Universiti Malaysia PAHANG

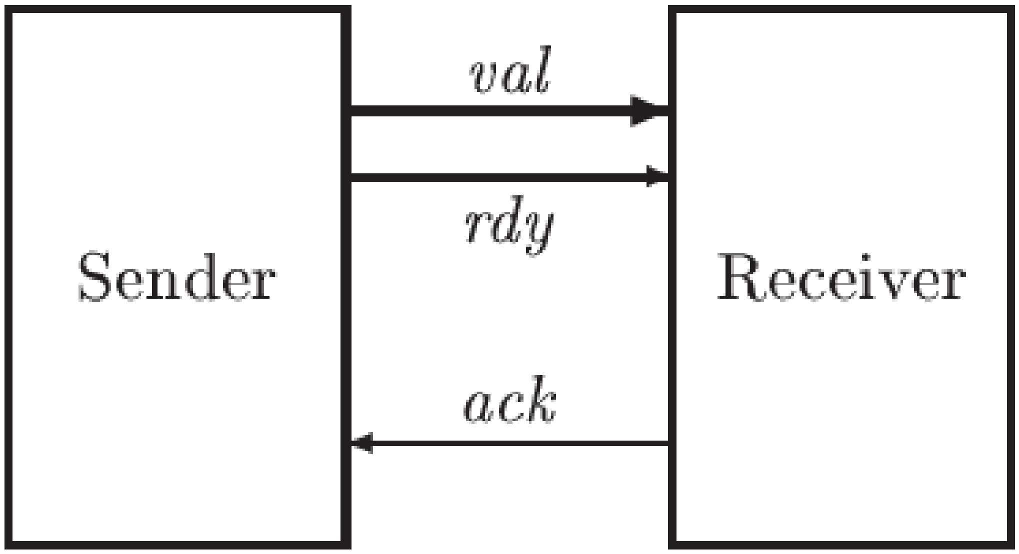
Faculty of Computer Systems & Software Engineering

Formal methods.

Specification of an Asynchronous Interface

Vitaliy Mezhuyev

An Asynchronous data transfer system



Components:

- Sender and Receiver
- Data line **val** (value)
- **rdy** (ready) and **ack** (acknowledgment) synchronization lines.

Data transfer protocol

$$\begin{bmatrix} val & = & 26 \\ rdy & = & 0 \\ ack & = & 0 \end{bmatrix} \xrightarrow{Send\ 37} \begin{bmatrix} val & = & 37 \\ rdy & = & 1 \\ ack & = & 0 \end{bmatrix} \xrightarrow{Ack} \begin{bmatrix} val & = & 37 \\ rdy & = & 1 \\ ack & = & 1 \end{bmatrix} \xrightarrow{Send\ 4}$$

$$\begin{bmatrix} val & = & 4 \\ rdy & = & 0 \\ ack & = & 1 \end{bmatrix} \xrightarrow{Ack} \begin{bmatrix} val & = & 4 \\ rdy & = & 0 \\ ack & = & 0 \end{bmatrix} \xrightarrow{Send\ 19} \begin{bmatrix} val & = & 19 \\ rdy & = & 1 \\ ack & = & 0 \end{bmatrix} \xrightarrow{Ack} \cdots$$

The sender must wait for an acknowledgment (Ack) for one data item before it can send the next.

Specification - declarations

Let assume that Data is a constant set of data can be sent

$$\text{CONSTANT } Data$$

To create a model of lines we will define variables

$$\text{VARIABLES } val, rdy, ack$$

Variables *rdy* and *ack* can have value only 0 or 1.
In other words {0, 1} is the type of *rdy* and *ack*

To specify this property we will define the logical formula and will call it ***type invariant***

$$TypeInvariant \triangleq (val \in Data) \land (rdy \in \{0, 1\}) \land (ack \in \{0, 1\})$$

Specification - type invariant

Generally speaking, an invariant **Inv** of a specification **Spec** is a state predicate such that

$$Spec \Rightarrow \Box Inv$$

is a theorem.

We can formulate our *TypeInvariant* in more short form*:

$$\begin{aligned} TypeInvariant \;\triangleq\; & \land\ val \in Data \\ & \land\ rdy \in \{0, 1\} \\ & \land\ ack \in \{0, 1\} \end{aligned}$$

* The indentation is significant in TLA notation

Specification – init predicate

$$\begin{aligned} Init \;\triangleq\; & \land\; val \in Data \\ & \land\; rdy \in \{0, 1\} \\ & \land\; ack = rdy \end{aligned}$$

Initially, **val** can equal any element of Data.
rdy and **ack** can start either both 0 or both 1.

