Bounded Re-transmission Protocol

Jean-Raymond Abrial
(edited by Thai Son Hoang)

Department of Computer Science
Swiss Federal Institute of Technology Zürich (ETH Zürich)

Bucharest DEPLOY 2-day Course, 14th-16th July, 2010

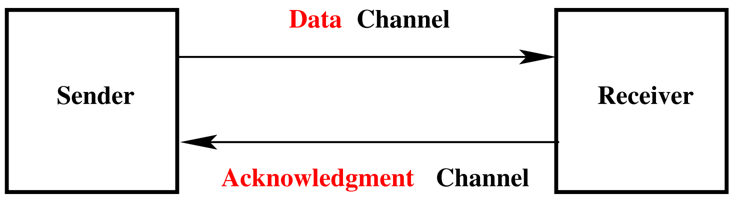
Purpose of this Lecture

- 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

Outline

The Bounded Retransmission Protocol

- 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

Requirements (1)

The goal of the BRP is to totally or partially transfer a certain non-empty original sequential file from one site to another.	FUN1

A total transfer means that the transmitted file is a copy of the original one.	FUN2

A partial transfer means that the transmitted file is a genuine prefix of the original one.	FUN3

Unreliability of the Communications (1)

- 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 *dl*

- This occurs if the Sender has not received an acknowledgment in the meantime

Unreliability of the Communications (2)

- *dl* 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 Sender sends an alternating bit together with a new data

- This ensures that the Receiver does not confuse (?) a new data with a retransmitted one.

Abortion of Protocol at the Sender Site

- The Sender can re-transmit the same data at most $MAX + 1$ times

- After this, the Sender decides to abort

- How does the Receiver know that the Sender aborted?

Abortion of Protocol at the Receiver Site

- Each time the Receiver receives a new piece of data, it starts a timer

- The timer wakes up the Receiver after a delay $(MAX + 1) \times dl$

- This occurs if the Receiver 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.

Final Situation of the Protocol

- 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

Requirements (2)

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	FUN4

When the Sender believes that the protocol has terminated successfully then the Receiver believes so too.	FUN5

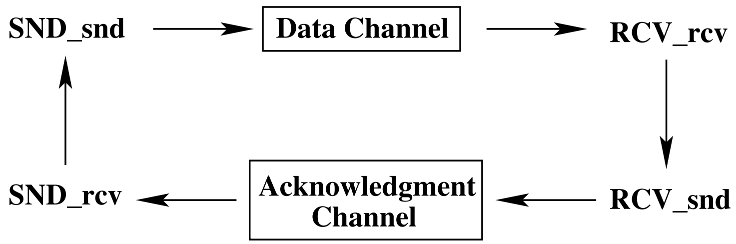
Requirements (3)

However, it is possible for the Sender to believe that the protocol has aborted while the Receiver believes that it has terminated successfully.	FUN6

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.	FUN7

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.	FUN8

Pseudo-code for the Protocol



The Sender sends Data

```
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
```


The Receiver Receives Data

```
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
    Reset Data Channel Interrupt;
    Wake up RCV_snd;
  end
```


The Receiver sends Acknowledgment

```
RCV_snd
  when
    RCV_snd is waken up
  then
    Activate Acknowledgment Channel;
  end
```

The Sender Receives Acknowledgment

```
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
```


Timer Interrupt Occurs at Sender's Site

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


Timer Interrupt occurs at Receiver's Site

```
RCV_timer
  when
    Receiver's timer interrupt occurs
  then
    Abort protocol on Receiver's site
  end
```


About the Pseudo-code

- Quite often, protocol 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

Outline

Refinement Strategy

(1) FUN1, FUN2, FUN3: partial transmission of the file in one shot.

(2) FUN4 to FUN8: each participant has access to the other

(3) Introducing unreliable channels and timers.

(4) Optimize protocol

Reminder (1)

The goal of the BRP is to totally or partially transfer a certain non-empty original sequential file from one site to another.	FUN1
A total transfer means that the transmitted file is a copy of the original one.	FUN2
A partial transfer means that the transmitted file is a genuine prefix of the original one.	FUN3

Reminder (2)

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	FUN4
When the Sender believes that the protocol has terminated successfully then the Receiver believes so too.	FUN5

Reminder (3)

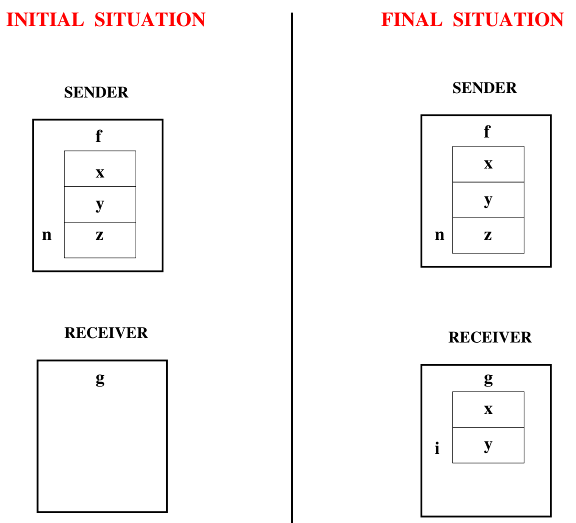
However, it is possible for the Sender to believe that the protocol has aborted while the Receiver believes that it has terminated successfully.	FUN6

