Verification of Real Time Systems - CS5270 lecture 11

Hugh Anderson

National University of Singapore
School of Computing

March, 2007

Time machine...

Outline

1. Administration
   - Assignment 3
   - The road map...
2. TCTL Model checking
   - TCTL model checking algorithm
3. Case studies for Uppaal
   - Uppaal coffee machine example
   - Uppaal simple protocol example
   - Bounded retransmission protocol

Assignment 3

A reminder... Assignment number 3:

- On the web site
- Due 9th April! ...

The immediate road map

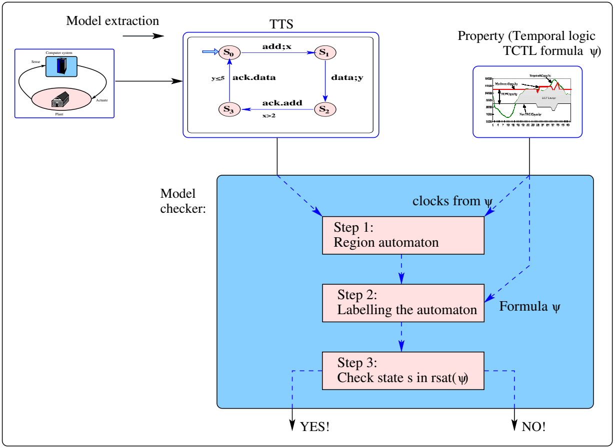
The topics:

- **CTL Model Checking**
  - CTL model checking relation, algorithm, optimizations
  - Example smv/spin - CTL/LTL model checkers
- **TCTL Model Checking**
  - TCTL model checking background

  - TCTL model checking relation
  - TCTL model checking algorithm
- **Uppaal examples**
  - Coffee machine
  - Simple protocol
  - More complex protocol - BRP

TCTL model checking

Construct Model, label states using rsat(), check inclusion



The TCTL satisfaction relation

TCTL model checker satisfaction relation:

set_of_Regions rsat(Property ψ) =

- **if** $\psi \in AP$ **then** $\{(r, f) \mid \psi \in \mathcal{L}(\bar{r})\}$
- **else case** ψ **of**
  - **true:** $\overline{R}$
  - **false:** $\emptyset$
  - ϕ**:** $\{(r, f) \mid v \cup f \models \phi\}$
  - $\neg\psi$**:** $\overline{R} - \mathbf{rsat}(\psi)$
  - $\psi_1 \wedge \psi_2$**:** $\mathbf{rsat}(\psi_1) \cap \mathbf{rsat}(\psi_2)$
  - $\psi_1 \vee \psi_2$**:** $\mathbf{rsat}(\psi_1) \cup \mathbf{rsat}(\psi_2)$
  - z in ψ_1**:** $\{(r, f) \mid (r, z \text{ in } f) \in \mathbf{rsat}(\psi_1)\}$
  - $\mathbf{A}(\psi_1 \mathbf{U} \psi_2)$**:** $\mathbf{lfp}(g(Z) = \mathbf{rsat}(\psi_2) \cup (\mathbf{rsat}(\psi_1) \cap \{(r, f) \mid \forall r' \in (r, f)^{\uparrow} \cap Z\}))$
  - $\mathbf{E}(\psi_1 \mathbf{U} \psi_2)$**:** $\mathbf{lfp}(h(Z) = \mathbf{rsat}(\psi_2) \cup (\mathbf{rsat}\ (\psi_1) \cap \{(r, f) \mid \exists r' \in (r, f)^{\uparrow} \cap Z\}))$

The Uppaal model checker

Uppaal: Reachability analysis only, on zones...

In Uppaal, the clock constraints for the formula must be $\emptyset$, and the syntax of the language accepted is restricted to only the following two temporal operators:

$$\mathrm{AG}(\psi)$$
$$\mathrm{EF}(\psi)$$

and ψ is of the form

$$\psi ::= a \mid x \,\mathrm{op}\, n \mid \neg\psi \mid \psi_1 \wedge \psi_2$$

where a is a location, and op is a simple comparison operator.

The Uppaal model checker

Reachability is simpler than full TCTL:

Even though the restrictions on Uppaal appear quite limiting, it is possible to express any sort of reachability query using this subset of TCTL.

The general strategy is to modify the model, adding clocks, and perhaps observers.

An algorithm to see if a particular starting state (s_0, r_0) can result in a particular final state s_f given a particular clock assignment ϕ is shown on the next slide.

This algorithm operates over the RTS, and is quite simple.

