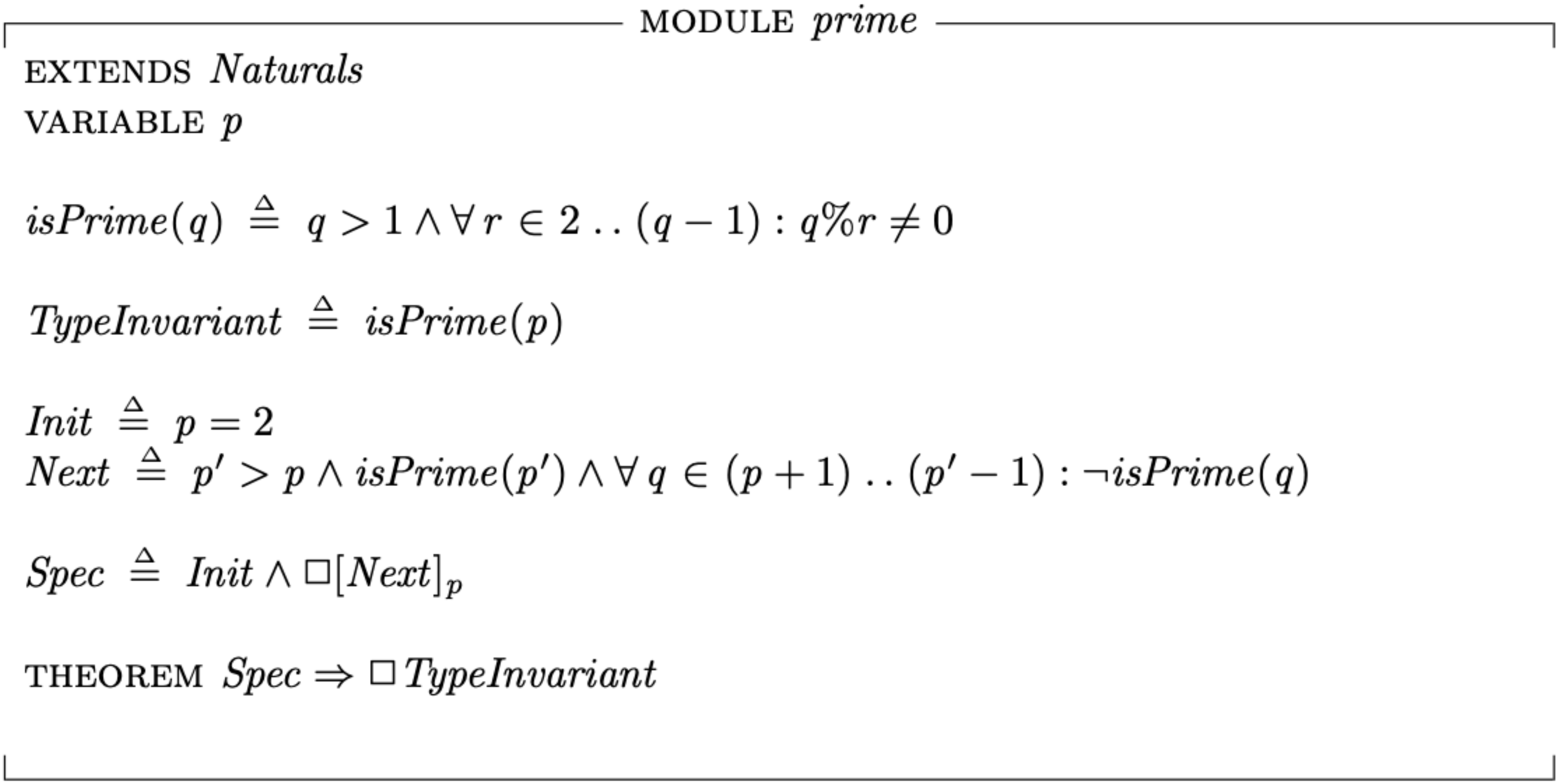
Lecture 2

Recall

- A *state* is an assignment of values to all variables
- A *step* is a pair of states
- A *stuttering step* wrt some variable leaves the variable unchanged
- An *action* is a predicate over a pair of states
  - If *x* is a variable in the old state, then *x'* is the same variable in the new state
- A *behavior* is an infinite sequence of states (with an initial state)
- A *specification* characterizes the initial state and actions

Spec that generates all prime numbers



MODULE *prime*

EXTENDS *Naturals*
VARIABLE p

$isPrime(q) \triangleq q > 1 \land \forall r \in 2 \,..\, (q-1) : q \% r \neq 0$

$TypeInvariant \triangleq isPrime(p)$

$Init \triangleq p = 2$
$Next \triangleq p' > p \land isPrime(p') \land \forall q \in (p+1) \,..\, (p'-1) : \neg isPrime(q)$