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.	FUN7

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.	FUN8

Outline

The Sender and the Receiver: a First View



Initial Model: the Constants

- Set D denotes the objects in the files
- Constant n denotes the size of the non-empty file
- Constant f denotes the original file.

set: D

constants: n
f

axm0_1: $0 < n$

axm0_2: $f \in 1 \,..\, n \rightarrow D$

Initial Model: the Variables

- Variable i denotes the size of file g

- Variable g denotes the transmitted file.

variables: i
g

inv0_1: $i \in 0 \,..\, n$

inv0_2: $g \in 1 \,..\, i \rightarrow D$

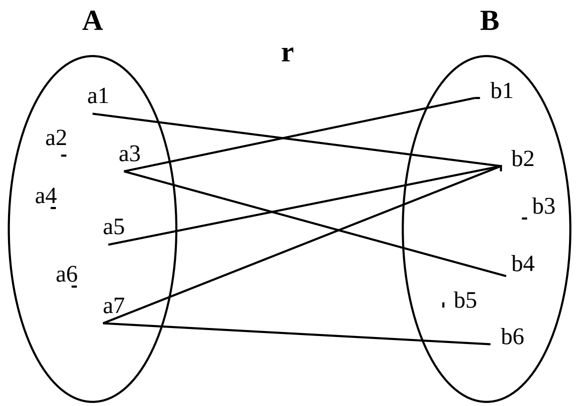
Reminder of Mathematical Conventions (1)

$x \in S$	set membership operator
$\mathbb{N}$	set of natural numbers: $\{0, 1, 2, 3, \ldots\}$
$a \,..\, b$	interval from a to b: $\{a, a+1, \ldots, b\}$ (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

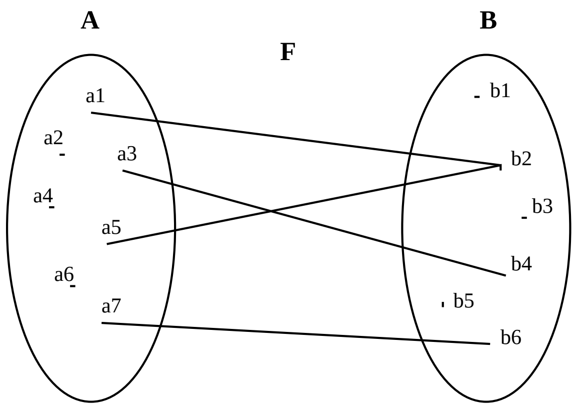
Reminder of Mathematical Conventions (2)

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

A Binary Relation r from a Set A to a Set B



A Partial Function F from a Set A to a Set B

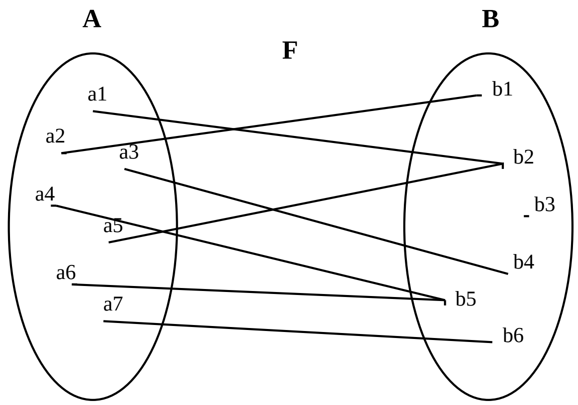


$$F = \{a1 \mapsto b2,\ a3 \mapsto b4,\ a5 \mapsto b2,\ a7 \mapsto b6\}$$

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

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

A Total Function F from a Set A to a Set B



$$\mathrm{dom}\,(F) = A$$

Initial model: a Single Event (no Protocol)

- 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
$i := 0$
$g := \varnothing$

brp
$i, g :\mid i' \in 0\,..\,n \;\wedge\; g' = (1\,..\,i') \lhd f$

- Operator $:\mid$ is to be read: "become such that ..."

Informal Meaning

brp
$$i, g :\mid \left(\begin{array}{l} i' \in 0\,..\,n \\ g' = (1\,..\,i') \lhd f \end{array} \right)$$