Specification – *Send action*

$$\begin{aligned} Send \;\triangleq\; & \land\ rdy = ack \\ & \land\ val' \in Data \\ & \land\ rdy' = 1 - rdy \\ & \land\ \text{UNCHANGED}\ ack \end{aligned}$$

- Send is enabled if ***rdy*** equals ***ack***.
- The new value **val**' of val can be any element of Data
- **UNCHANGED ack** means ***ack***' = ***ack***

Specification – *Rcv* (Receive) *action*

$$
\begin{aligned}
Rcv \quad \triangleq \quad & \land\ rdy \neq ack \\
& \land\ ack' = 1 - ack \\
& \land\ \text{UNCHANGED}\ \langle val, rdy \rangle
\end{aligned}
$$

- ***rdy*** is different from ***ack***
- ***val*** and ***rdy*** remains unchanged
- TLA tuples in ASCII are defined with double triangle brackets, e.g. <<val, rdy>>

Specification – the next predicate

A step of the protocol either sends a value or receives a value.

$$Next \triangleq Send \lor Rcv$$

The specification allows stuttering steps, i.e. the steps that leave <<val, rdy, ack>> unchanged

$$Spec \triangleq Init \land \Box[Next]_{\langle val, rdy, ack \rangle}$$

Print values in TLC

- TLC allows to print values during model checking
- Operator **Print** is defined in the standard module TLC
- To use **Print,** include **TLC** by **EXTENDS** keyword.
- TLA definition of Print

Print(exp1, exp2) == exp2

the return value of Print(exp1, exp2) is exp2

Use Print in formulas

- To use Print in formulas we can specify

Print(exp, TRUE)

- To print more, than one expression we can use tuple

Print(<<id, exp>>, TRUE)

Example

```
Send ==    /\ ack' = 0
           /\ val' = IF val # 12 THEN val+1 ELSE 1
           /\ Print(<<"Send ", val>>, TRUE)
```

Using records

The fields of a record are not ordered,

$$[val : Data,\ rdy : \{0, 1\},\ ack : \{0, 1\}]$$

$$[ack : \{0, 1\},\ val : Data,\ rdy : \{0, 1\}]$$

This allow us to use one variable **chan** in exchange of val, rdy, ack

$\text{VARIABLE}\quad chan$

$$TypeInvariant \triangleq chan \in [val : Data,\ rdy : \{0, 1\},\ ack : \{0, 1\}]$$

Adding parameters to actions

The Send action now send unspecified data value
TLA allows to define Send with parameter, showing data to be sent ***Send(d)***

Send(d) *means that exists* ***d*** *in* ***Data****, such that the step satisfies* ***Send(d)***

This also allows us to modify the ***Next*** action

$$Next \triangleq (\exists\, d \in Data : Send(d)) \lor Rcv$$

Different possible of syntax of Send(d)

$$
\begin{aligned}
Send(d) \;\triangleq\; & \\
& \land\; chan.rdy = chan.ack \\
& \land\; chan' = [val \mapsto d,\; rdy \mapsto 1 - chan.rdy,\; ack \mapsto chan.ack]
\end{aligned}
$$

TLA also allows other syntax

$$
\begin{aligned}
Send(d) \;\triangleq\; & \land\; chan.rdy = chan.ack \\
& \land\; chan' = [chan \;\text{EXCEPT}\; !.val = d,\; !.rdy = 1 - @]
\end{aligned}
$$

Here **!.val, !.rdy** stands for **chan.val, chan.rdy** correspondingly
@ allow not repeat **chan.rdy**

Specification of Async Interface

MODULE *Channel*

EXTENDS *Naturals*

CONSTANT *Data*

VARIABLE *chan*

$$TypeInvariant \triangleq chan \in [val : Data,\ rdy : \{0,1\},\ ack : \{0,1\}]$$

$$Init \triangleq \land TypeInvariant$$
$$\qquad \land chan.ack = chan.rdy$$

$$Send(d) \triangleq \land chan.rdy = chan.ack$$
$$\qquad \land chan' = [chan \text{ EXCEPT } !.val = d,\ !.rdy = 1 - @]$$

$$Rcv \triangleq \land chan.rdy \neq chan.ack$$
$$\qquad \land chan' = [chan \text{ EXCEPT } !.ack = 1 - @]$$

$$Next \triangleq (\exists d \in Data : Send(d)) \lor Rcv$$

$$Spec \triangleq Init \land \Box[Next]_{chan}$$

THEOREM $Spec \Rightarrow \Box TypeInvariant$

MODULE *AsynchInterface*

EXTENDS *Naturals*

CONSTANT *Data*

VARIABLES *val*, *rdy*, *ack*

$$TypeInvariant \triangleq \land\ val \in Data \\ \land\ rdy \in \{0, 1\} \\ \land\ ack \in \{0, 1\}$$

$$Init \triangleq \land\ val \in Data \\ \land\ rdy \in \{0, 1\} \\ \land\ ack = rdy$$