$Spec \triangleq Init \land \Box[Next]_p$

THEOREM $Spec \Rightarrow \Box\, TypeInvariant$

Spec that generates all prime numbers

```
------------------------------ MODULE prime ------------------------------
EXTENDS Naturals
VARIABLE p

isPrime(q) == q > 1 /\ \A r \in 2..(q-1): q%r /= 0

TypeInvariant == isPrime(p)

Init == p = 2
Next == p' > p /\ isPrime(p') /\ \A q \in (p+1)..(p'-1): ~isPrime(q)

Spec == Init /\ [] [Next]_p

THEOREM Spec => []TypeInvariant
```

Some more terms

- A *state function* is a first-order logic expression
- A *state predicate* is a Boolean state function
- A *temporal formula* is an assertion about behaviors
- *A theorem* of a specification is a temporal formula that holds over every behavior of the specification
- If S is a specification and I is a predicate and $S \Rightarrow \Box I$ is a theorem then we call I an *invariant* of S.

Temporal Formula

Based on Chapter 8 of Specifying Systems

- A *temporal formula* F assigns a Boolean value to a behavior σ
- $\sigma \vDash F$ means that F holds over σ
- If P is a state predicate, then $\sigma \vDash P$ means that P holds over the first state in σ
- If A is an action, then $\sigma \vDash A$ means that A holds over the first two states in σ
  - i.e., the first step in σ is an A step
  - note that a state predicate is simply an action without primed variables
- If A is an action, then $\sigma \vDash [A]_v$ means that the first step in σ is an A step or a stuttering step with respect to v

☐Always

- $\sigma \vDash \Box F$ means that F holds over every suffix of σ
- More formally
  - Let σ^{+n} be σ with the first n states removed
  - Then $\sigma \vDash \Box F \triangleq \forall n \in \mathbb{N}\colon \sigma^{+n} \vDash F$

Boolean combinations of temporal formulas

- $\sigma \vDash (F \wedge G) \triangleq (\sigma \vDash F) \wedge (\sigma \vDash G)$
- $\sigma \vDash (F \vee G) \triangleq (\sigma \vDash F) \vee (\sigma \vDash G)$
- $\sigma \vDash \neg F \triangleq \neg (\sigma \vDash F)$
- $\sigma \vDash (F \Rightarrow G) \triangleq (\sigma \vDash F) \Rightarrow (\sigma \vDash G)$
- $\sigma \vDash (\exists r : F) \triangleq \exists r : \sigma \vDash F$
- $\sigma \vDash (\forall r \in S : F) \triangleq \forall r \in S : \sigma \vDash F$ // if S is a constant set

Example

What is the meaning of $\sigma \vDash \Box((x = 1) \Rightarrow \Box(y > 0))$?

$$\begin{aligned}
&\sigma \vDash \Box((x = 1) \Rightarrow \Box(y > 0)) \\
&\equiv \forall n \in \mathbb{N}\colon \sigma^{+n} \vDash ((x = 1) \Rightarrow \Box(y > 0)) \\
&\equiv \forall n \in \mathbb{N}\colon (\sigma^{+n} \vDash (x = 1)) \Rightarrow (\sigma^{+n} \vDash \Box(y > 0)) \\
&\equiv \forall n \in \mathbb{N}\colon (\sigma^{+n} \vDash (x = 1)) \Rightarrow (\forall m \in \mathbb{N}\colon (\sigma^{+n})^{+m} \vDash (y > 0))
\end{aligned}$$

If $x = 1$ in some state, then henceforth $y > 0$ in all subsequent states

Not: once x = 1, x will always be 1. That would be

$$\sigma \vDash \Box((x = 1) \Rightarrow \Box(x = 1))$$

Not every temporal formula is a TLA+ formula

- TLA+ formulas are temporal formulas that are *invariant under stuttering*
  - They hold even if you add or remove stuttering steps
- Examples
  - P if P is a state predicate
  - $\Box P$ if P is a state predicate
  - $\Box[A]_v$ if A is an action and v is a state variable (or even state function)
- But not
  - $x' = x + 1$ not satisfied by $[x = 1] \longrightarrow [x = 1] \longrightarrow [x = 2]$
  - $[x' = x + 1\,]_x$ satisfied by $[x = 1] \longrightarrow [x = 1] \longrightarrow [x = 3]$
    but not by $[x = 1] \longrightarrow [x = 3]$
- Yet $\Box[x' = x + 1\,]_x$ is a TLA+ formula!

