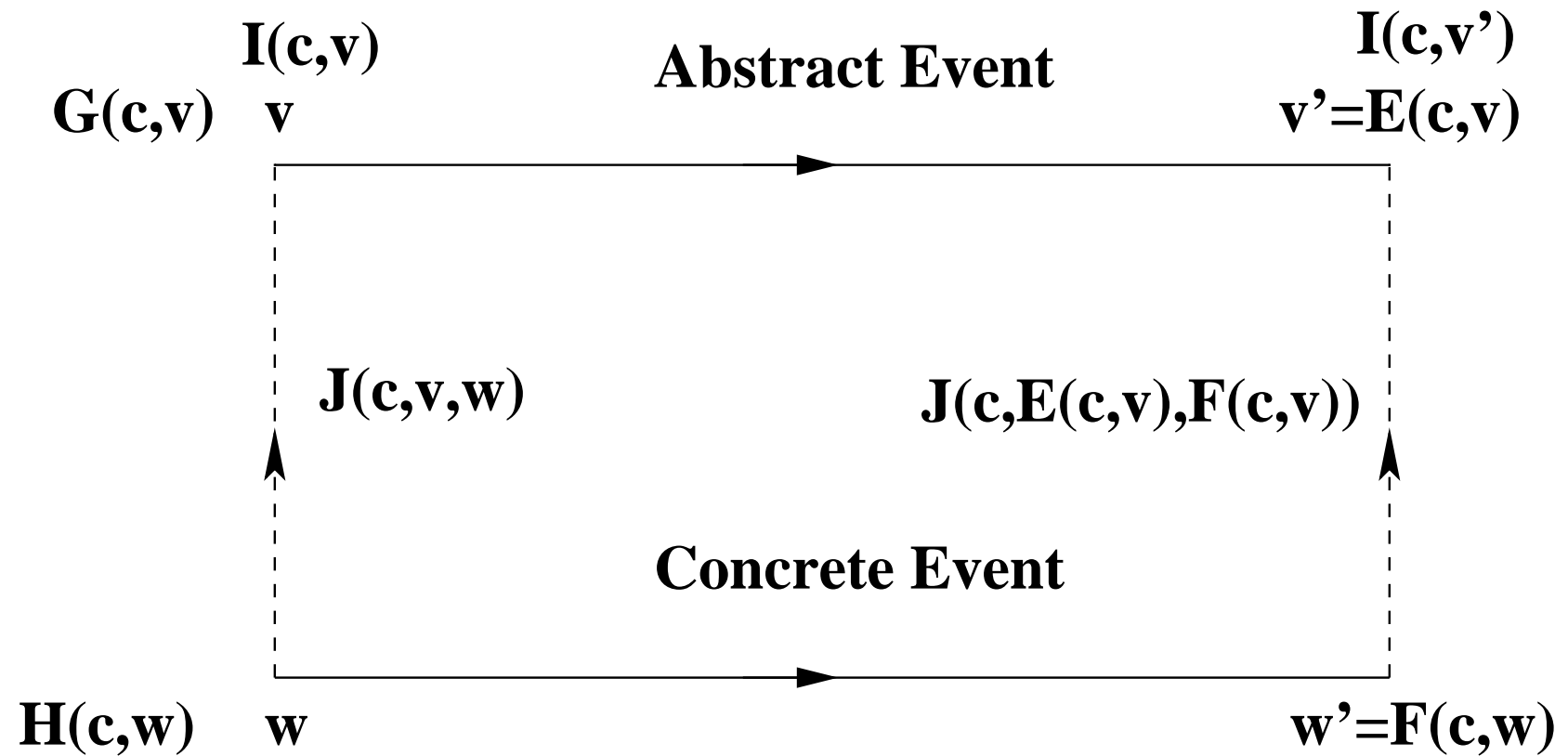
    (abstract_)Event
    when
       $G(c, v)$ 
    then
       $v := E(c, v)$ 
    end
    
```

```

    (concrete_)Event
    when
       $H(c, w)$ 
    then
       $w := F(c, w)$ 
    end
    
```

$P(c)$ $I(c, v)$ $J(c, v, w)$ $H(c, w)$ \vdash $G(c, v)$	GRD_REF	$P(c)$ $I(c, v)$ $J(c, v, w)$ $H(c, w)$ \vdash $J(c, E(c, v), F(c, w))$	INV_REF
---	---------	--	---------

Guard strengthening and invariant preservation



- For old events only

Properties Abstract invariants Concrete invariants Concrete guards of the event ┆ Abstract guards of the event	GRD_REF
---	---------

- For init event only

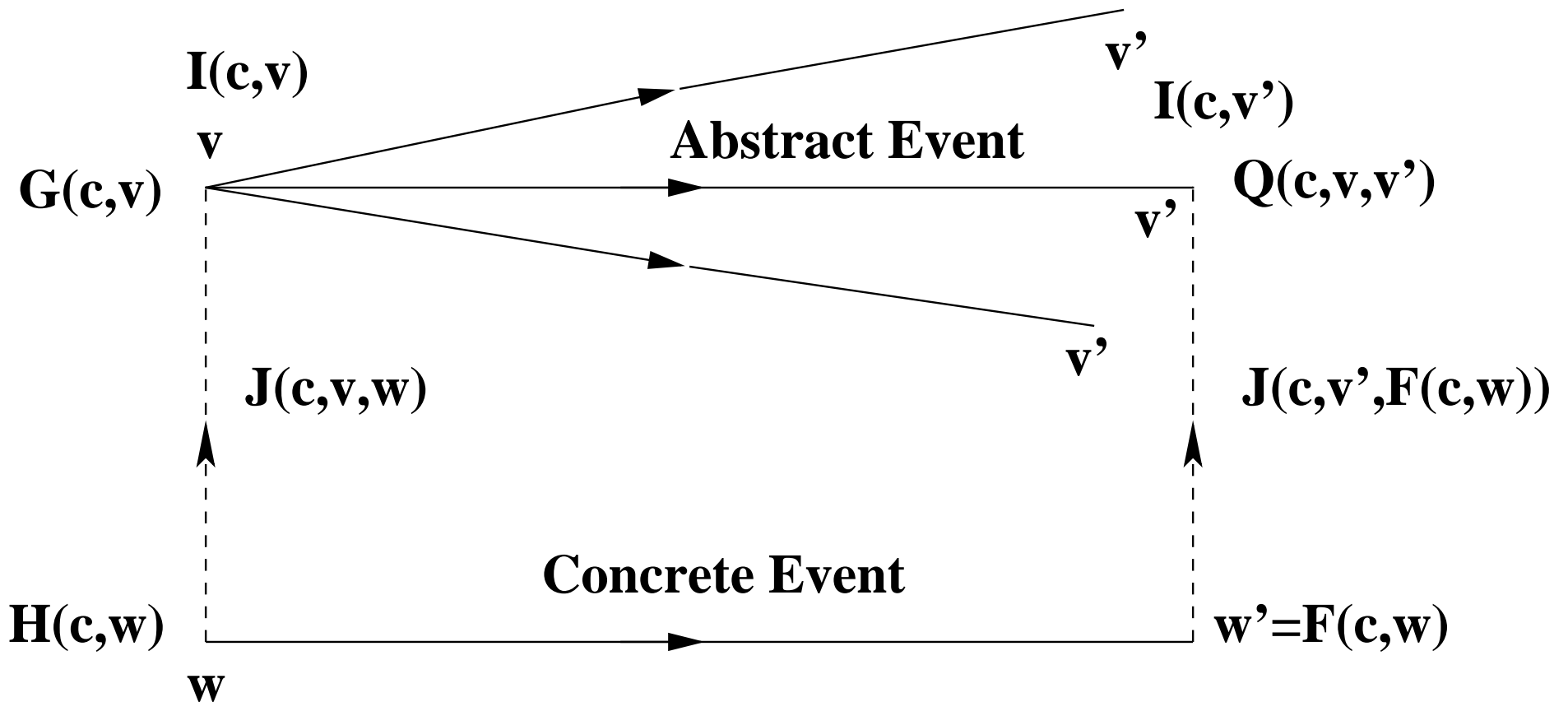
- **Deterministic Case**

Properties ┆ Modified concrete invariant	INI_INV_REF
--	-------------

- For all events (except init)
- New events refine an implicit non-guarded event with skip action
- **Deterministic Case**

Properties Abstract invariants Concrete invariants Concrete guards of the event ┆ Modified concrete invariant	INV_REF
--	---------

- The **abstract** event is **non-deterministic**
- The **concrete** event is **deterministic**



(abstract_)E

when

$G(c, v)$

then

$v := Q(c, v, v')$

end

(concrete_)E

when

$H(c, w)$

then

$w := F(c, w)$

end

$P(c)$

$I(c, v)$

$J(c, v, w)$

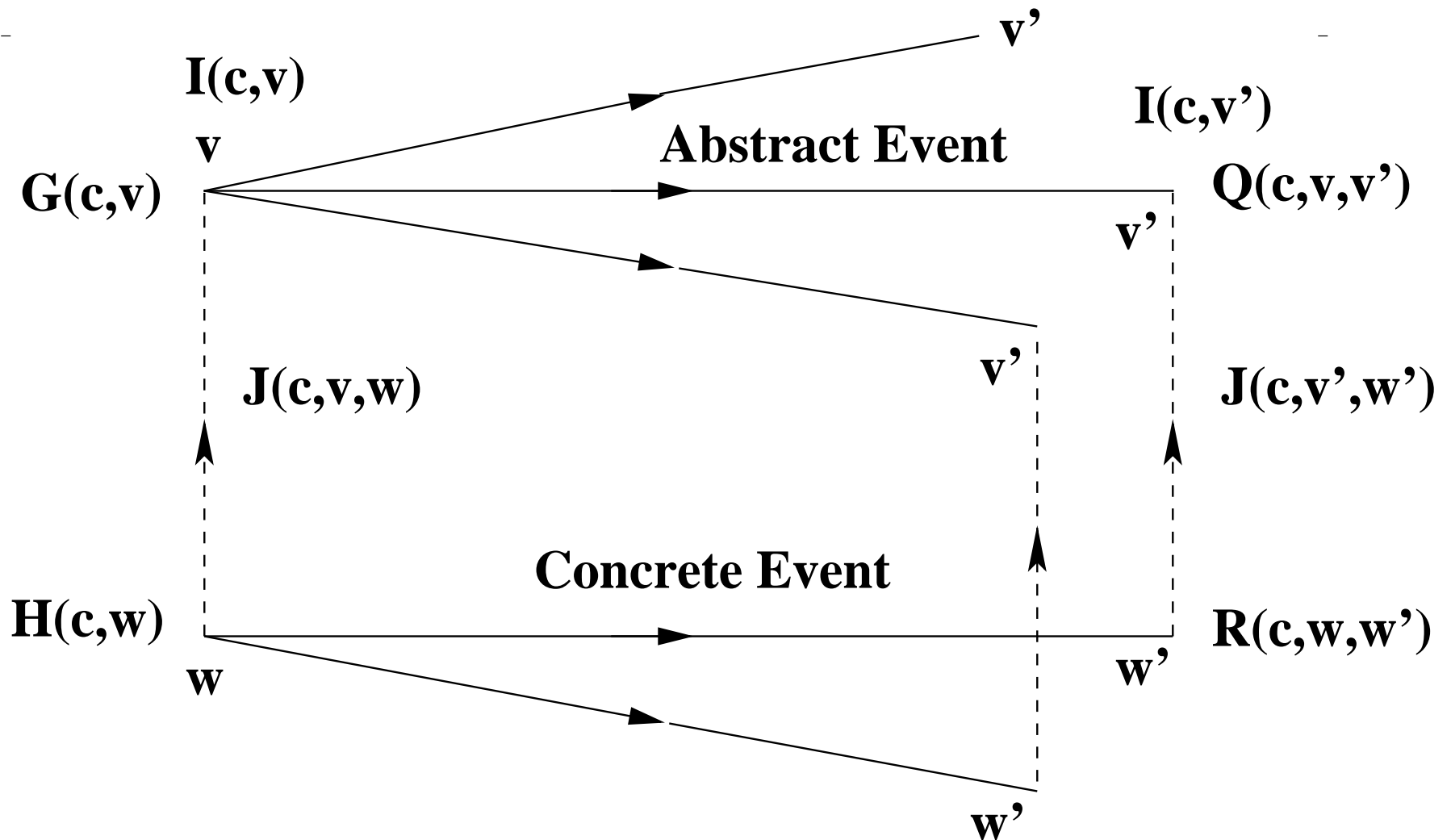
$H(c, w)$

⊢

$\exists v' \cdot (Q(c, v, v') \wedge J(c, v', F(c, w)))$

INV_REF