Reachability algorithm

The reachable() function:

```
set_of_regions checked = ∅;
set_of_regions toCheck = {(s_0, r_o)};
set_of_states final = –s_f";
boolean reachable( clock_assignment φ ) =
  while ( toCheck≠ ∅ ) do
    foreach ( (s,r) ∈toCheck ) do
      if ( s = s_f ∧ r ∩ φ ≠ ∅ ) then return TRUE;
      if ( ∀(s,r') ∈ checked,r ⊄ r' ) then
        checked = checked+(s,r);
        foreach ( (s',r'),(s,r) → (s',r') ) do
          toCheck = toCheck+(s',r');
  return FALSE;
```

Uppaal coffee machine: Specification

Model system with coffee machine, person and observer:

1. The person repeatedly tries to insert a coin, tries to extract coffee after which (s)he will make a publication. Between each action the person requires a suitable time-delay before being ready to participate in the next one.
2. After receiving a coin the machine should take some time for brewing the coffee. The machine should time-out if the brewed coffee has not been taken before a certain upper time-limit.
3. The observer should complain if at any time more than 8 time-units elapses between two consecutive publications.

Uppaal coffee machine: Underspecification

The system is only partially described:

In the specification there is a worrying phrase: “The Machine should time-out if the brewed coffee has not been taken before a certain upper time-limit”.

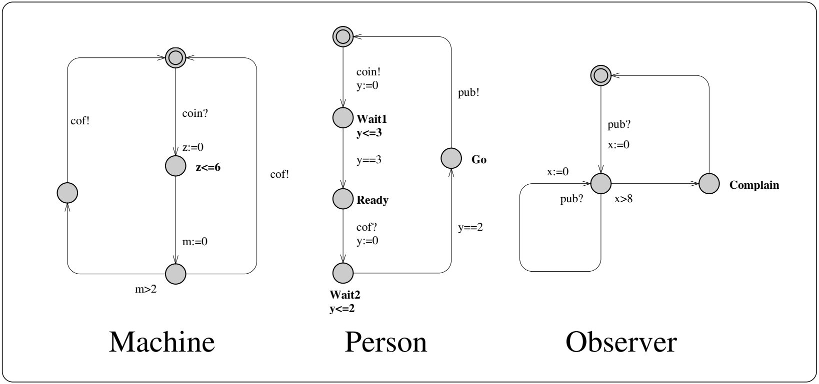
This phrase is worrying because it is an under-specification of the system.

For example: “What does the machine do if it times out?”. If it times out and then dumps the coffee, the system will deadlock, as the Person automata must pay and then drink.

So - rather than modifying the specified Person automaton, the machine specified here times out and then synchronizes on the dispensing of coffee.

Uppaal coffee machine: Model

The three timed transition systems:



Uppaal coffee machine: Properties

Interesting Temporal properties/queries:

In UPPAAL, path operators ◊ and □ are written as `<>` and `[]`
To test the model, the temporal query

```
E<> Observer.Complain
```

is used, which corresponds to the CTL formula

```
EF Observer.Complain
```

specifying that:

- for at least one computation path, at some time state `Observer.Complain` is reached.

In addition the system is tested with `A[] not deadlock`.

Uppaal coffee machine: Properties

The results of the testing are as follows:

1. System is deadlock free
2. `Observer.Complain` is reached if the coffee timeout is `7` or more
3. `Observer.Complain` is never reached if the coffee timeout is `6` or less

The last two tests were done by trial and error - setting the value in the coffee machine model to different values, and rerunning the model checker.

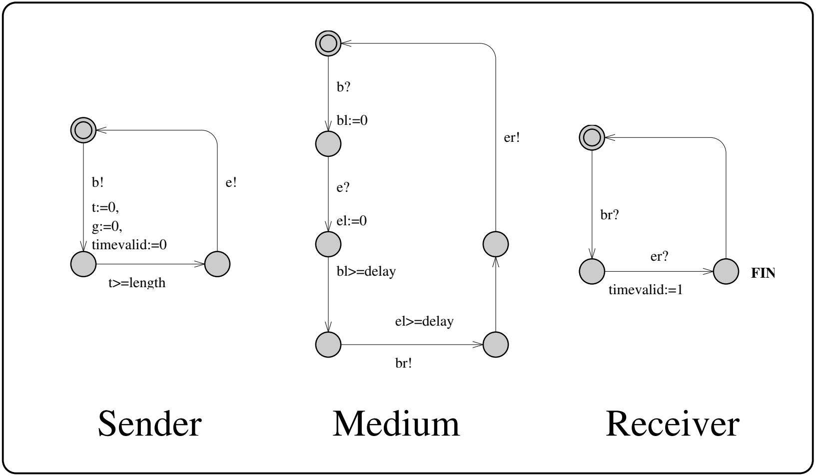
Uppaal simple protocol: Specification

Model system with medium, sender and receiver:

1. The sender just synchronizes on the beginning and end of the message, ensuring a time of length between the two synchronizations. The `timevalid` and `g` variables are global variables used to time the total transit time of the message.
2. The receiver just synchronizes on the beginning and end of the message after it arrives from the medium, setting `timevalid` as the `FIN` state is entered.
3. The medium uses two local clocks, `bl` and `el`, to delay a message enroute to the receiver.

Uppaal simple protocol: Model

The first step is to model the system, assuming length<delay:



Uppaal simple protocol: Properties

Interesting Temporal properties/queries:

- A quick test with `A[] not deadlock` shows that it is deadlock free.
- To find out the total time between begin send and end receive, a global clock variable *g* is reset by the sender at the beginning of a message, and its value in state `Receiver.FIN` tells us the total time between the beginning of sending the message and the end of receiving the message.
- To test this a global variable `timevalid` was added to the system, and if the receiver is in the `Receiver.FIN` state, and the time is valid, then we can run various tests

Uppaal simple protocol: Properties

Interesting Temporal properties/queries:

The query

```
E<> (Receiver.FIN and timevalid==1 and g<maxtime)
```

(where `maxtime` is `length+delay`) is always unsatisfied, which tells us there is no time sequence shorter than `length+delay`.

The query

```
A[] (Receiver.FIN and timevalid==1 imply g>=maxtime)
```

is satisfied, which tells us that the time will always be greater than or equal to `length+delay`.

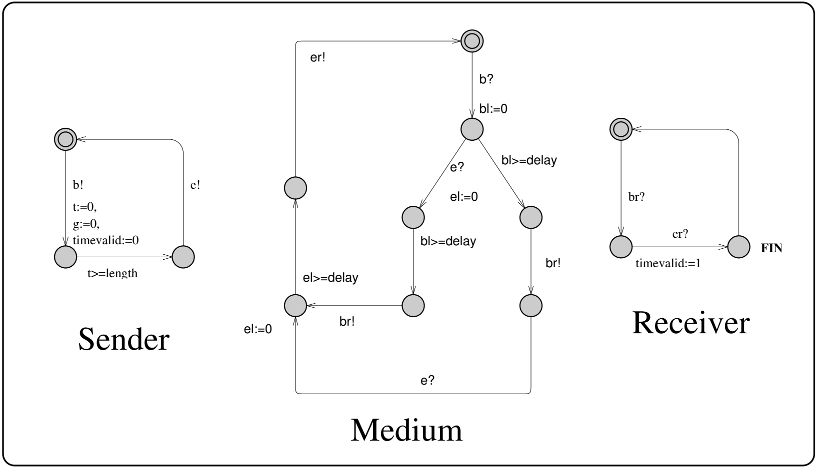
Uppaal simple protocol: Extending the model

Handling more messages:

The preceding model could only handle systems in which the length of the message was less than the medium delay time. We can extend the medium to also handle messages with `length>=delay`.
The only thing that needs changing is the definition of medium, given in next slide...

Uppaal simple protocol: Extending the model

A new medium:



Uppaal simple protocol: results

The new extended model:

The two queries before both still produce the expected results, no matter what the relationship between length and delay:

```
E<> (Receiver.FIN and timevalid==1 and g<maxtime)
A[] (Receiver.FIN and timevalid==1 imply g>=maxtime)
```

From the Uppaal website

The examples directory:

There are lots of examples, all more complicated than we have just looked at.

- Bang & Olufsen Audio/Video Protocol.
- Bang & Olufsen Power Down Protocol.
- Commercial Field Bus Protocol.
- Gear Box Controller.
- Multimedia Stream.
- ...and BRP...

BRP: the bounded retransmission protocol

Overview:

- Developed by Phillips Electronics Corporation.
- A real-time bounded variant of the alternating-bit protocol.
- Used to transfer in burst-mode a list of data (a file) via an infra-red communication medium between AV equipment and a remote control unit, with a lossy medium!
- The file is transmitted in chunks. If an acknowledgment for a sent-chunk is not received in time the chunk is retransmitted.
- If the number of retransmissions for the same chunk exceed a bound then the transmission is aborted..