HourClock revisitied

Module HourClock

hr is a parameter of the specification HourClock

- **Variable $\boldsymbol{hr}$**
- HCini $\triangleq hr \in \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12\}$
- HCnxt $\triangleq hr' = hr \bmod 12 + 1$
- HC $\triangleq$ HCini $\land \square[\text{HCnxt}]_{hr}$

Eventually F

$$\Diamond F \triangleq \neg \Box \neg F$$

$$
\begin{aligned}
& \sigma \models \Diamond F \\
& \equiv \sigma \models \neg \Box \neg F \\
& \equiv \neg (\sigma \models \Box \neg F) \\
& \equiv \neg (\forall n \in \mathbb{N}:\ \sigma^{+n} \models \neg F) \\
& \equiv \neg (\forall n \in \mathbb{N}:\ \neg (\sigma^{+n} \models F)) \\
& \equiv \exists n \in \mathbb{N}:\ (\sigma^{+n} \models F)
\end{aligned}
$$

Eventually an A step occurs…

$$\Diamond \langle A \rangle_v \triangleq \neg\Box[\neg A]_v$$

$$\sigma \vDash \Diamond \langle A \rangle_v$$

$$\equiv \sigma \vDash \neg\Box[\neg A]_v$$

$$\equiv \neg(\sigma \vDash \Box[\neg A]_v)$$

$$\equiv \neg(\forall n \in \mathbb{N}: \sigma^{+n} \vDash [\neg A]_v)$$

$$\equiv \neg(\forall n \in \mathbb{N}: \sigma^{+n} \vDash (\neg A \lor v' = v))$$

$$\equiv \exists n \in \mathbb{N}: \sigma^{+n} \vDash (A \land v' \neq v)$$

HourClock with *liveness*

clock that never stops

Module HourClock

- Variable hr
- HCini $\triangleq$ $hr \in \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12\}$
- HCnxt $\triangleq$ $hr' = hr \bmod 12 + 1$
- HC $\triangleq$ HCini $\wedge$ $\Box[\text{HCnxt}]_{hr}$
- LiveHC $\triangleq$ HC $\wedge$ $\Box(\Diamond\langle \text{HCnxt} \rangle_{hr})$

Module Channel with Liveness

Constant $Data$ Variable $chan$

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

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

$Send(d) \triangleq chan.rdy = chan.ack \land chan' =$
$[\,val \mapsto d, rdy \mapsto 1 - chan.rdy, ack \mapsto chan.ack\,]$

$Recv \triangleq chan.rdy \neq chan.ack \land chan' =$
$[\,val \mapsto chan.val, rdy \mapsto chan.rdy, ack \mapsto 1 - chan.ack\,]$

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

$Spec \triangleq Init \land \Box[Next]_{chan}$ $LiveSpec \triangleq Spec \land \Box(\Diamond\langle Next\rangle_{chan})$???

Module Channel with Liveness

Constant $Data$

Variable $chan$

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

$Init \triangleq chan.val \in Data \wedge chan.rdy \in \{ 0, 1 \} \wedge chan.ack = chan.rdy$

$Send(d) \triangleq chan.rdy = chan.ack \wedge chan' =$
$[\, val \mapsto d, rdy \mapsto 1 - chan.rdy, ack \mapsto chan.ack \,]$

$Recv \triangleq chan.rdy \neq chan.ack \wedge char$
$[\, val \mapsto chan.val, rd$ $- chan.ack \,]$

Too Strong --- If nothing to send that should be ok

$Next \triangleq \exists d \in Data: Send(d) \vee Recv$

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

$LiveSpec \triangleq Spec \wedge \Box(\Diamond\langle Next\rangle_{chan})$???

Module Channel with Liveness

Constant $Data$ Variable $chan$

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

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

$$Send(d) \triangleq chan.rdy = chan.ack \land chan' = [val \longmapsto d, rdy \longmapsto 1 - chan.rdy, ack \longmapsto chan.ack]$$

$$Recv \triangleq chan.rdy \neq chan.ack \land chan' = [val \longmapsto chan.val, rdy \longmapsto chan.rdy, ack \longmapsto 1 - chan.ack]$$

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

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

$LiveSpec \triangleq Spec \land \Box(chan.rdy \neq chan.ack \Rightarrow \Diamond\langle Recv\rangle_{chan})$

Weak Fairness as a liveness condition

- $\text{ENABLED}\ \langle A \rangle_v$ means action A is possible in some state
  - State predicate conjuncts all hold
- $WF_v(A) \triangleq \Box(\Box\text{ENABLED}\ \langle A \rangle_v \ \Rightarrow \Diamond\langle A \rangle_v)$

- HourClock: $WF_{hr}(HCnxt)$
- Channel: $WF_{hr}(Recv)$

(surprising) Weak Fairness equivalence

- $WF_v(A) \triangleq \Box(\Box\text{ENABLED } \langle A \rangle_v \Rightarrow \Diamond\langle A \rangle_v$
  $\equiv \Box\Diamond(\neg\text{ENABLED } \langle A \rangle_v) \vee \Box\Diamond\langle A \rangle_v$
  $\equiv \Diamond\Box(\text{ENABLED } \langle A \rangle_v) \Rightarrow \Box\Diamond\langle A \rangle_v$