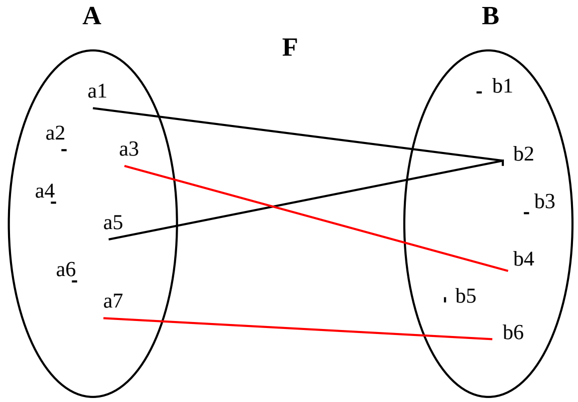
i and g are assigned any values i' and g' such that the following holds:

$$i' \in 0\,..\,n \;\wedge\; g' = (1\,..\,i') \lhd f$$

More Mathematical Conventions: Restrictions

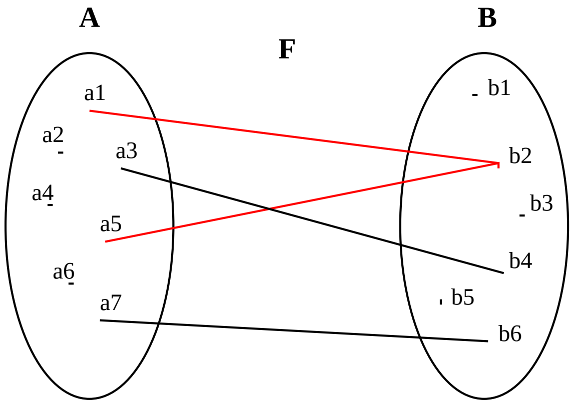
$s \lhd r$	domain restriction operator
$s \mathbin{⩤} r$	domain subtraction operator
$r \rhd t$	range restriction operator
$r \mathbin{⩥} t$	range subtraction operator

The Domain Restriction Operator



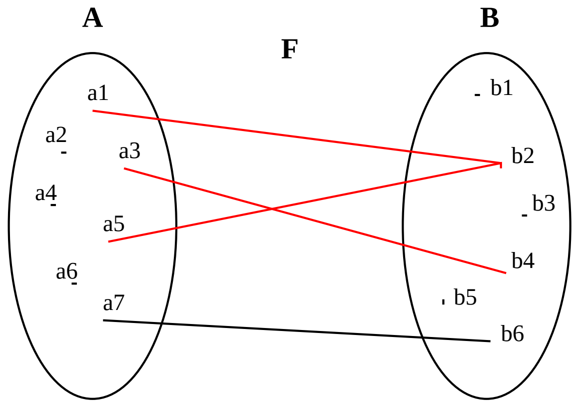
$\{a3,\ a7\} \lhd F$

The Domain Subtraction Operator



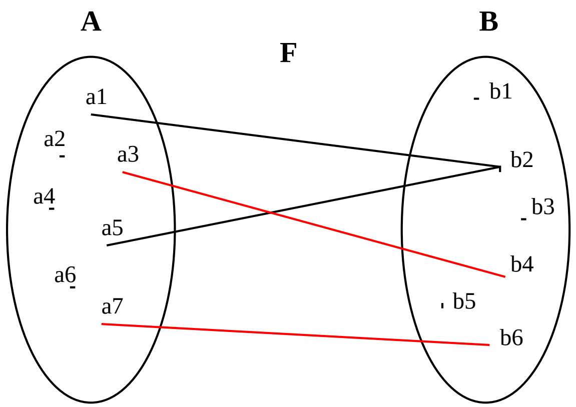
$\{a3,\ a7\} \mathbin{⩤} F$

The Range Restriction Operator



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

The Range Subtraction Operator

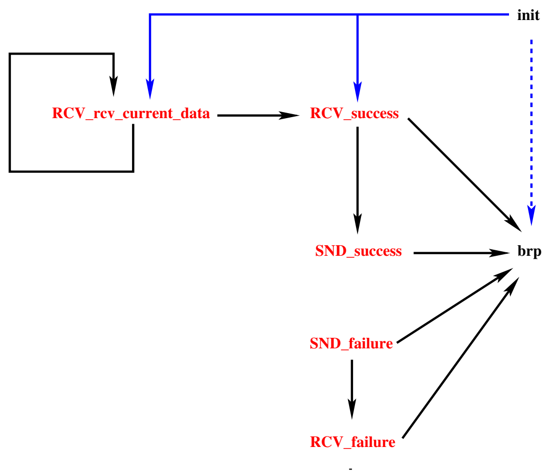


$F \rhd \{b2\}$

The Abstract Situation

Outline

First Refinement: Introducing New Events



About new Events in a Refinement

- 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)

First Refinement: Defining more Constants

set: *STATUS*

constants: . . .
working
success
failure

axm1_1: $STATUS = \{working, success, failure\}$

axm1_2: $working \neq success$

axm1_3: $working \neq failure$

axm1_4: $success \neq failure$

First Refinement: Variables

- Variables i and g are replaced by variables r and h
- Variable r denotes the size of the transmitted file
- Variable h denotes the transmitted file
- Variables s_st and r_st denote the status of the participants (Sender and Receiver respectively).

variables: r
h
s_st
r_st

First Refinement: Invariants (1)

- Variables h is a prefix of constant f (invariant **inv1_1** and **inv1_2**)

inv1_1: $r \in 0 \,..\, n$

inv1_2: $h = (1 \,..\, r) \lhd f$

First Refinement: Invariants (2)