- Both **abstract** and **concrete** events are **non-deterministic**



(abstract_)E

when

$G(c, v)$

then

$v :| Q(c, v, v')$

end

(concrete_)E

when

$H(c, w)$

then

$w :| R(c, w, w')$

end

$P(c)$

$I(c, v)$

$J(c, v, w)$

$H(c, w)$

$R(c, w, w')$

⊢

$\exists v' \cdot (Q(c, v, v') \wedge J(c, v', w'))$

INV_REF

- For all events (except init)
- New events refine an implicit non-guarded event with skip action
- **Non-deterministic Case**

Properties Abstract invariants Concrete invariants Concrete guards of the event Before-after predicate of concrete event ⊢ $\exists v' \cdot \left(\begin{array}{l} \text{Before-after predicate of abstract event} \wedge \\ \text{Conjunction of all modified concrete invariants} \end{array} \right)$	INV_REF
--	---------

where v' denotes the **after values** of the **abstract variables** v

- Some abstract variables are still used in the refinement
- **Deterministic Case**

Properties Abstract invariants Concrete invariants Concrete guards of the event \vdash Equality of abstract and concrete expressions assigned to common variables	EQL_REF
---	---------

- Some abstract variables v are still used in the refinement
- **Non-deterministic Case**
- Rename the common variables v to $v1$ in the abstract state
- Add the gluing invariant $v = v1$
- Proceed as in the normal non-deterministic case

prp0_1
 prp0_2
 prp0_3
 inv0_1
 inv0_2
 inv1_1
 inv1_2
 additional gluing
 invariants
 concrete
 guards
 concrete before-after