- Always, if A is enabled forever, then an A step eventually occurs
- A if infinitely often disabled or infinitely many A steps occur
- If A is eventually enabled forever then infinitely many A steps occur

Strong Fairness

- $SF_v(A) \triangleq \Diamond\Box(\neg \text{ENABLED } \langle A\rangle_v) \lor \Box\Diamond\langle A\rangle_v$

  $\equiv \Box\Diamond(\text{ENABLED } \langle A\rangle_v) \Rightarrow \Box\Diamond\langle A\rangle_v$

- A is eventually disabled forever or infinitely many A steps occur
- If A is infinitely often enabled then infinitely many A steps occur

$SF_v(A)$: an A step must occur if A is continually enabled

$WF_v(A)$: an A step must occur if A is continuously enabled

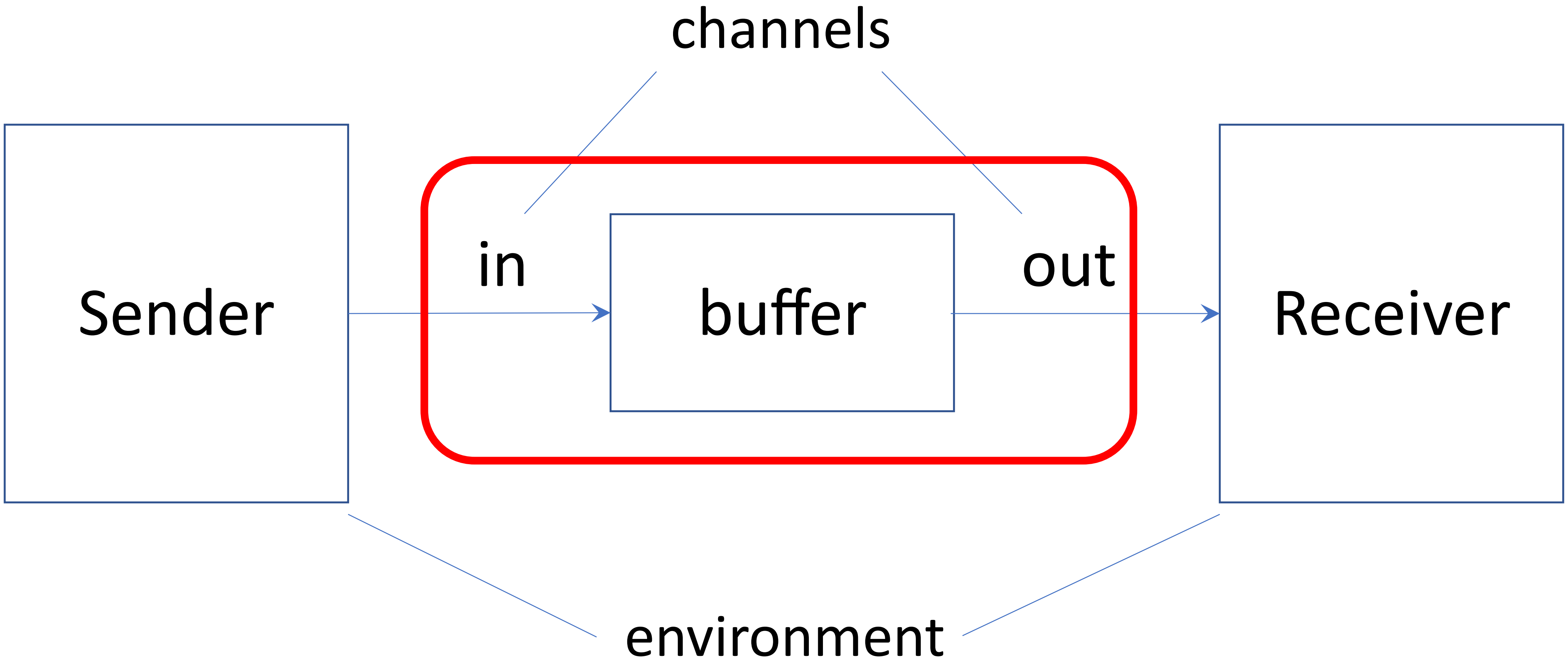
As always, better to make the weaker assumption if you can

How important is liveness?

- Liveness rules out behaviors that have only stuttering steps
  - Add non-triviality of a specification
- In practice, “eventual” is often not good enough
- Instead, need to specify performance requirements
  - Service Level Objectives (SLOs)
  - Usually done quite informally

A “FIFO” (async buffered FIFO channel)

Chapter 4 from Specifying Systems


channels

Sender

in

buffer

out

Receiver

environment

Module Channel

Constant $\boldsymbol{Data}$ **Variable $\boldsymbol{chan}$**

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

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

$$Send(d) \triangleq chan.rdy = chan.ack \land chan' = \\ [\, val \mapsto d, rdy \mapsto 1 - chan.rdy, ack \mapsto chan.ack \,]$$

$$Recv \triangleq chan.rdy \neq chan.ack \land chan' = \\ [\, val \mapsto chan.val, rdy \mapsto chan.rdy, ack \mapsto 1 - chan.ack \,]$$

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

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

Instantiating a Channel

$$InChan \triangleq \text{INSTANCE } Channel \text{ WITH } Data \leftarrow Message, chan \leftarrow in$$

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

$$\Downarrow$$

$$InChan!TypeInvariant \equiv in \in [val: Message, rdy: \{0,1\}, ack: \{0,1\}]$$

Instantiation is Substitution!