- The typing of variables s_st and r_st is implicit (FUN4)

- Requirements FUN7 and FUN8 (Receiver's belief is true) is taken care invariant **inv1_3**

- Requirements FUN5 and FUN6 (Sender's status) are taken care by invariant **inv_4**

inv1_3: $r\_st = success \;\Leftrightarrow\; r = n$

inv1_4: $s\_st = success \;\Rightarrow\; r\_st = success$

Requirements Document | Initial Model
Formal Development | First Refinement
What about Probability | Second Refinement
Third Refinement

First Refinement: the Events (1)

- Initialisation

init
$r := 0$
$h := \varnothing$
$r\_st := working$
$s\_st := working$

Requirements Document | Initial Model
Formal Development | First Refinement
What about Probability | Second Refinement
Third Refinement

First Refinement: the Events (2)

- Event (concrete_)brp now does nothing

- We give witnesses for the abstract after values i' and g'

(abstract_)brp
$i, g :| \begin{pmatrix} i' \in 0\,..\,n \\ g' = (1\,..\,i') \lhd a \end{pmatrix}$

(concrete_)brp
when
$r\_st \neq working$
$s\_st \neq working$
with
i' : $i' = r$
g' : $g' = h$
then
skip
end

Requirements Document | Initial Model
Formal Development | First Refinement
What about Probability | Second Refinement
Third Refinement

First Refinement: the Events (3)

RCV_rcv_current_data
when
$r\_st = working$
$r + 1 < n$
then
$r := r + 1$
$h := h \cup \{r + 1 \mapsto f(r + 1)\}$
end

- This event is "cheating" (accessing constant f)
- This new event maintains invariant **inv1_3** and it refines skip

inv1_3: $r\_st = success \;\Leftrightarrow\; r = n$

First Refinement: the Events (4)

RCV_success
when
$r\_st = working$
$r + 1 = n$
then
$r\_st := success$
$r := r + 1$
$h := h \cup \{n \mapsto f(n)\}$
end

RCV_failure
when
$r\_st = working$
$s\_st = failure$
then
$r\_st := failure$
end

- These new events are cheating (accessing f and s_st)

- These new events maintain **inv1_3** and **inv1_4** and they refine skip

inv1_3: $r\_st = success \Leftrightarrow r = n$
inv1_4: $s\_st = success \Rightarrow r\_st = success$

First Refinement: the Events (5)

SND_success
when
$s\_st = working$
$r\_st = success$
then
$s\_st := success$
end

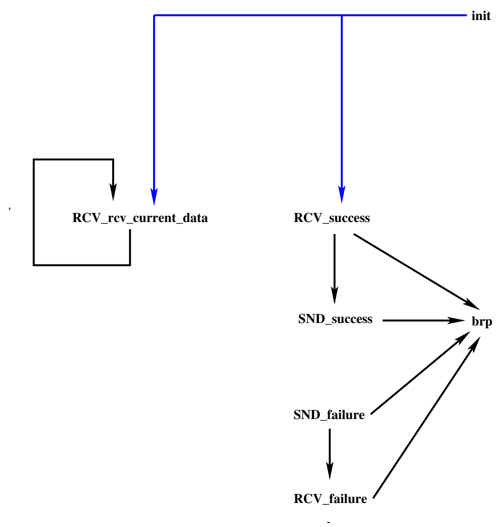
SND_failure
when
$s\_st = working$
then
$s\_st := failure$
end

- Event SND_success is cheating (accessing r_st)
- Event SND_success maintains invariant **inv1_4**

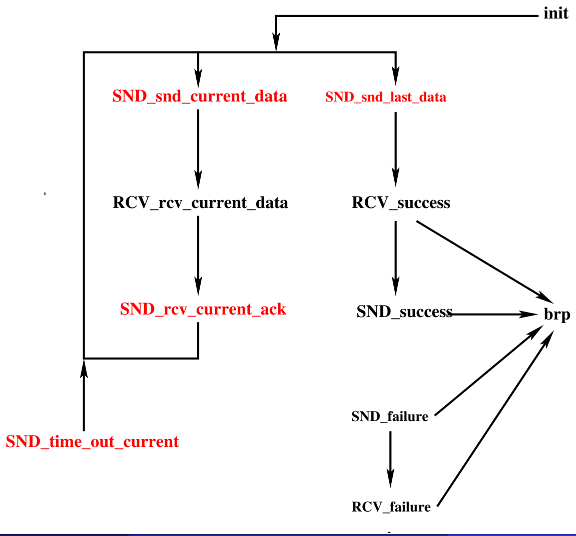
inv1_4: $s\_st = success \Rightarrow r\_st = success$