 predicates

⊢

$$\exists i1', f1' \cdot \left(\begin{array}{l} \text{abstract before-after} \\ \text{predicates} \\ \text{modified concrete} \\ \text{invariants} \end{array} \right)$$

$n \in \mathbb{N}$
 $0 < n$
 $a \in 1 .. n \rightarrow D$
 $i \in 0 .. n$
 $f \in 1 .. i \rightarrow D$
 $m \in 0 .. n$
 $b = (1 .. m) \triangleleft a$

$$i = i1$$

$$f = f1$$

$r \neq \text{working}$

$s \neq \text{working}$

$i' = m$

$f' = b$

⊢

$$\exists i1', f1' \cdot \left(\begin{array}{l} i1' \in 0 .. n \\ f1' = (1 .. i') \triangleleft a \\ i' = i1' \\ f' = f1' \end{array} \right)$$

One can eliminate the quantification by using the **one-point rule**

$$n \in \mathbb{N}$$

$$0 < n$$

$$a \in 1..n \rightarrow D$$

$$i \in 0..n$$

$$f \in 1..i \rightarrow D$$

$$m \in 0..n$$

$$b = (1..m) \triangleleft a$$

$$i = i1$$

$$f = f1$$

$$r \neq \text{working}$$

$$s \neq \text{working}$$

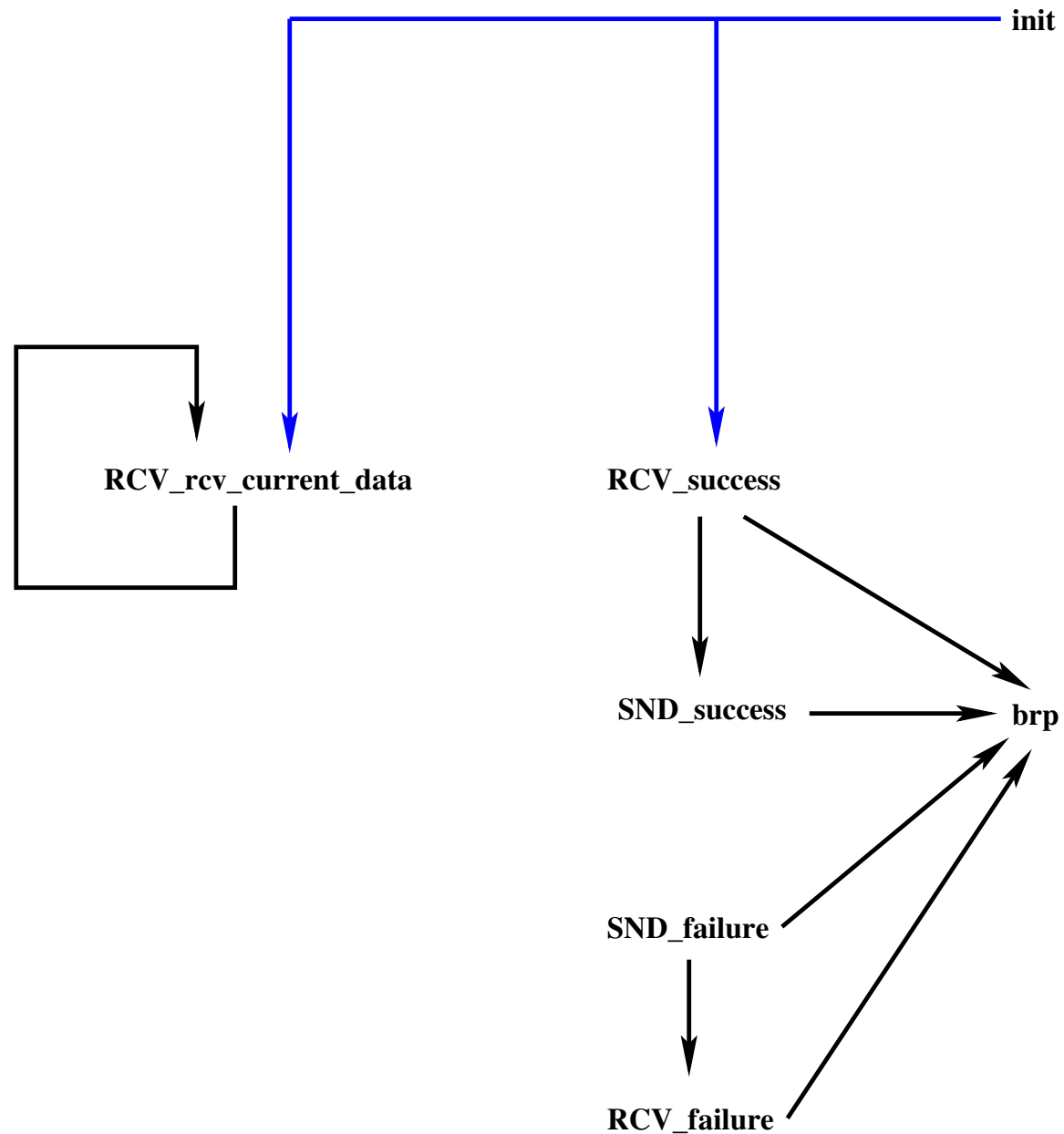
$$i' = m$$

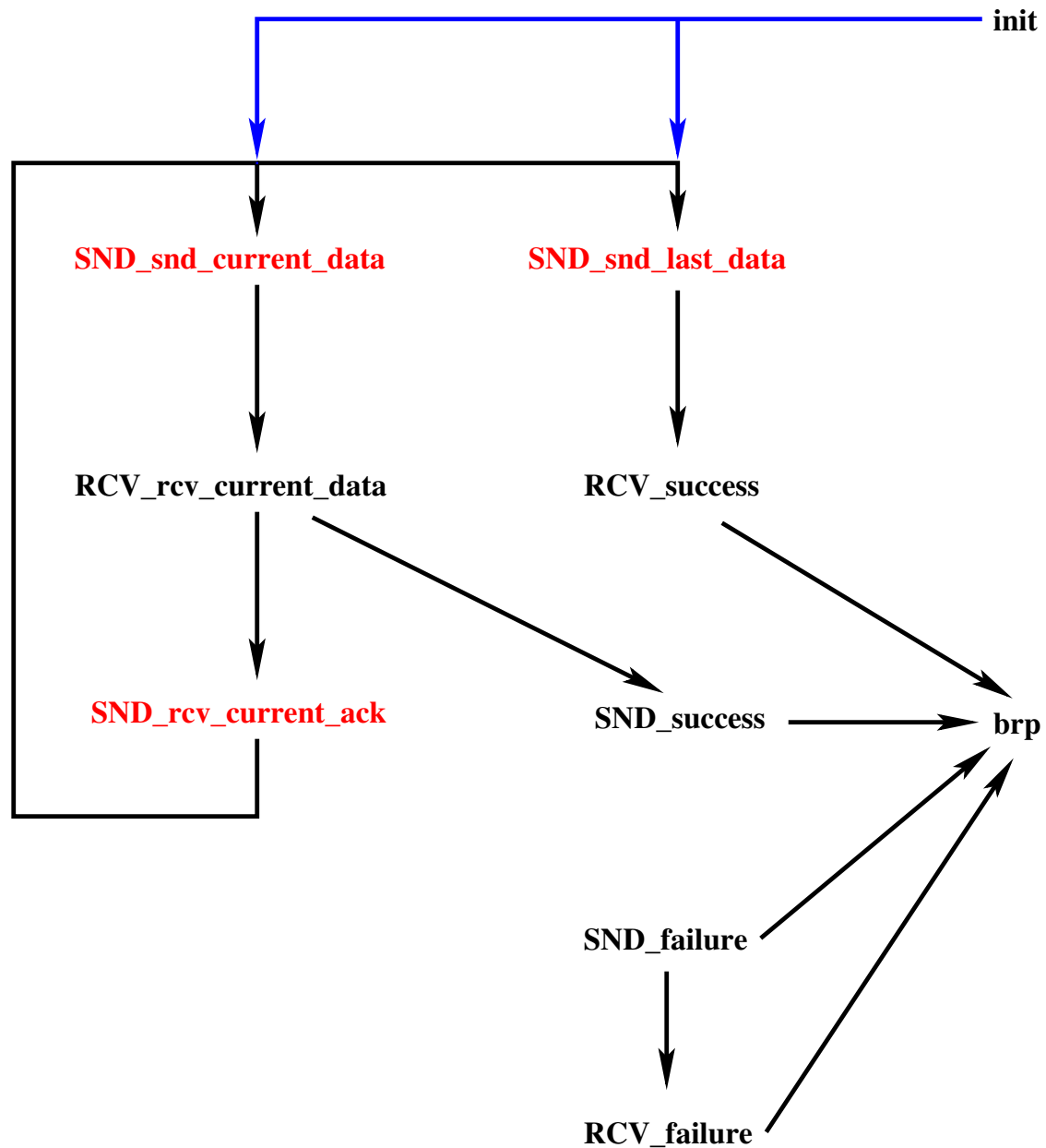
$$f' = b$$

┆

$$i' \in 0..n$$

$$f' = (1..m) \triangleleft a$$





- Variable p is the Sender **pointer** sent to the Receiver
- Variable d is the **data sent** to the Receiver
- Variable w is an **activation bit**

variables:

$m, b,$

$r, s,$

p, d, w

inv2_1: $p \in 0 .. n - 1$

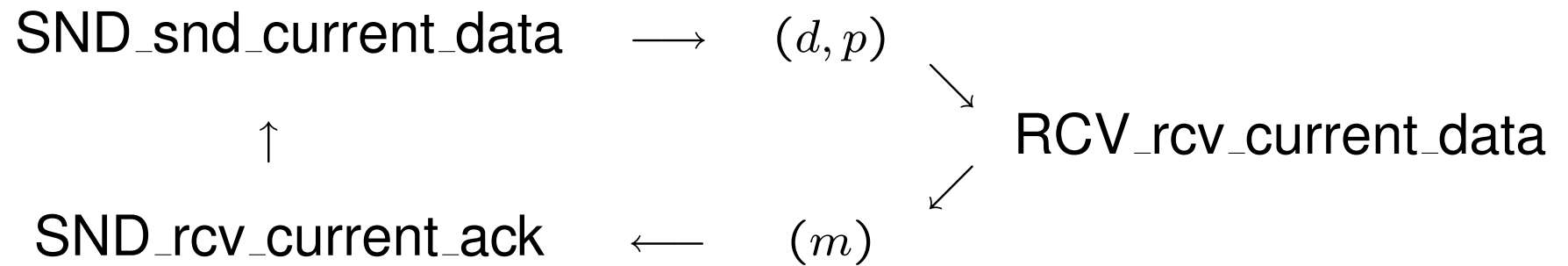
inv2_2: $d \in D$

inv2_3: $w \in \{0, 1\}$

inv2_4: $m \in p .. p + 1$

inv2_5: $m = p \wedge w = 0 \Rightarrow d = a(p + 1)$