MODULE *InnerFIFO*

EXTENDS *Naturals*, *Sequences*
CONSTANT *Message*
VARIABLES *in*, *out*, *q*

$InChan \triangleq$ INSTANCE *Channel* WITH $Data \leftarrow Message,\ chan \leftarrow in$
$OutChan \triangleq$ INSTANCE *Channel* WITH $Data \leftarrow Message,\ chan \leftarrow out$

$$\begin{aligned} Init \triangleq\ & \land\ InChan!Init \\ & \land\ OutChan!Init \\ & \land\ q = \langle\,\rangle \end{aligned}$$

$$\begin{aligned} TypeInvariant \triangleq\ & \land\ InChan!TypeInvariant \\ & \land\ OutChan!TypeInvariant \\ & \land\ q \in Seq(Message) \end{aligned}$$

$$\begin{aligned} SSend(msg) \triangleq\ & \land InChan!Send(msg) \\ & \land \text{UNCHANGED } \langle out, q \rangle \end{aligned}$$

Send *msg* on channel *in*.

$$\begin{aligned} BufRcv \triangleq\ & \land InChan!Rcv \\ & \land q' = Append(q, in.val) \\ & \land \text{UNCHANGED } out \end{aligned}$$

Receive message from channel *in* and append it to tail of q.

$$\begin{aligned} BufSend \triangleq\ & \land q \neq \langle \rangle \\ & \land OutChan!Send(Head(q)) \\ & \land q' = Tail(q) \\ & \land \text{UNCHANGED } in \end{aligned}$$

Enabled only if q is nonempty. Send $Head(q)$ on channel *out* and remove it from q.

$$\begin{aligned} RRcv \triangleq\ & \land OutChan!Rcv \\ & \land \text{UNCHANGED } \langle in, q \rangle \end{aligned}$$

Receive message from channel *out*.

$$\begin{aligned} Next \;&\triangleq\; \lor\ \exists\, msg \in Message : SSend(msg) \\ &\;\; \lor\ BufRcv \\ &\;\; \lor\ BufSend \\ &\;\; \lor\ RRcv \end{aligned}$$

$$Spec \;\triangleq\; Init \land \Box[Next]_{\langle in,\, out,\, q \rangle}$$

$$\textsc{theorem}\ Spec \Rightarrow \Box\, TypeInvariant$$

Parametrized Instantiation

(not *parameterized* instantiation ☺)

$InChan \triangleq \text{INSTANCE } Channel \text{ WITH } Data \leftarrow Message, chan \leftarrow in$

⬇

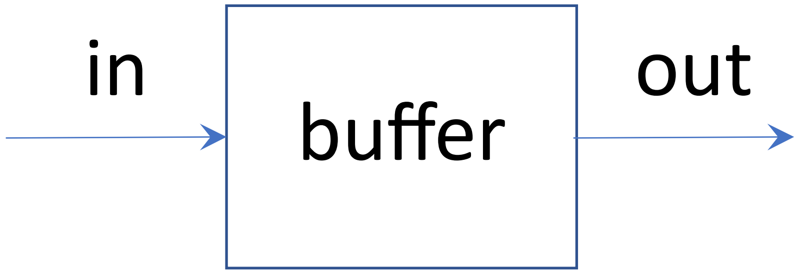
$Chan(ch) \triangleq \text{INSTANCE } Channel \text{ WITH } Data \leftarrow Message, chan \leftarrow ch$

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

⬇

$\mathit{Chan(in)!TypeInvariant} \equiv in \in [val: Message, rdy: \{0,1\}, ack: \{0,1\}]$

