Embedded Systems Specification and Design
Model-based Design and Verification

David Kendall

Temporal Formulas

You will have noticed in the Pluscal translations of your algorithms, definitions such as

```
Spec == Init /\ [][Next]_vars
```

where `vars` is defined as a tuple of the variables declared in your specification, e.g.

```
vars == << balance, pc, current >>
```

appears in the BankAccount specification

In fact, *all* of the Pluscal translations contain exactly the same specification of `Spec`

The definition of `Spec` is an example of a *temporal formula*

TLA$^+$ is the **T**emporal **L**ogic of **A**ctions

TLA$^+$ seeks to minimise the need to use temporal formulas in specifications but cannot avoid them entirely, since we need to be able to talk about behaviours, not just states

The purpose of this lecture is to explain the meaning of `[][Next]_vars` and, more generally, the meaning of *temporal formulas* in TLA$^+$

Example - HourClock

The pattern of

```
Spec == Init /\ [][Next]_vars
```

can be seen clearly in the translation of a Pluscal version of Lamport's hour clock example

This emphasises the fact that a TLA$^+$ specification is the conjunction of a *state formula*, `Init`, that determines the *initial states*, and an *action formula*, `Next`, that determines what *steps* are allowed from one state to another

It specifies a system with only one state variable, `hr`, which initially takes any value in the set `1..12`, and then is incremented at each *step* until it becomes `12`, at which point it is reset to `1` on the next step, and the process continues

HourClock - Pluscal

```
---- MODULE PCalHourClock ----
EXTENDS Naturals
(*
--algorithm PCalHourClock
  variable
    hr \in 1..12;
  begin
    while TRUE do
      if hr < 12 then
        hr := hr + 1;
      else
        hr := 1;
      end if
    end while
  end algorithm
*)
```

HourClock - Translation

```
\* BEGIN TRANSLATION
VARIABLE hr

vars == << hr >>

Init == (* Global variables *)
        /\ hr \in 1..12

Next == IF hr < 12
           THEN /\ hr' = hr + 1
           ELSE /\ hr' = 1

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

\* END TRANSLATION
====
```

The Standard Model

- A state is an assignment of values to variables
- A state expression is an expression that may involve the (unprimed) variables of the state
- A state formula is a boolean-valued state expression
- A behaviour is a sequence of states
- A system is modelled as a collection of behaviours

Behaviours

Definition (Behaviour)

Given a specification S, a behaviour of S is an infinite sequence of states $s_0, s_1, s_2, \ldots$ such that s_0 satisfies *Init* and for $i \in Nat$, $s_i \longrightarrow s_{i+1}$ satisfies $[Next]_{vars}$

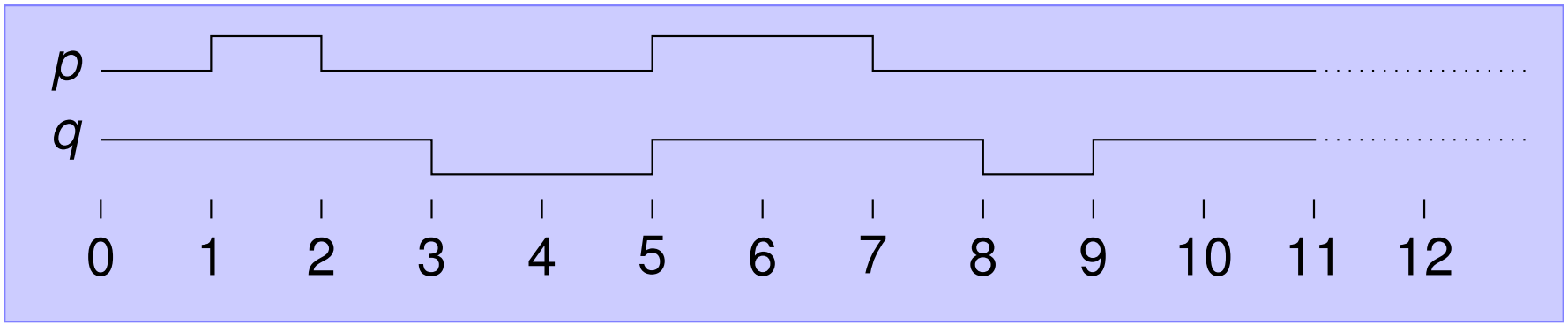
Example

$$[hr : 1] \longrightarrow [hr : 2] \longrightarrow [hr : 3] \longrightarrow [hr : 4] \longrightarrow [hr : 5] \longrightarrow \cdots$$