$$Send \triangleq \land\ rdy = ack \\ \land\ val' \in Data \\ \land\ rdy' = 1 - rdy \\ \land\ \text{UNCHANGED}\ ack$$

$$Rcv \triangleq \land\ rdy \neq ack \\ \land\ ack' = 1 - ack \\ \land\ \text{UNCHANGED}\ \langle val, rdy \rangle$$

$$Next \triangleq Send \lor Rcv$$

$$Spec \triangleq Init \land \Box[Next]_{\langle val, rdy, ack \rangle}$$

THEOREM $Spec \Rightarrow \Box TypeInvariant$

Using comments in TLA

- A comment may appear anywhere enclosed between (* and *).
- An end-of-line comment is preceded by *.
- Comments may be nested, so you can comment out a section of a specification by enclosing it between (* and *)

Using comments in TLA

```
---------------------- MODULE HourClock ----------------------
  (*********************************************************)
  (* This module specifies a digital clock that displays  *)
  (* the current hour.  It ignores real time, not         *)
  (* specifying when the display can change.              *)
  (*********************************************************)
EXTENDS Naturals
VARIABLE hr     \* Variable hr represents the display.
HCini  == hr \in (1 .. 12)   \* Initially, hr can have any
                             \* value from 1 through 12.
HCnxt  (* This is a weird place for a comment. *)  ==
  (*************************************************)
  (* The value of hr cycles from 1 through 12.     *)
  (*************************************************)
  hr' = IF hr # 12 THEN hr + 1 ELSE 1
HC   ==  HCini /\ [][HCnxt]_hr
  (* The complete spec.  It permits the clock to stop. *)
--------------------------------------------------------------
THEOREM  HC => []HCini  \* Type-correctness of the spec.
==============================================================
```

Calculation of amount of distinct states

$$
\begin{aligned}
TypeInvariant \;\triangleq\; & \land\; val \in Data \\
& \land\; rdy \in \{0, 1\} \\
& \land\; ack \in \{0, 1\}
\end{aligned}
$$

What is the model?

Specify the values of declared constants.

```
Data <-  [ model value ] {d1, d2, d3}
```

Amount $= |Data| * |\{0,1\}| * |\{0,1\}|$

$= 3*2*2 = 12$

Analyses of specification

A step of the protocol either sends a value or receives a value.

$$Next \triangleq Send \lor Rcv$$

$$\begin{aligned} Send \triangleq\ & \land\ rdy = ack \\ & \land\ val' \in Data \\ & \land\ rdy' = 1 - rdy \\ & \land\ \text{UNCHANGED}\ ack \end{aligned}$$

$$\begin{aligned} Rcv \triangleq\ & \land\ rdy \neq ack \\ & \land\ ack' = 1 - ack \\ & \land\ \text{UNCHANGED}\ \langle val, rdy \rangle \end{aligned}$$

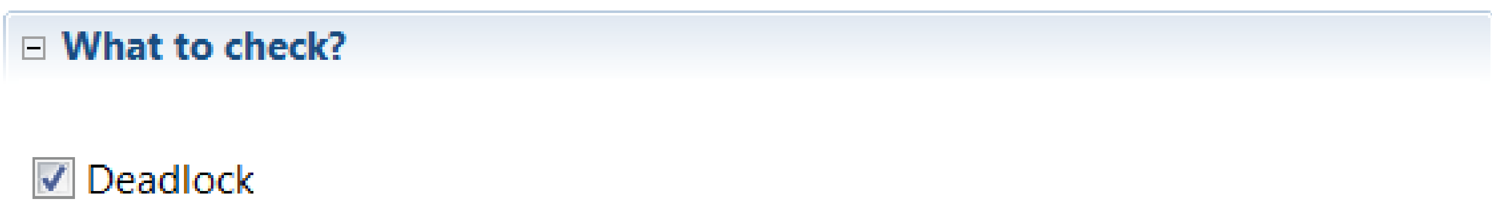
Analyses of deadlock

- The absence of deadlock is a particular property we often want to check. It is expressed by the invariance property

$$\Box(\textsc{enabled}\ Next)$$

- A counterexample to this property is a behavior exhibiting deadlock, i.e., reaching a state in which Next is not enabled, so no further (nonstuttering) step is possible.

- TLC normally checks for deadlock by setting (in Model overview):



- Note, it can be unchecked, since, for some systems, deadlock may just indicate successful termination.

Questions

1. What is Asynchronous data transfer system?
2. What are components of Asynchronous data transfer system?
3. What are actions of Asynchronous data transfer system?
4. How to calculate amount of distinct states?
5. What is deadlock?
6. Is deadlock always unwanted property?
7. How to print values in TLC?
8. How to Use Print in TLA formulas?

Thank you for your attention!
Please ask questions