Internal (= Non-Interface) Variables



There is no *q* here

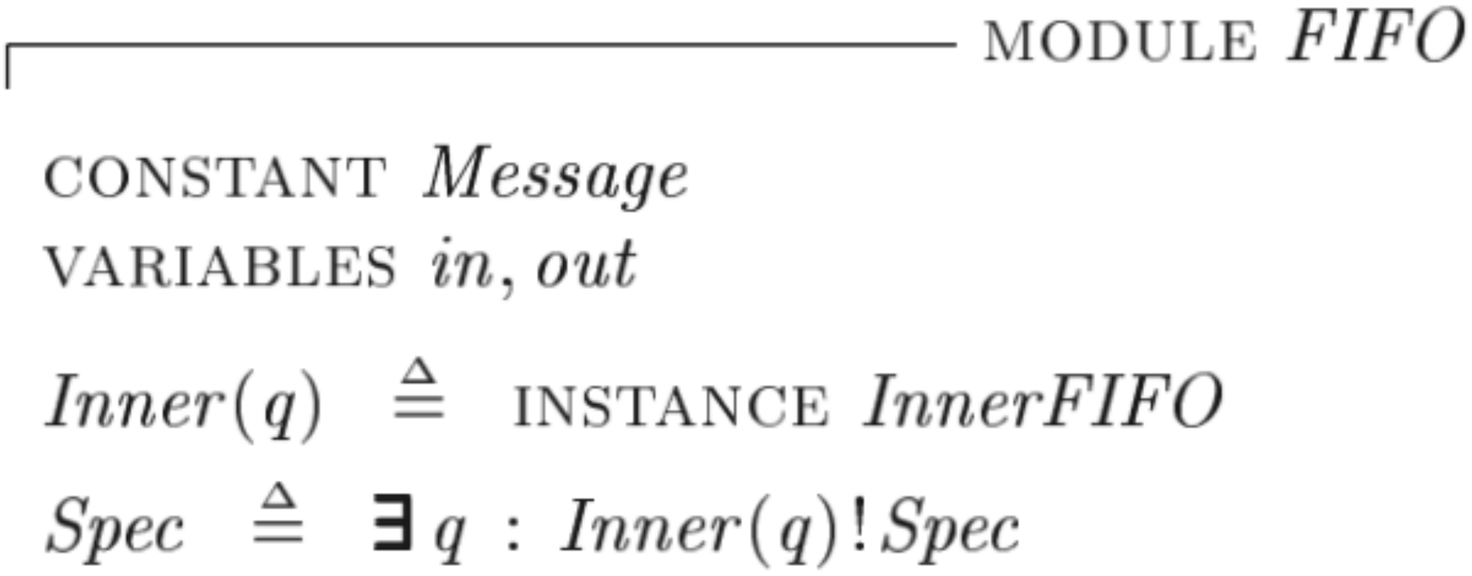
MODULE *InnerFIFO*

EXTENDS *Naturals*, *Sequences*
CONSTANT *Message*
VARIABLES *in*, *out*, *q*

But there is a *q* here

Not incorrect, but don’t really want *q* to be a specification parameter

Hiding Internal Variables



MODULE *FIFO*

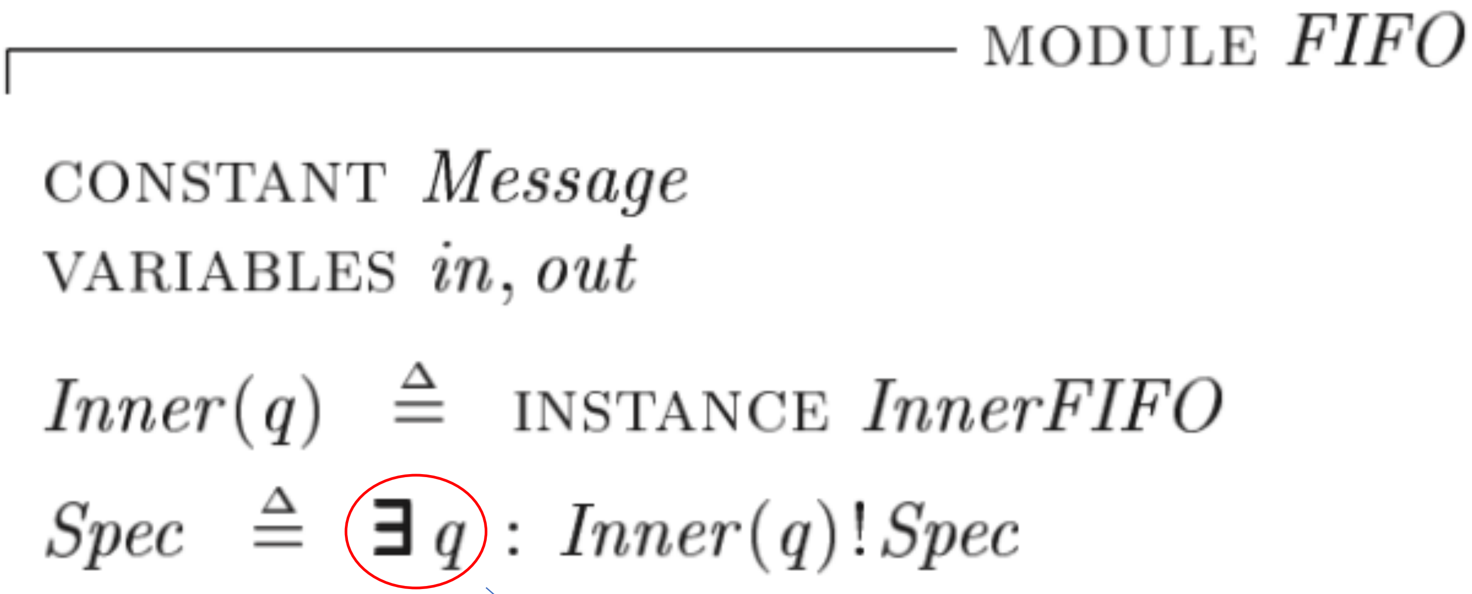
CONSTANT *Message*

VARIABLES *in*, *out*

$Inner(q) \triangleq$ INSTANCE *InnerFIFO*

$Spec \triangleq \exists\!\!\!\exists\, q : Inner(q)!Spec$

Hiding Internal Variables



MODULE *FIFO*

CONSTANT *Message*

VARIABLES *in*, *out*

$Inner(q) \triangleq$ INSTANCE *InnerFIFO*