BRP: the bounded retransmission protocol

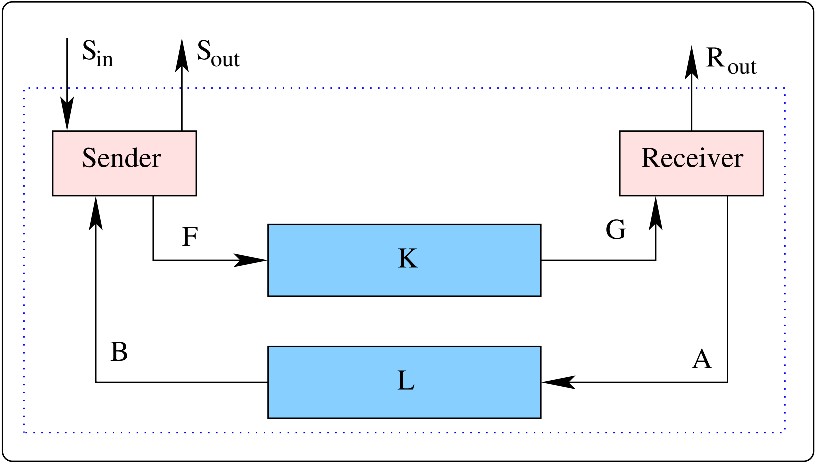
Timing aspects:

1. The sender has a timer to decide when to retransmit a chunk.
2. The receiver has a timer to detect when a transmission has been aborted by the sender.
3. The correctness of the system (with respect to the specification) is not just in terms of DATA.

Timing considerations are also important.

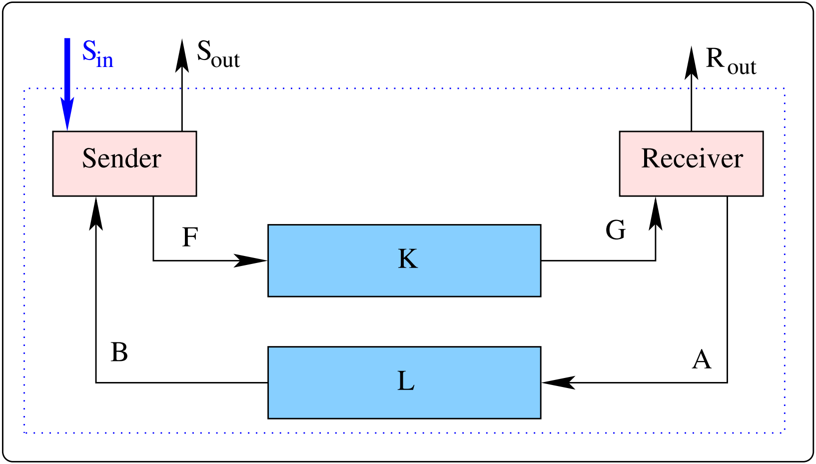
BRP: the bounded retransmission protocol

Senders, receivers and media:



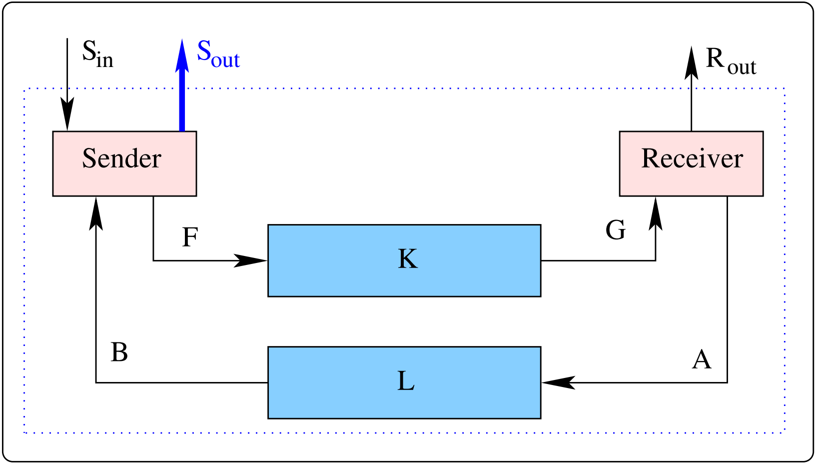
BRP: the bounded retransmission protocol

$(d_1, d_2, \ldots d_n)$ a file with n chunks of data



BRP: the bounded retransmission protocol

$(I_{\mathrm{OK}}, I_{\mathrm{NOK}}, I_{\mathrm{DK}})$ indications of what happened