inv2_6: $p + 1 = n \wedge m = p + 1 \Rightarrow r = success$



init

$m := 0$

$b := \emptyset$

$r := \textit{working}$

$s := \textit{working}$

$p := 0$

$d := a(1)$

$w := 1$

brp

when

$r \neq \textit{working}$

$s \neq \textit{working}$

then

skip

end

- **New Events**: the Sender prepares **data d** and **pointer p** to be sent

SND_snd_current_data

when

$s = \textit{working}$

$w = 1$

$p + 1 < n$

then

$d := a(p + 1)$

$w := 0$

end

SND_snd_last_data

when

$s = \textit{working}$

$w = 1$

$p + 1 = n$

then

$d := a(p + 1)$

$w := 0$

end

- These events clearly **refine skip** and maintain the following invariant

inv2_5: $m = p \wedge w = 0 \Rightarrow d = a(m + 1)$

- The Receiver receives data d and pointer p . It sends **pointer m**

RCV_rcv_current_data

when

$r = \textit{working}$

$m + 1 < n$

$m = p$

$w = 0$

then

$m := m + 1$

$b := b \cup \{m + 1 \mapsto d\}$

end

RCV_success

when

$r = \textit{working}$

$m + 1 = n$

$m = p$

$w = 0$

then

$r := \textit{success}$

$m := m + 1$

$b := b \cup \{n \mapsto d\}$

end

(abstract-)RCV_rcv_current_data

when

$r = \textit{working}$

$m + 1 < n$

then

$m := m + 1$

$b := b \cup \{m + 1 \mapsto a(m + 1)\}$

end

RCV_rcv_current_data

when

$r = \textit{working}$

$m + 1 < n$

$m = p$

$w = 0$

then

$m := m + 1$

$b := b \cup \{m + 1 \mapsto d\}$

end

This invariant helps proving **event refinement**

inv2_5: $m = p \wedge w = 0 \Rightarrow d = a(m + 1)$

```