$Spec \triangleq \boldsymbol{\exists}\, q : Inner(q)!Spec$

Not the normal existential quantifier!!!

In temporal logic, this means that for every state in a behavior, there is a value for *q* that makes *Inner(q)!Spec* true

Pretty. Now for something cool!

- Suppose we wanted to implemented a bounded buffer
- That is, $\square len(q) \leq N$ for some constant $N > 0$
- The only place where q is extended is in *BufRcv*

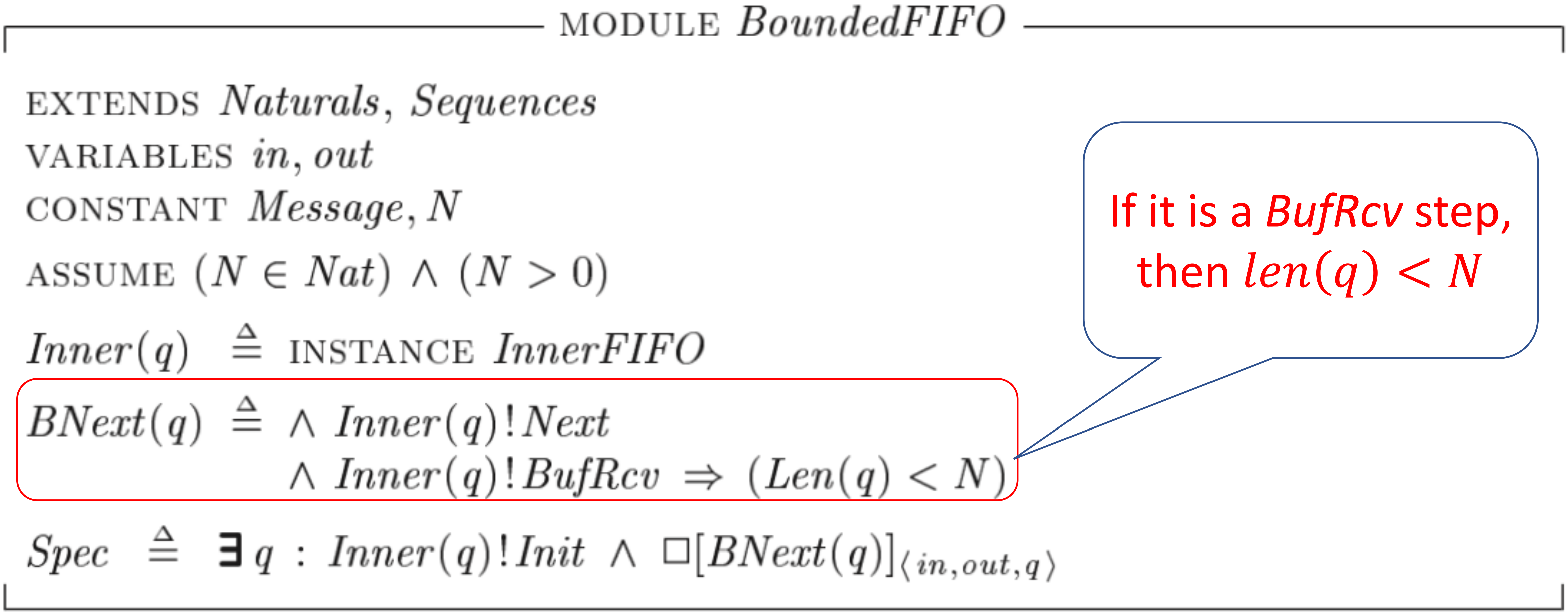
$$
\begin{aligned}
BufRcv \;\triangleq\; & \land\ InChan!Rcv \\
& \land\ q' = Append(q,\ in.val) \\
& \land\ \text{UNCHANGED}\ out
\end{aligned}
$$

Pretty. Now for something cool!