BRP: the bounded retransmission protocol

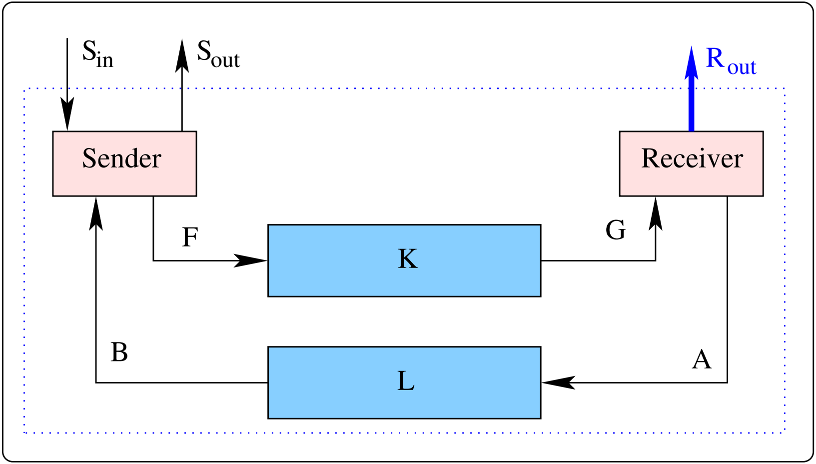
The values for S_{out}:

- I_{OK} All the acknowledgments were received. All the chunks were transmitted successfully and were received by the receiver.
- I_{NOK} Some acks failed to arrive in time; the MAX count of retransmissions for that chunk was exhausted without receiving an ack.
- I_{DK} The acks were received for all the chunks except the last one. Dont know whether the transmission was successful or not.

This is due to asynchronous communication via a lossy channel. It is Byzantine - agreement is impossible!

BRP: the bounded retransmission protocol

$(e_1, i_1)(e_2, i_2) \ldots (e_k, i_k)$ output



BRP: the bounded retransmission protocol

The values for R_{out}:

$(e_1, i_1)(e_2, i_2), \ldots, (e_k, i_k)$

- with $0 \leq k \leq n$, and
- $i_j \in \{I_{\text{FST}}, I_{\text{INC}}, I_{\text{OK}}, I_{\text{NOK}}\}$ with $0 < j \leq k$

I_{FST} The first chunk of the file but not the last one.

I_{OK} The last chunk of the file.

I_{INC} For all other chunks.

I_{NOK} Something has gone wrong. In this case $j = k$ and $e_k = *$ (no datum).

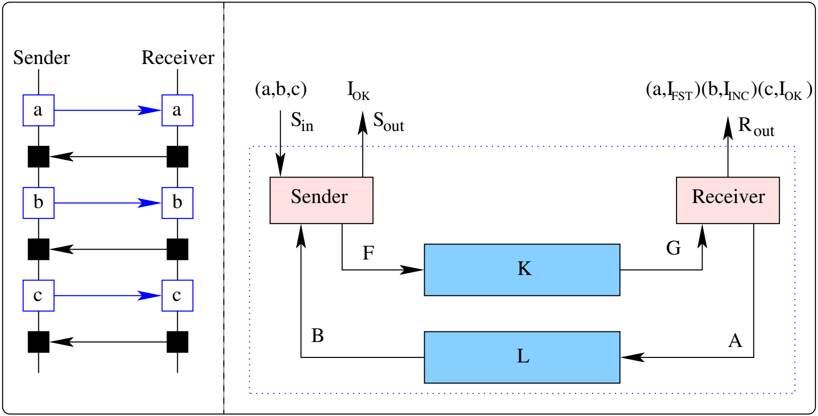
BRP: the bounded retransmission protocol

The specification. In (e_j, i_j):

- For every $0 < j \leq k$, if $i_j \neq I_{\mathrm{NOK}}$ then $e_j = d_j$. The datum delivered is the chunk that was sent.
- If $n > 1$ then $i_1 = I_{\mathrm{FST}}$
- I_{NOK} is put out only if something at all is received.
- If $1 < j < k$ then $i_j = I_{\mathrm{INC}}$
- $i_k = I_{\mathrm{OK}}$ OR $i_k = I_{\mathrm{NOK}}$ - The last output must signal positive or negative termination.
    - $i_k = I_{\mathrm{OK}}$ implies $k = n$. Successful transmission.
    - $i_k = I_{\mathrm{NOK}}$ implies $k > 1$. Unsuccessful only if something was received to start with.