(abstract-)RCV_success
when
   $r = \textit{working}$ 
   $m + 1 = n$ 
then
   $r := \textit{success}$ 
   $m := m + 1$ 
   $b := b \cup \{n \mapsto a(n)\}$ 
end
```

```
(concrete-)RCV_success
when
   $r = \textit{working}$ 
   $m + 1 = n$ 
   $m = p$ 
   $w = 0$ 
then
   $r := \textit{success}$ 
   $m := m + 1$ 
   $b := b \cup \{n \mapsto d\}$ 
end
```

Observe **guard strengthening**

- The first event is **new**. It clearly **refines skip**
- The Sender receives acknowledgment (pointer m)

SND_rcv_current_ack

when

$s = \textit{working}$

$w = 0$

$p + 1 < n$

$m = p + 1$

then

$w := 1$

$p := p + 1$

end

SND_success

when

$s = \textit{working}$

$w = 0$

$p + 1 = n$

$m = p + 1$

then

$s := \textit{success}$

end

(abstract-)SND_success

when

$s = \textit{working}$

$r = \textit{success}$

then

$s := \textit{success}$

end

(concrete-)SND_success

when

$s = \textit{working}$

$w = 0$

$p + 1 = n$

$m = p + 1$

then

$s := \textit{success}$

end

- The presence of **inv2_6** ensures that the **guard is strengthened**

inv2_6: $p + 1 = n \wedge m = p + 1 \Rightarrow r = \textit{success}$

carrier sets: $D, STATUS$

constants: a, n, M

- Constant M denotes the **maximum number of retries**

prp3_1: $M \in \mathbb{N}$

- Variables db and ab are **channel activation bits**
- Variable l is the **last data indicator** sent to the Receiver
- Variable c is the **retry counter**

variables:

$m, b,$
 $r, s,$
 p, d, w, db, ab, c