Outline

Second Refinement: Introducing More Events



Second Refinement: Introducing More Events



Second Refinement: More Variables

- Variable s is the Sender pointer sent to the Receiver
- Variable d is the data sent to the Receiver
- Variable w is the Sender activation bit
- When w is TRUE it means the Sender has just received the acknowledgement
- When w is FALSE it means the Sender has sent the information to the Receiver

variables: ... w s d

inv2_1: $s \in 0\,..\,n-1$

inv2_2: $r \in s\,..\,s+1$

inv2_3: $w = \text{FALSE} \Rightarrow d = f(s+1)$

The Main Communication

SND_snd_current_data ⟶ (d, s) ↘ RCV_rcv_current_data

↑

SND_rcv_current_ack ⟵ (r) ↙

Second Refinement: the Events (1)

init
  $r := 0$
  $h := \varnothing$
  $r\_st := working$
  $s\_st := working$
  $s := 0$
  $d :\in D$
  $w := \text{TRUE}$

brp
  when
    $r\_st \neq working$
    $s\_st \neq working$
  then
    skip
  end

Second Refinement: the Events (2)

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

```
SND_snd_current_data
  when
    s_st = working
    w = TRUE
    s + 1 < n
  then
    d := f(s + 1)
    w := FALSE
  end
```

```
SND_snd_last_data
  when
    s_st = working
    w = TRUE
    s + 1 = n
  then
    d := f(s + 1)
    w := FALSE
  end
```

- These events clearly refine skip and maintain invariant **inv2_3**

inv2_3: $w = \mathrm{FALSE} \Rightarrow d = f(s+1)$

Second Refinement: the Events (3)

- The Receiver receives data d and pointer s. It sends pointer r.

```
RCV_rcv_current_data
  when
    r_st = working
    w = FALSE
    r = s
    r + 1 < n
  then
    r := r + 1
    h := h ∪ {r + 1 ↦ d}
  end
```

```
RCV_success
  when
    r_st = working
    w = FALSE
    r = s
    r + 1 = n
  then
    r_st := success
    r := r + 1
    h := h ∪ {r + 1 ↦ d}
  end
```

- The Receiver still cheats: it accesses constant n

Refinement of RCV_rcv_current_data

```
(abstract-)RCV_rcv_current_data
  when
    r_st = working
    r + 1 < n
  then
    r := r + 1
    h := h ∪ {r + 1 ↦ f(r + 1)}
  end
```

```
(concrete-)RCV_rcv_current_data
  when
    r_st = working
    w = FALSE
    r = s
    r + 1 < n
  then
    r := r + 1
    h := h ∪ {r + 1 ↦ d}
  end
```

- Observe guard strengthening

- This invariant helps proving event refinement

inv2_3: $w = \mathrm{FALSE} \Rightarrow d = f(s+1)$

Refinement of RCV_success

```
(abstract-)RCV_success
  when
    r_st = working
    r + 1 = n
  then
    r_st := success
    r := r + 1
    h := h ∪ {n ↦ f(n)}
  end
```

```
(concrete-)RCV_success
  when
    r_st = working
    w = FALSE
    r = s
    r + 1 = n
  then
    r_st := success
    r := r + 1
    h := h ∪ {r + 1 ↦ d}
  end
```

- Observe guard strengthening

- This invariant helps proving event refinement

inv2_3: $w = \mathrm{FALSE} \Rightarrow d = f(s+1)$

Second Refinement: the Events (4)

- The first event is new. It clearly refines skip
- The activation bit is set to TRUE (activating SND_snd_current_data)
- The Sender receives acknowledgment (pointer r)

```
SND_rcv_current_ack
  when
    s_st = working
    w = FALSE
    s + 1 < n
    r = s + 1
  then
    w := TRUE
    s := s + 1
  end
```

```
SND_success
  when
    s_st = working
    w = FALSE
    s + 1 = n
    r = s + 1
  then
    s_st := success
  end
```


Refinement of SND_success

```
(abstract-)SND_success
  when
    s_st = working
    r_st = success
  then
    s_st := success
  end
```

```
(concrete-)SND_success
  when
    s_st = working
    w = FALSE
    s + 1 = n
    r = s + 1
  then
    s_st := success
  end
```

- The presence of **inv1_3** ensures that the guard is strengthen

inv1_3: $r\_st = success \Leftrightarrow r = n$

Second Refinement: the Events (5)

- This new events will receive a full explanation in the next refinement

```
SND_time_out_current
  when
    s_st = working
    w = FALSE
  then
    w := TRUE
  end
```


Outline

Third Refinement: Introducing more Activation Bits

- At most one activation bit is TRUE at a time

variables: ...
db
ab
v

inv3_1: $w = \mathrm{TRUE} \Rightarrow db = \mathrm{FALSE}$

inv3_2: $w = \mathrm{TRUE} \Rightarrow ab = \mathrm{FALSE}$

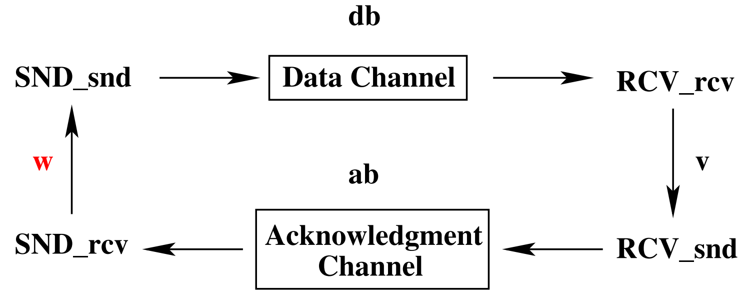
inv3_3: $w = \mathrm{TRUE} \Rightarrow v = \mathrm{FALSE}$