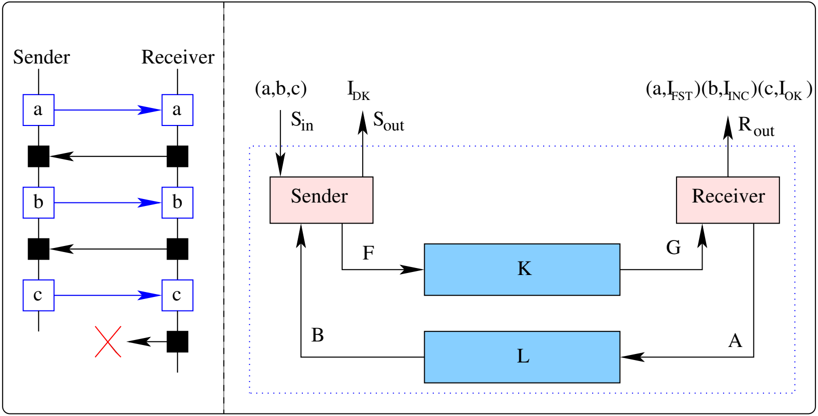
BRP: the bounded retransmission protocol

Normal operation:



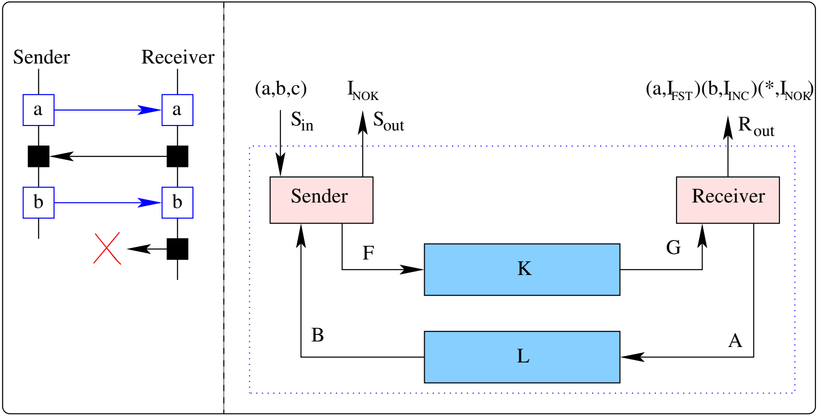
BRP: the bounded retransmission protocol

Lost last acknowledgement:



BRP: the bounded retransmission protocol

Lost acknowledgement during transmission:



BRP: the bounded retransmission protocol

Timing behaviour of the protocol:

- The sender reads the file (with n chunks $(d_1, d_2, \ldots, d_n)$) and sets a retry counter to 0.
- It then starts sending over the chunks one by one: Its sets a timer T_1 and sends the first frame into the channel K.
- A frame is of the form $\langle b_1, b_2, \mathrm{ab}, d_i \rangle$:
  - b_1 indicates that this chunk is the first one.
  - b_2 indicates that this chunk is the last one.
  - ab is the alternating bit. It is used to distinguish between a retry and a fresh chunk.
  - d_i is the chunk.

BRP: the bounded retransmission protocol

Timing behaviour of the protocol:

- After sending the frame $\langle b_1, b_2, \mathrm{ab}, d_1 \rangle$, the sender module waits for an acknowledgment or a time-out.
  - If an acknowledgement is received in time then T_1 is reset, and the next frame $\langle b_1', b_2', \neg\mathrm{ab}, d_2 \rangle$ is sent or (if $b_2 = 1$ in the previous round), it signals $R_{\mathrm{OUT}} = I_{\mathrm{OK}}$.
  - If it times out, the frame $\langle b_1, b_2, \mathrm{ab}, d_1 \rangle$ is resent after resetting the timer and incrementing the retry counter. If MAX is exceeded in the process of incrementing the counter, the transmission is broken off;
    - it signals $R_{\mathrm{OUT}} = I_{\mathrm{NOK}}$ or $R_{\mathrm{OUT}} = I_{\mathrm{DK}}$ depending on n and how many acknowledgement messages were received.

BRP: the bounded retransmission protocol

Timed analysis problem:

Using tools like spin or smv, we can check ordering and correct operation of *data* aspects of the protocol, but the protocol has embedded timing constraints.

The paper demonstrates how these can be modelled in Uppaal, and showed that there were some (timing) assumptions about the protocol that were not specified.

The analysis showed how to tighten the specification so that the protocol worked correctly.