- Suppose we wanted to implemented a bounded buffer
- That is, $\Box len(q) \leq N$ for some constant $N > 0$
- The only place where q is extended is in *BufRcv*

$$
\begin{aligned}
BufRcv \;\triangleq\; & \land\; InChan!Rcv \\
& \land\; q' = Append(q,\ in.val) \\
& \land\; \text{UNCHANGED}\ out \\
& \land\; len(q) < N
\end{aligned}
$$

Even cooler (but tricky)



MODULE *BoundedFIFO*

EXTENDS *Naturals*, *Sequences*
VARIABLES *in*, *out*
CONSTANT *Message*, *N*

ASSUME $(N \in Nat) \land (N > 0)$

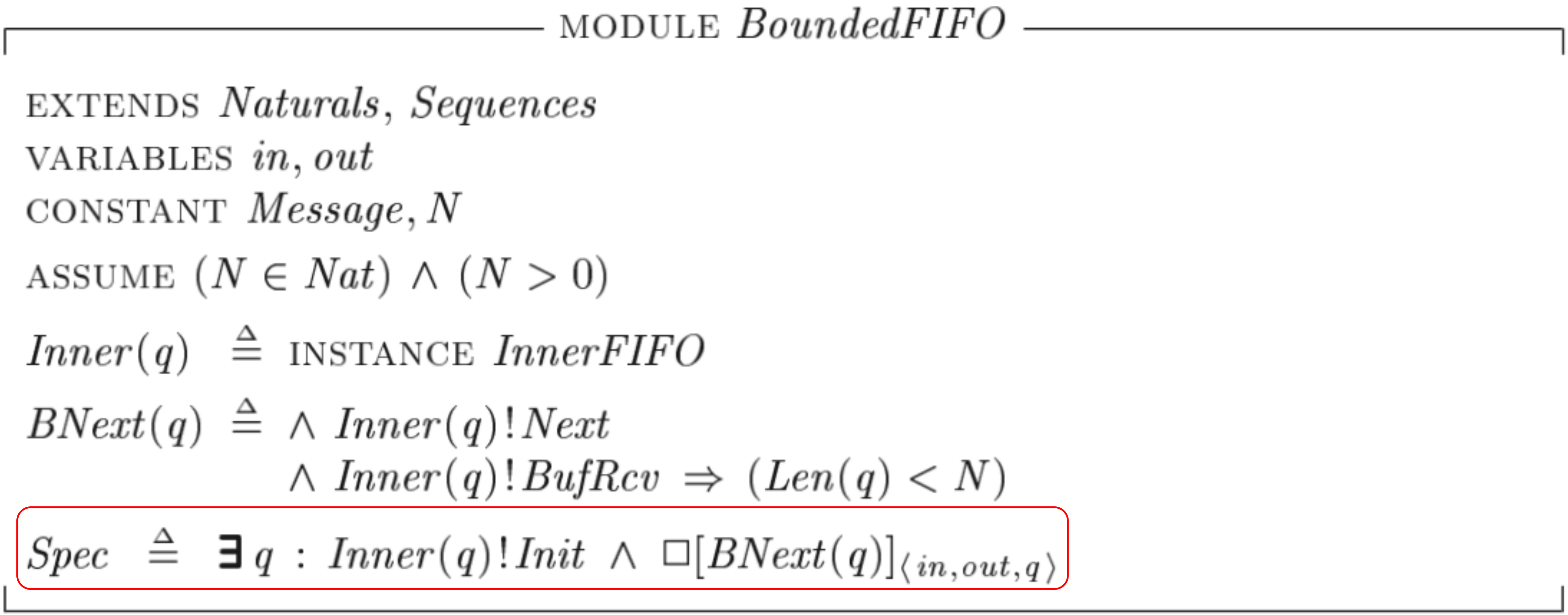
$$Inner(q) \triangleq \text{INSTANCE } InnerFIFO$$

$$\begin{aligned} BNext(q) \triangleq\ & \land\ Inner(q)!Next \\ & \land\ Inner(q)!BufRcv \Rightarrow (Len(q) < N) \end{aligned}$$

$$Spec \triangleq \boldsymbol{\exists}\, q : Inner(q)!Init \land \Box[BNext(q)]_{\langle in, out, q \rangle}$$

If it is a *BufRcv* step, then $len(q) < N$

Even cooler (but tricky)



MODULE *BoundedFIFO*

EXTENDS *Naturals*, *Sequences*
VARIABLES *in*, *out*
CONSTANT *Message*, *N*

$$\text{ASSUME } (N \in Nat) \land (N > 0)$$

$$Inner(q) \triangleq \text{INSTANCE } InnerFIFO$$

$$BNext(q) \triangleq \begin{array}{l} \land\ Inner(q)!Next \\ \land\ Inner(q)!BufRcv \Rightarrow (Len(q) < N) \end{array}$$

$$Spec \triangleq \boldsymbol{\exists}\, q : Inner(q)!Init \land \Box[BNext(q)]_{\langle in, out, q \rangle}$$