inv3_4: $db = \mathrm{TRUE} \Rightarrow ab = \mathrm{FALSE}$

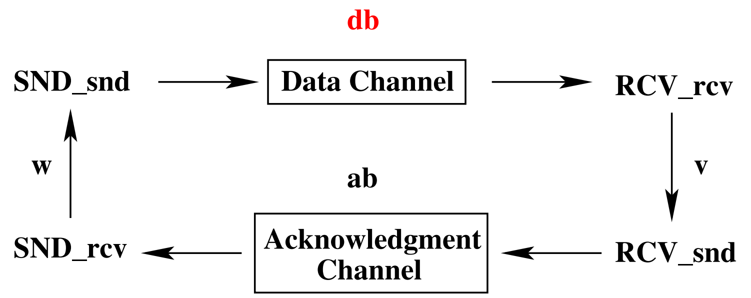
inv3_5: $db = \mathrm{TRUE} \Rightarrow v = \mathrm{FALSE}$

inv3_6: $ab = \mathrm{TRUE} \Rightarrow v = \mathrm{FALSE}$

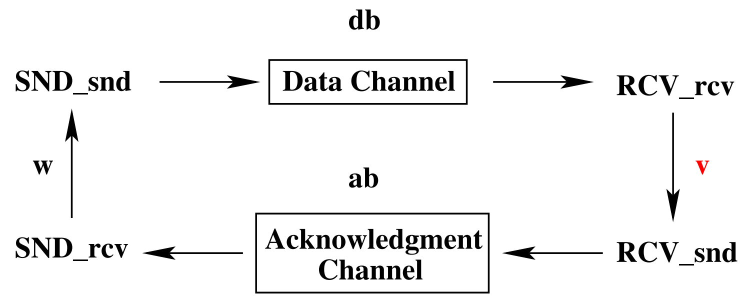
Activation bits at work



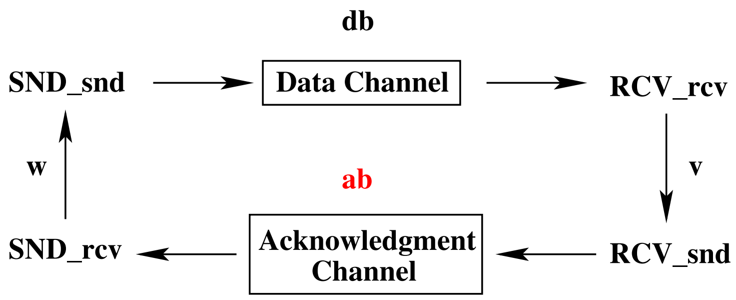
Activation bits at work



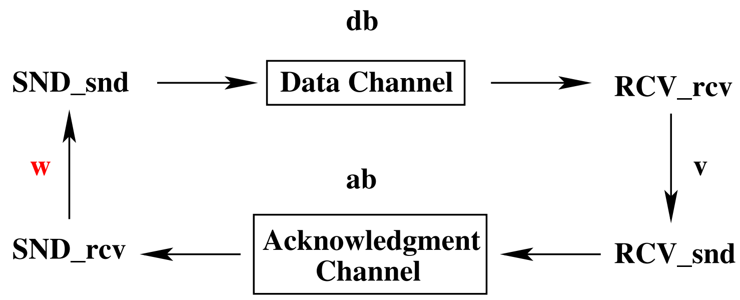
Activation bits at work



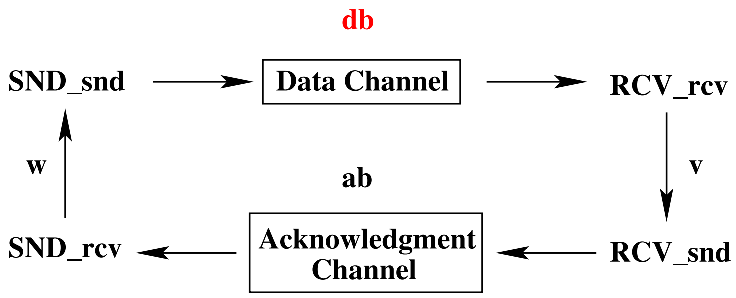
Activation bits at work



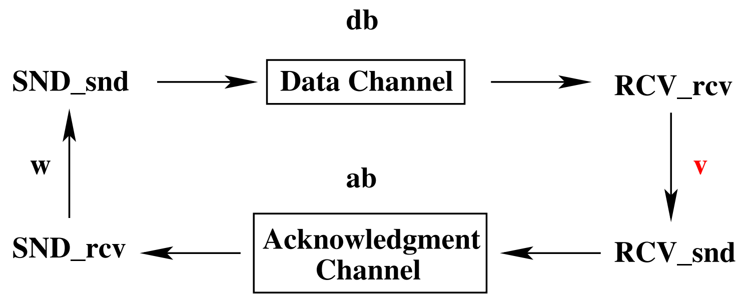
Activation bits at work



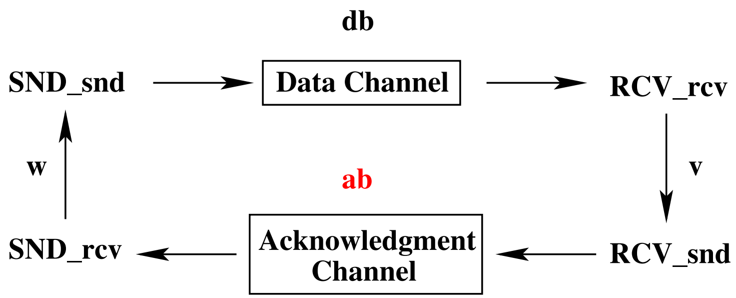
Activation bits at work



Activation bits at work



Activation bits at work



Third Refinement: Introducing the Last Item Indicator

- These invariants define the last data indicator

variables: . . .
l

inv3_7: $db = \mathrm{TRUE} \wedge r = s \wedge l = \mathrm{FALSE} \Rightarrow r + 1 < n$

inv3_8: $db = \mathrm{TRUE} \wedge r = s \wedge l = \mathrm{TRUE} \Rightarrow r + 1 = n$

- This bit is sent by the Sender to the Receiver

- When equal to TRUE, this bit indicates that the sent item is the last one

Third Refinement: Introducing the Retry Counter c

- Constant MAX denotes the maximum number of retries

- The sender fails iff the retry counter c exceeds MAX (**inv3_10**)

constants: . . .
MAX

axm3_1: $MAX \in \mathbb{N}$

variables: . . .
c

inv3_9: $c \in 0 \,..\, MAX + 1$

inv3_10: $c = MAX + 1 \Leftrightarrow s\_st = failure$

Third Refinement: the Events (1)

init
- $r := 0$
- $h := \varnothing$
- $r\_st := working$
- $s\_st := working$
- $s := 0$
- $d :\in D$
- $w := \mathrm{TRUE}$
- $db := \mathrm{FALSE}$
- $ab := \mathrm{FALSE}$
- $v := \mathrm{FALSE}$
- $l := \mathrm{FALSE}$
- $c := 0$

brp
when
- $r \neq working$
- $s \neq working$

then
- skip

end

Third Refinement: the Events (2)

```
SND_snd_current_data
  when
    s_st = working
    w = TRUE
    s + 1 < n