inv2_1: $db \in \{0, 1\}$

inv3_2: $ab \in \{0, 1\}$

inv3_3: $l \in \{0, 1\}$

inv3_4: $c \in 0 .. M + 1$

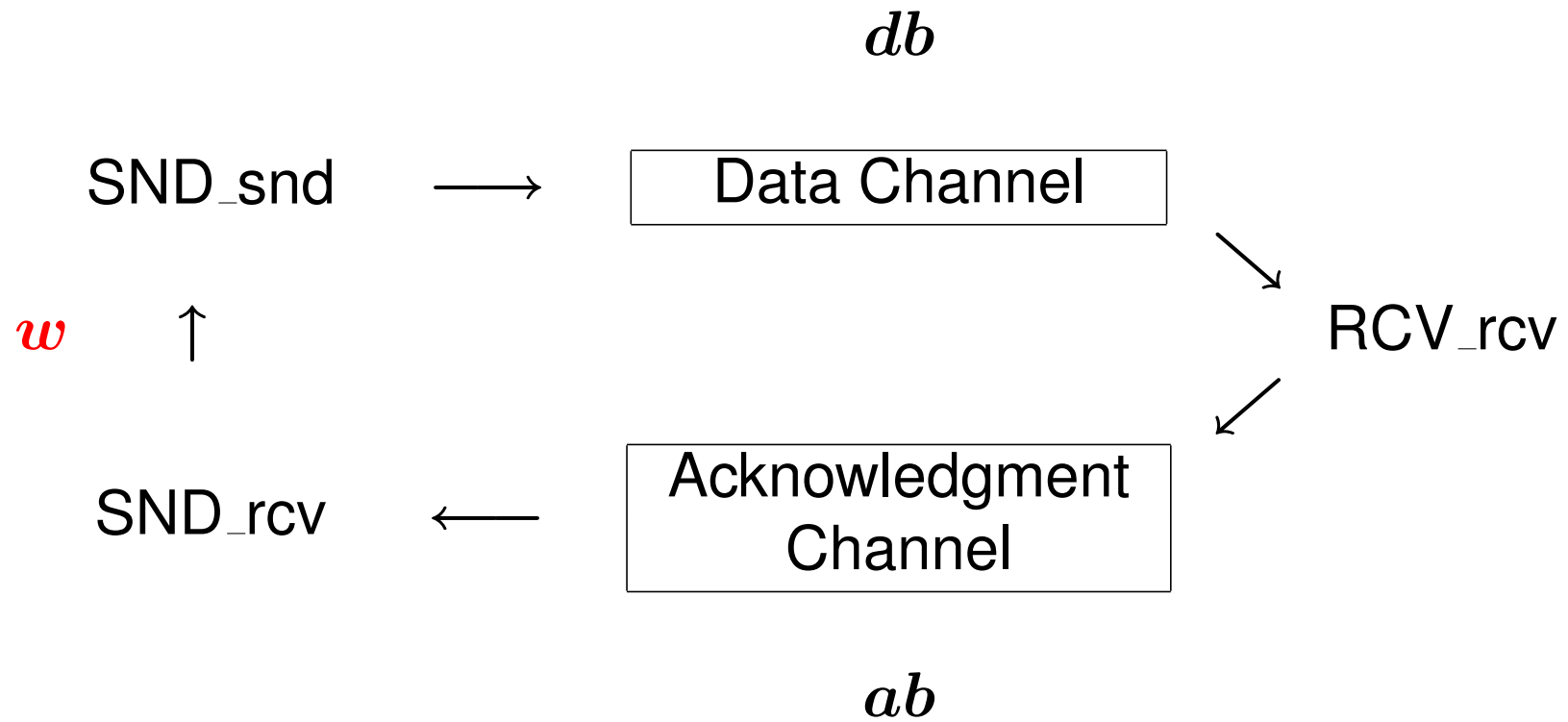
- **At most** one activation bit is "on" at a time

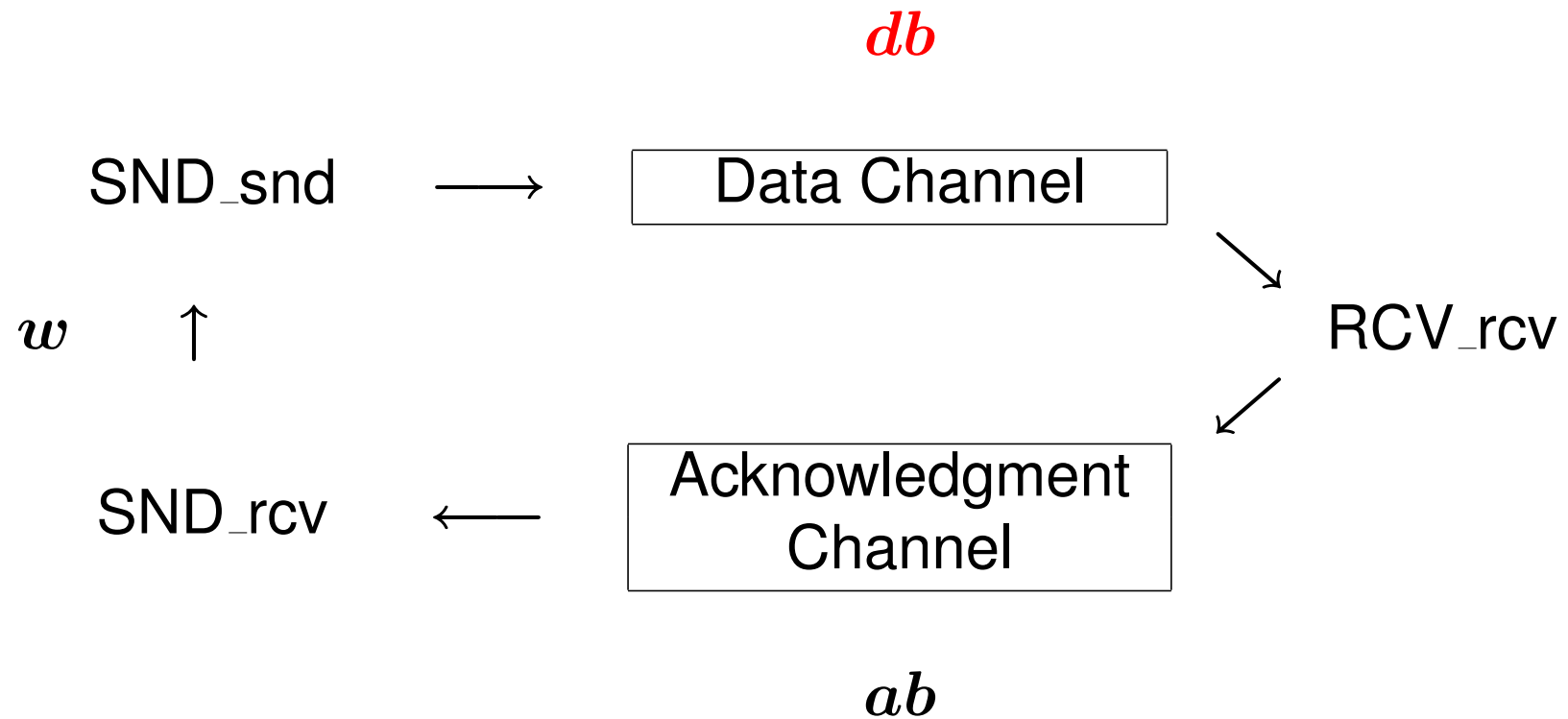
$$\mathbf{inv3_5:} \quad db = 1 \Rightarrow ab = 0$$

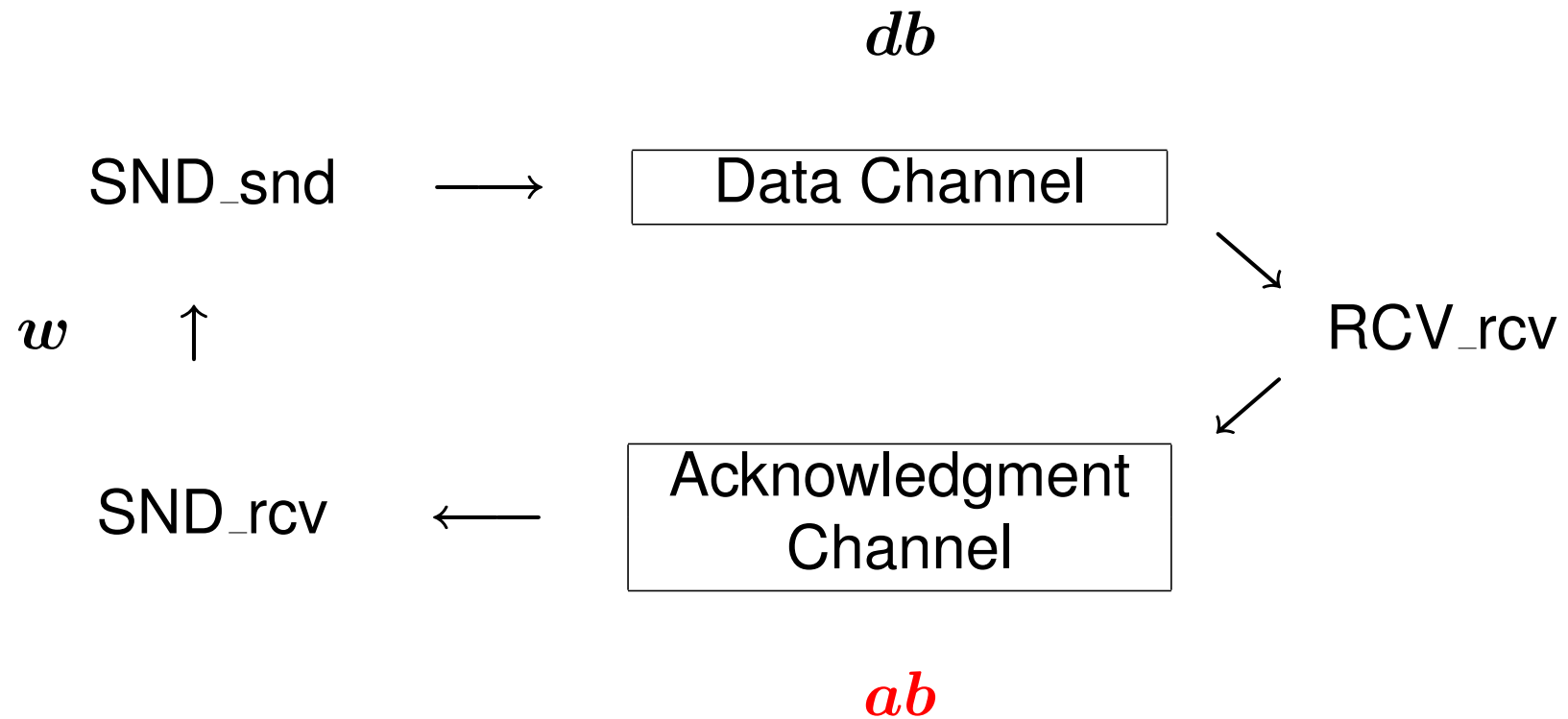
$$\mathbf{inv3_6:} \quad w = 1 \Rightarrow db = 0$$

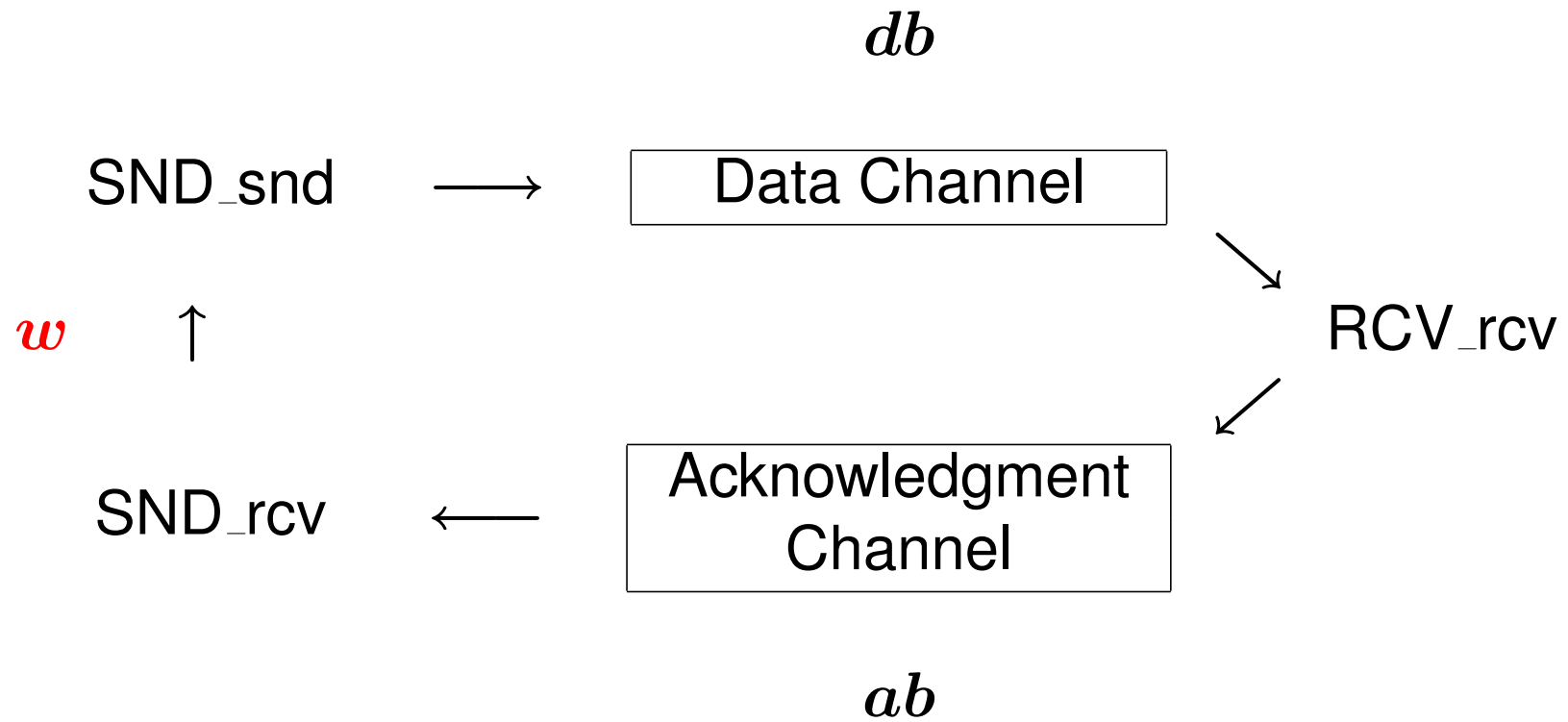
$$\mathbf{inv3_7:} \quad w = 1 \Rightarrow ab = 0$$

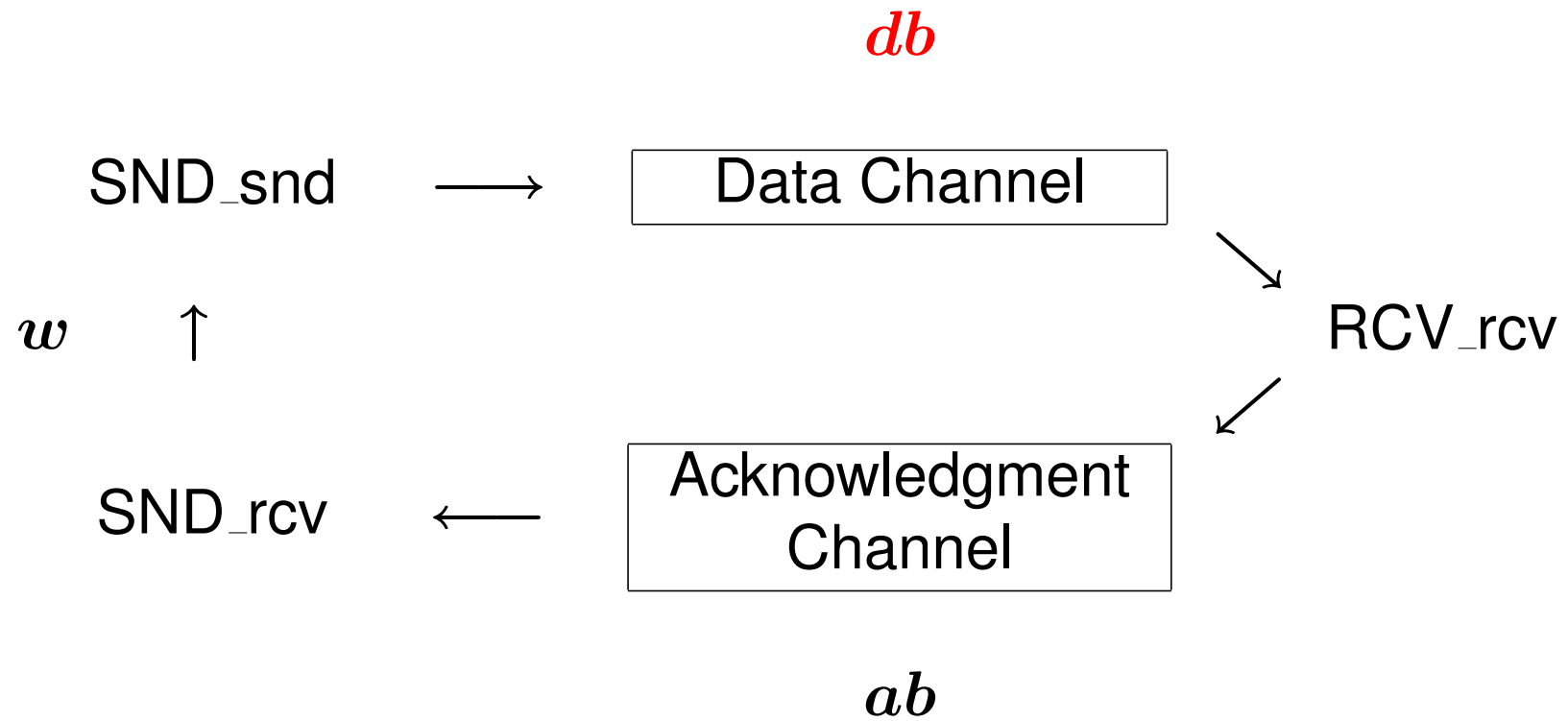
$$\mathbf{inv3_8:} \quad ab = 1 \Rightarrow m \neq p$$

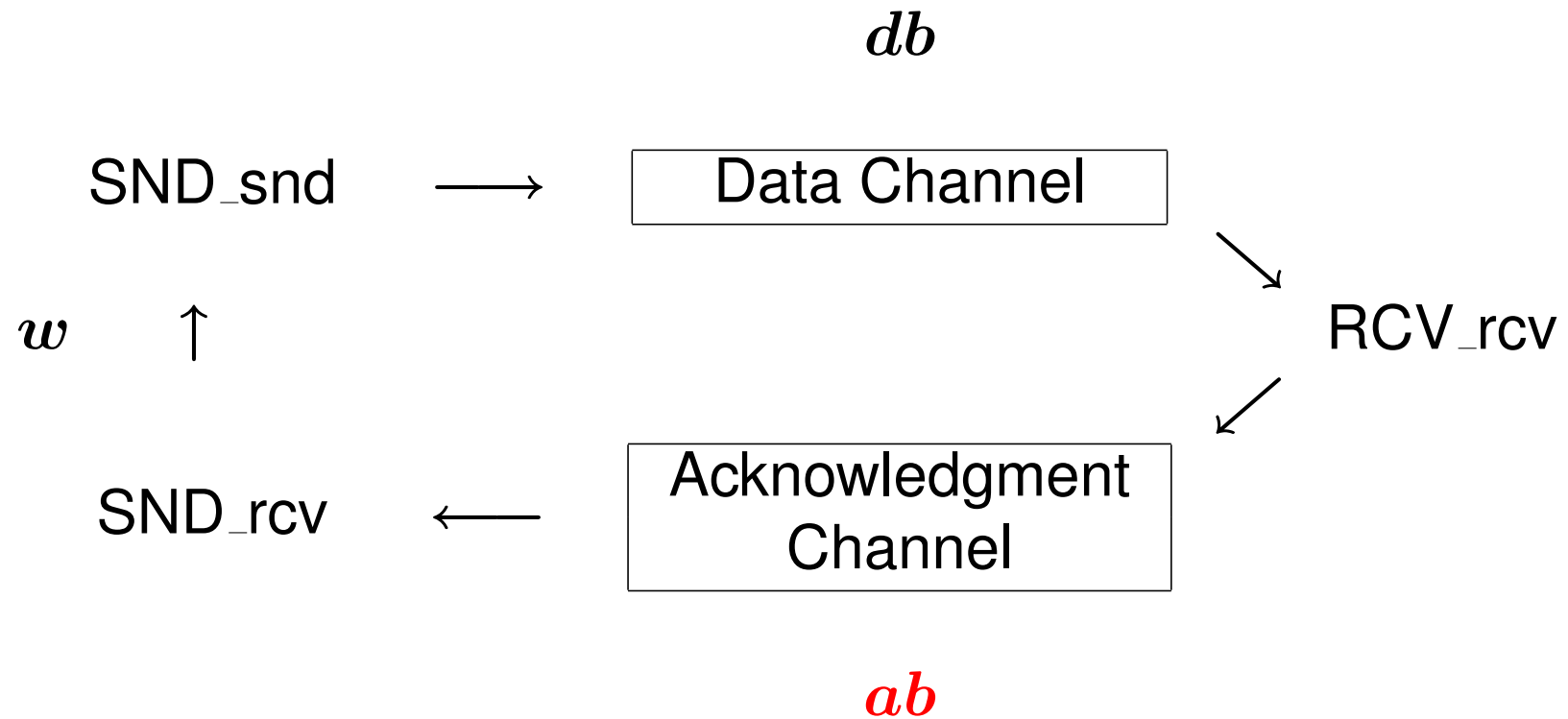


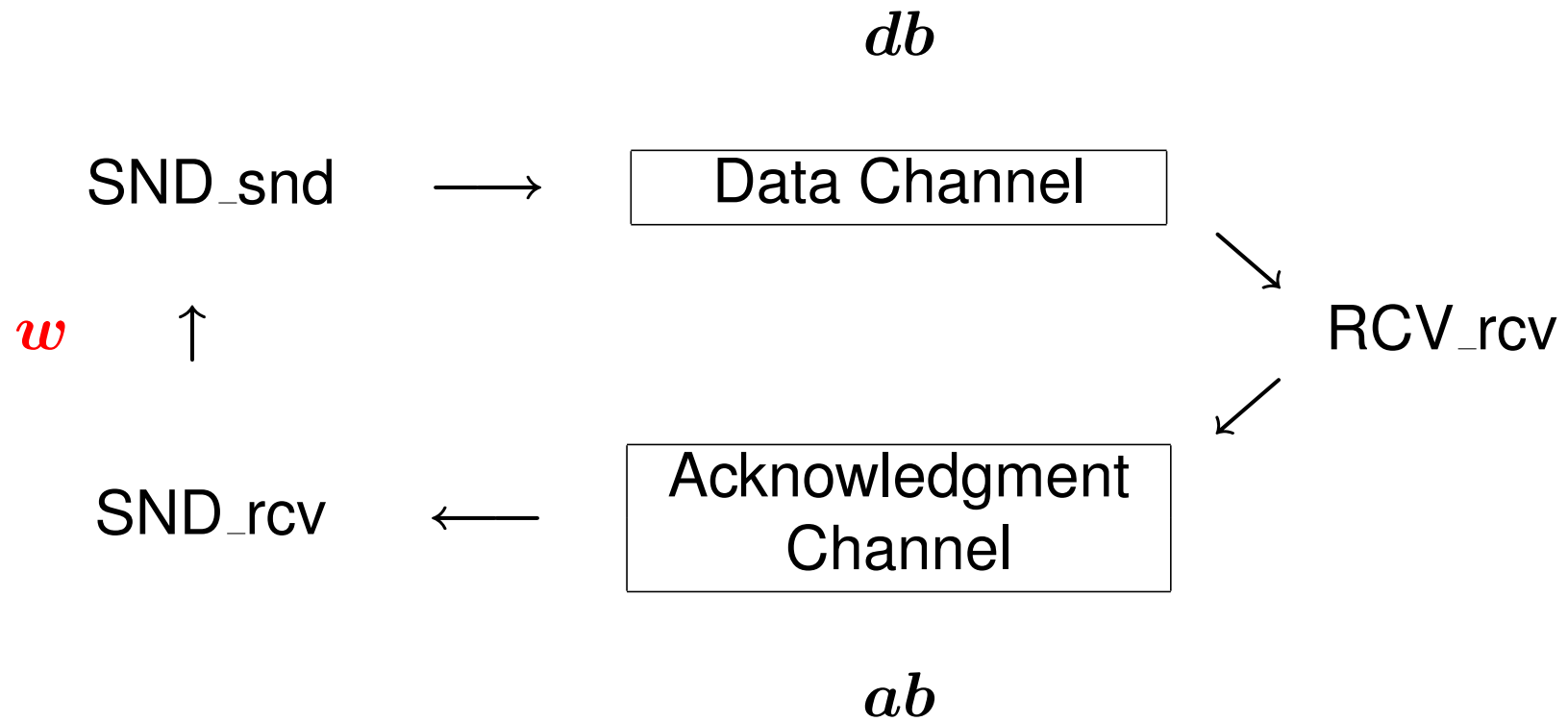


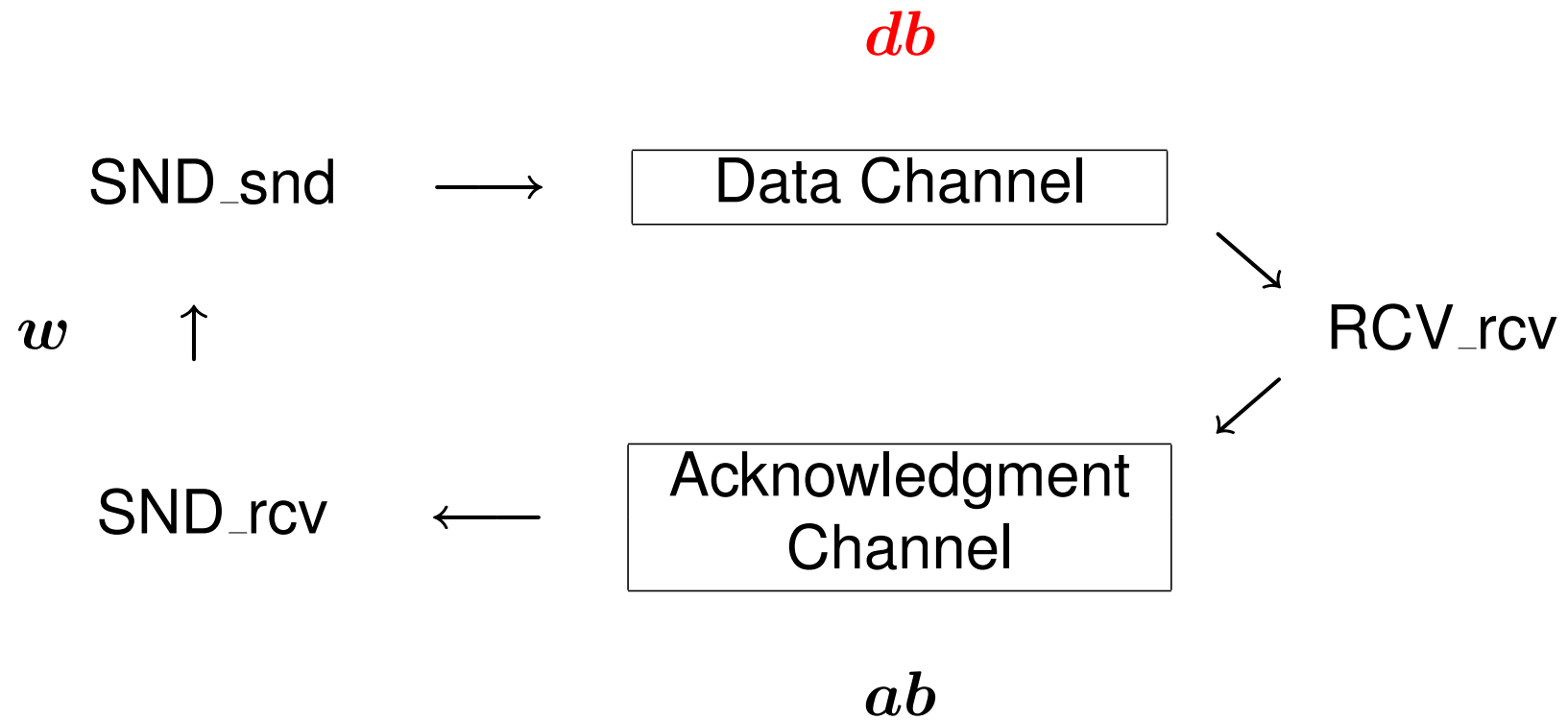


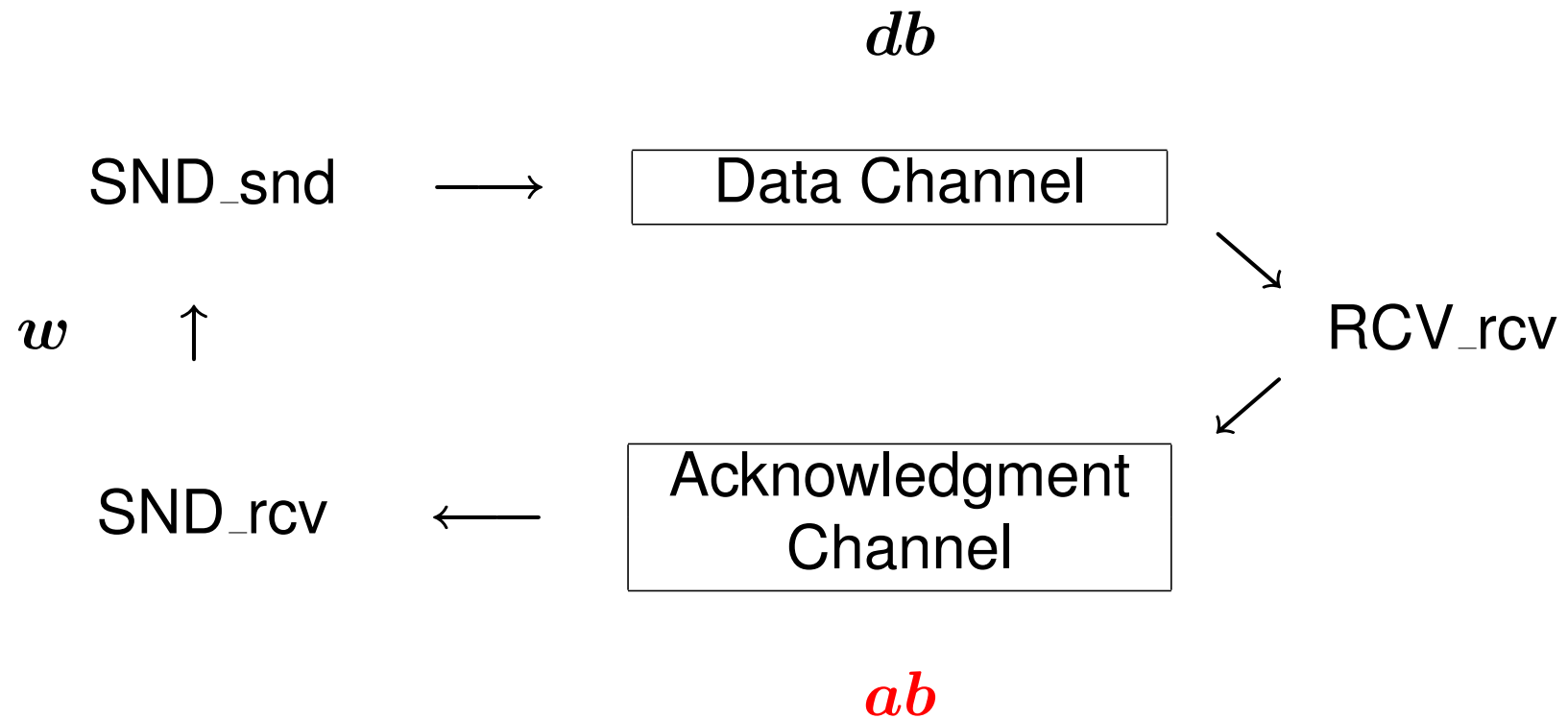


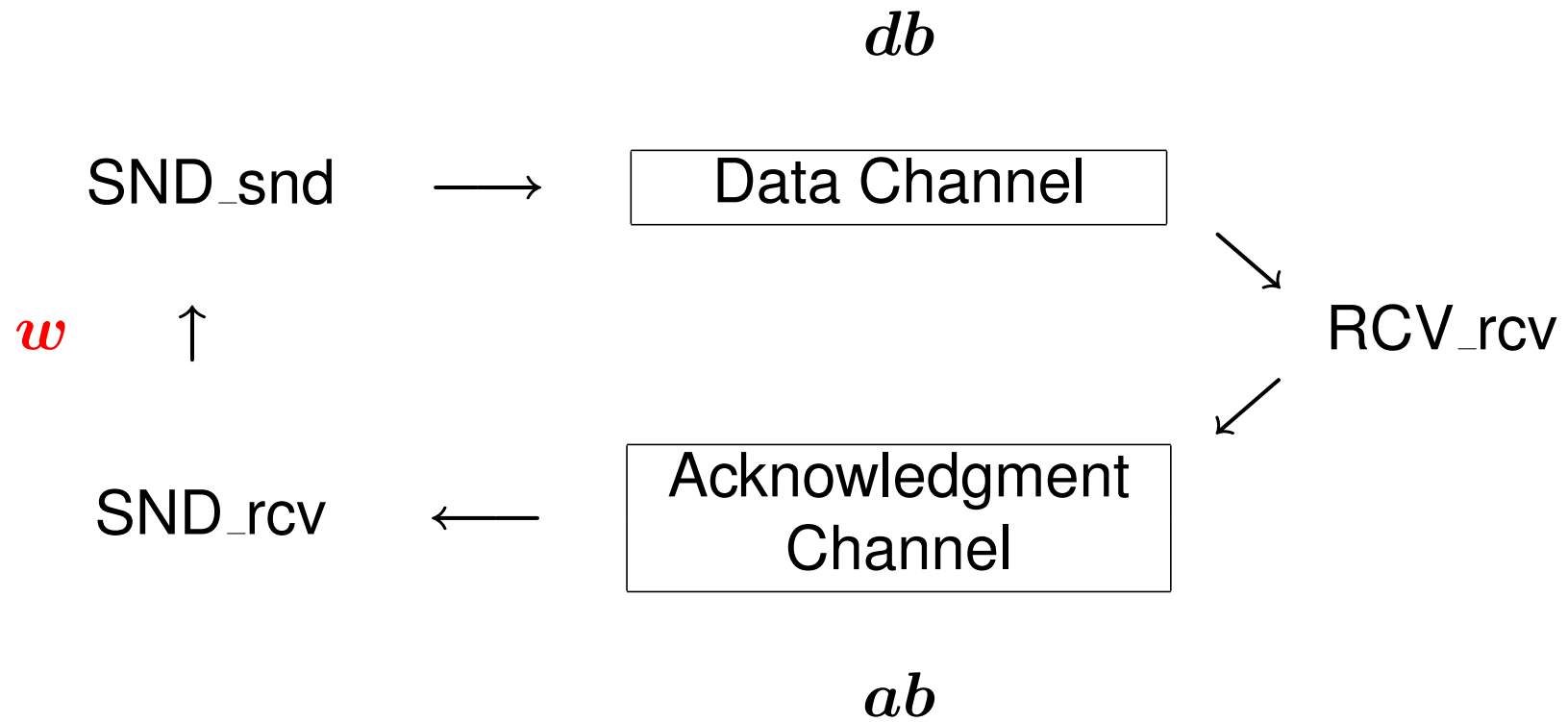












- These invariants define the **last data indicator**

$$\begin{array}{l} \text{inv3_9:} \\ \Rightarrow \\ \end{array} \quad db = 1 \wedge m = p \wedge l = 0 \\ m + 1 < n$$

$$\begin{array}{l} \text{inv3_10:} \\ \Rightarrow \\ \end{array} \quad db = 1 \wedge m = p \wedge l = 1 \\ m + 1 = n$$

- This invariant defines **when the Sender fails**

$$\text{inv3_11:} \quad c = M + 1 \Leftrightarrow s = \textit{failure}$$

init

$m := 0$

$b := \emptyset$

$r := \textit{working}$

$s := \textit{working}$

$p := 0$

$c := 0$

$d := a(1)$

$db := 0$

$ab := 0$

$w := 1$

$l := 0$

brp

when

$r \neq \textit{working}$

$s \neq \textit{working}$