  then
    d := f(s + 1)
    w := FALSE
    db := TRUE
    l := FALSE
  end
```

```
SND_snd_last_data
  when
    s_st = working
    w = TRUE
    s + 1 = n
  then
    d := f(s + 1)
    w := FALSE
    db := TRUE
    l := TRUE
  end
```


Third Refinement: New Events

- Daemons are breaking the channels

```
DMN_data_channel
  when
    db = TRUE
  then
    db = FALSE
  end
```

```
DMN_ack_channel
  when
    ab = TRUE
  then
    ab = FALSE
  end
```

- A failure is characterized by all activation bits being FALSE

Third Refinement: the Events (3)

```
SND_time_out_current
  when
    s_st = working
    w = FALSE
    ab = FALSE
    db = FALSE
    v = FALSE
    c < MAX
  then
    w := TRUE
    c := c + 1
  end
```

```
SND_failure
  when
    s_st = working
    w = FALSE
    ab = FALSE
    db = FALSE
    v = FALSE
    c = MAX
  then
    s_st := failure
    c := c + 1
  end
```

- Sender aborts after $MAX + 1$ tries

Third Refinement: the Events (4)

```
RCV_rcv_current_data
  when
    r_st = working
    db = TRUE
    r = s
    l = FALSE
  then
    r := r + 1
    h := h ∪ {r + 1 ↦ d}
    db := FALSE
    v := TRUE
  end
```

```
RCV_success
  when
    r_st = working
    db = TRUE
    r = s
    l = TRUE
  then
    r_st := success
    r := r + 1
    h := h ∪ {r + 1 ↦ d}
    db := FALSE
    v := TRUE
  end
```

Reminder: l is the last data indicator

Third Refinement: Guard Srengthening (1)

(abstract-)RCV_rcv_current_data
when
$r\_st = working$
$w = \text{FALSE}$
$r = s$
$r + 1 < n$
then
$r := r + 1$
$h := h \cup \{r + 1 \mapsto d\}$
end

(concrete-)RCV_rcv_current_data
when
$r\_st = working$
$db = \text{TRUE}$
$r = s$
$l = \text{FALSE}$
then
$r := r + 1$
$h := h \cup \{r + 1 \mapsto d\}$
$db := \text{FALSE}$
$v := \text{TRUE}$
end

inv3_1': $db = \text{TRUE} \Rightarrow w = \text{FALSE}$

inv3_7: $db = \text{TRUE} \wedge r = s \wedge l = \text{FALSE} \Rightarrow r + 1 < n$