then

skip

end

SND_snd_current_data

when

$s = \textit{working}$

$w = 1$

$p + 1 < n$

then

$d := a(p + 1)$

$w := 0$

$db := 1$

$l := 0$

end

SND_snd_last_data

when

$s = \textit{working}$

$w = 1$

$p + 1 = n$

then

$d := a(p + 1)$

$w := 0$

$db := 1$

$l := 1$

end

SND_time_out_current

when

$s = \textit{working}$

$w = 0$

$ab = 0$

$db = 0$

$c < M$

then

$c := c + 1$

$w := 1$

end

SND_failure

when

$s = \textit{working}$

$w = 0$

$ab = 0$

$db = 0$

$c = M$

then

$c := c + 1$

$s := \textit{failure}$

end

- Sender aborts after $M + 1$ tries

Rcv_rcv_current_data

when

$r = \textit{working}$

$m = p$

$db = 1$

$l = 0$

then

$m := m + 1$

$b := b \cup \{m + 1 \mapsto d\}$

$db := 0$

$ab := 1$

end

Rcv_success

when

$r = \textit{working}$

$m = p$

$db = 1$

$l = 1$

then

$r := \textit{success}$

$m := m + 1$

$b := b \cup \{n \mapsto d\}$

$db := 0$

$ab := 1$

end

Reminder: l is the last data indicator

```
RCV_retry
  when
     $m \neq p$ 
     $db = 1$ 
  then
     $db := 0$ 
     $ab := 1$ 
  end
```

```
RCV_failure
  when
     $r = \textit{working}$ 
     $c = M + 1$ 
  then
     $r := \textit{failure}$ 
  end
```

SND_rcv_current_ack

when

s = working

ab = 1

p + 1 < n

then

c := 0

ab := 0

w := 1

p := p + 1

end

SND_success

when

s = working

ab = 1

p + 1 = n

then

c := 0

ab := 0

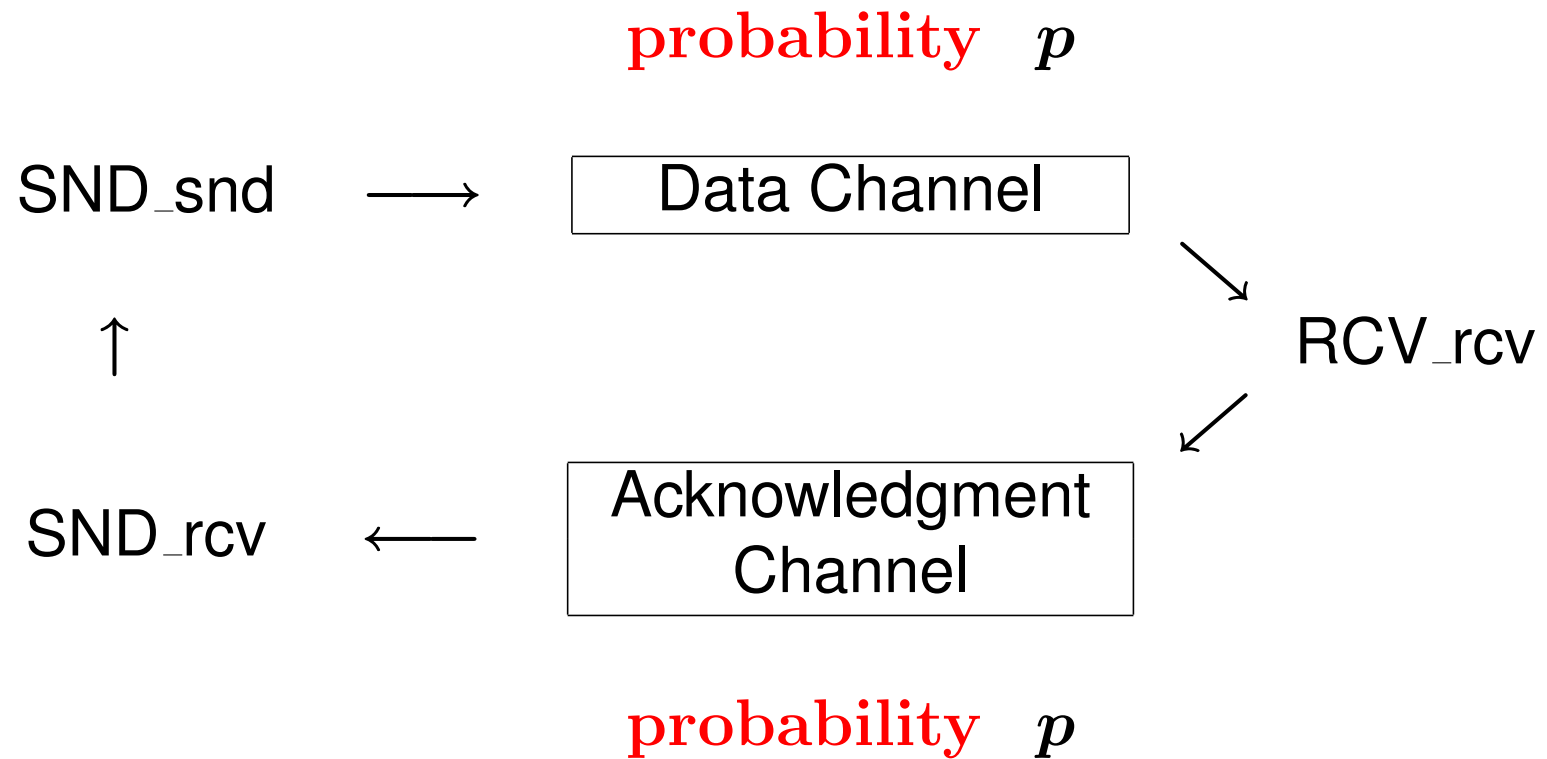
s := success

end

- Daemons are **breaking the channels**