Third Refinement: Guard Srengthening (2)

(abstract-)RCV_success
when
$r\_st = working$
$w = \text{FALSE}$
$r = s$
$r + 1 = n$
then
$r := r + 1$
$h := h \cup \{r + 1 \mapsto d\}$
end

(concrete-)RCV_success
when
$r\_st = working$
$db = \text{TRUE}$
$r = s$
$l = \text{TRUE}$
then
$r\_st := success$
$r := r + 1$
$h := h \cup \{r + 1 \mapsto d\}$
$db := \text{FALSE}$
$v := \text{TRUE}$
end

inv3_1': $db = \text{TRUE} \Rightarrow w = \text{FALSE}$

inv3_8: $db = \text{TRUE} \wedge r = s \wedge l = \text{TRUE} \Rightarrow r + 1 = n$

Third Refinement: the Events (5)

RCV_rcv_retry
when
$db = \text{TRUE}$
$r \neq s$
then
$db := \text{FALSE}$
$v := \text{TRUE}$
end

RCV_snd_ack
when
$v = \text{TRUE}$
then
$v := \text{FALSE}$
$ab := \text{TRUE}$
end

RCV_failure
when
$r\_st = working$
$c = MAX + 1$
then
$r\_st := failure$
end

Third Refinement: the Events (6)

SND_rcv_current_ack
when
$s\_st = working$
$ab = \text{TRUE}$
$s + 1 < n$
then
$w := \text{TRUE}$
$s := s + 1$
$c := 0$
$ab := \text{FALSE}$
end

SND_success
when
$s\_st = working$
$ab = \text{TRUE}$
$s + 1 = n$
then
$s\_st := success$
$c := 0$
$ab := \text{FALSE}$
end

Third Refinement: Guard Strengthening (1)

(abstract-)SND_rcv_current_ack
when
$s\_st = working$
$w = \text{FALSE}$
$s + 1 < n$
$r = s + 1$
then
$w := \text{TRUE}$
$s := s + 1$
end

(concrete-)SND_rcv_current_ack
when
$s\_st = working$
$ab = \text{TRUE}$
$s + 1 < n$
then
$w := \text{TRUE}$
$s := s + 1$
$c := 0$
$ab := \text{FALSE}$
end

inv3_2': $ab = \text{TRUE} \Rightarrow w = \text{FALSE}$

- In order to prove guard strengthening we need invariant **inv3_11**, and invariant **inv3_12** is needed to prove **inv3_11**

inv3_11: $ab = \text{TRUE} \Rightarrow r = s + 1$

inv3_12: $v = \text{TRUE} \Rightarrow r = s + 1$

Third Refinement: Guard Strengthening (2)

(abstract-)SND_success
when
$s\_st = working$
$w = \text{FALSE}$
$s + 1 = n$
$r = s + 1$
then
$s\_st := success$
end

(concrete-)SND_success
when
$s\_st = working$
$ab = \text{TRUE}$
$s + 1 = n$
then
$s\_st := success$
$c := 0$
$ab := \text{FALSE}$
end

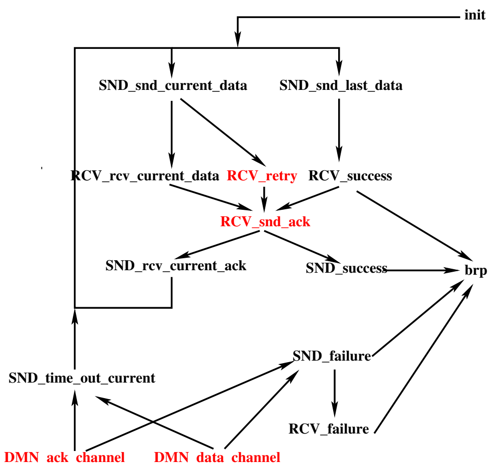
inv3_2': $ab = \text{TRUE} \Rightarrow w = \text{FALSE}$

- In order to prove guard strengthening we need invariant **inv3_11**, and invariant **inv3_12** is needed to prove **inv3_11**

inv3_11: $ab = \text{TRUE} \Rightarrow r = s + 1$

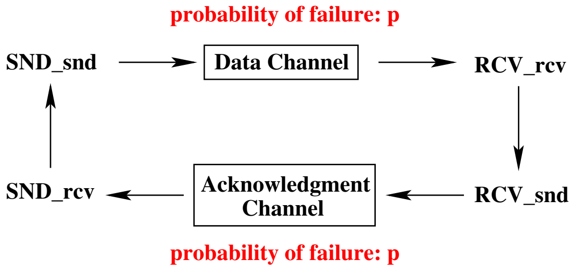
inv3_12: $v = \text{TRUE} \Rightarrow r = s + 1$

Final Synchronization of the Events



Outline

Computing Probabilities



- We would like to compute the probability of success
- It is a function of:
    - p: probability of failure for one channel
    - n: size of the file
    - $MAX+1$: number of re-tries

Computing Probabilities

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

$p = .1$
$MAX = 5$
$n = 100$ .995