```
DMN_data
  when
    db = 1
  then
    db := 0
  end
```

```
DMN_ack
  when
    ab = 1
  then
    ab := 0
  end
```



Failure on one channel p

Failure on one channel p

Success on one channel $1 - p$

Failure on one channel	p
Success on one channel	$1 - p$
Success on both channels	$(1 - p)^2$

Failure on one channel	p
Success on one channel	$1 - p$
Success on both channels	$(1 - p)^2$
Failure on one try	$1 - (1 - p)^2$

Failure on one channel	p
Success on one channel	$1 - p$
Success on both channels	$(1 - p)^2$
Failure on one try	$1 - (1 - p)^2$
Failure on $M + 1$ tries	$(1 - (1 - p)^2)^{M+1}$

Failure on one channel	p
Success on one channel	$1 - p$
Success on both channels	$(1 - p)^2$
Failure on one try	$1 - (1 - p)^2$
Failure on $M + 1$ tries	$(1 - (1 - p)^2)^{M+1}$
Success on $M + 1$ tries	$1 - (1 - (1 - p)^2)^{M+1}$

Failure on one channel	p
Success on one channel	$1 - p$
Success on both channels	$(1 - p)^2$
Failure on one try	$1 - (1 - p)^2$
Failure on $M + 1$ tries	$(1 - (1 - p)^2)^{M+1}$
Success on $M + 1$ tries	$1 - (1 - (1 - p)^2)^{M+1}$
Success for n data	$(1 - (1 - (1 - p)^2)^{M+1})^n$

Failure on one channel	p
Success on one channel	$1 - p$
Success on both channels	$(1 - p)^2$
Failure on one try	$1 - (1 - p)^2$
Failure on $M + 1$ tries	$(1 - (1 - p)^2)^{M+1}$
Success on $M + 1$ tries	$1 - (1 - (1 - p)^2)^{M+1}$
Success for n data	$(1 - (1 - (1 - p)^2)^{M+1})^n$
$p = .1, M = 5, n = 100$.995