is the prefix of a possible behaviour of PCalHourClock

Timing diagrams

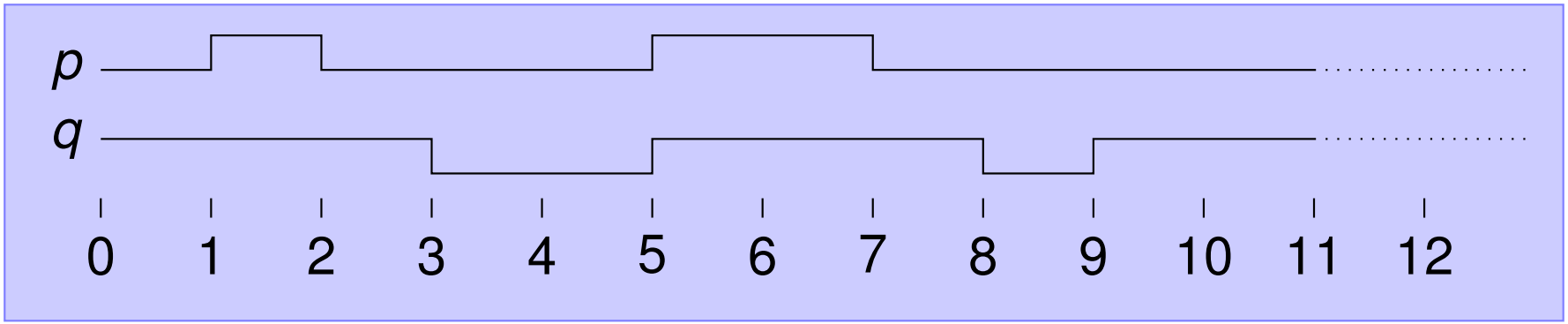
- The truth value of state formulas depend upon the values of the variables in the current state
- Their truth values can change from state to state
- Timing diagrams can be a useful tool for visualising the changing values of state formulas over time



- Each state formula's value is shown as a line of the timing diagram.
- Line high – state formula is true.
- Line low – state formula is false.

State formulas

- State formulas are formed from simple propositions about states using the logical operators $(\wedge, \vee, \Rightarrow, \Leftrightarrow, \neg)$
- For a behaviour s and state formula f, we can ask whether f is true at some state s_i in s, if it is we write $s_i \models f$



For which i do we have (a) $s_i \models p \wedge q$, (b) $s_i \models p \vee q$, and (c) $s_i \models p \Rightarrow q$?

Temporal Operators

So far, we can say what is true at each individual state but we can't say anything about what is true of a behaviour as a whole

How can we say something about behaviours as they evolve over time?

We introduce *temporal operators*

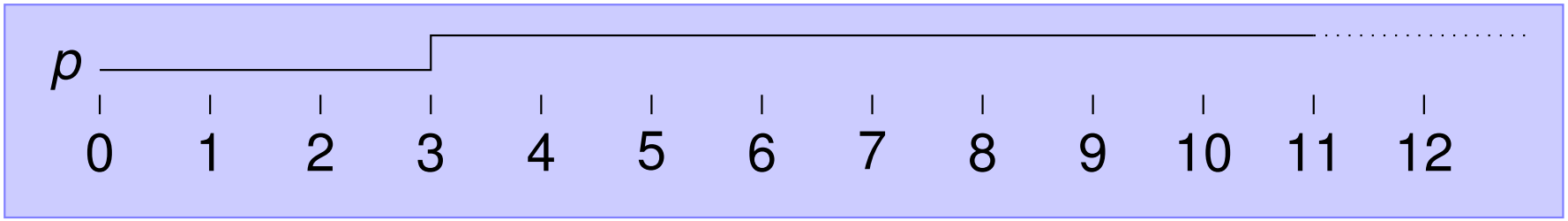
TLA$^+$ uses the temporal operators $\Box$ (*always*) and $\Diamond$ (*eventually*)

- $\Box$ – **Always**: $\Box\phi$ is true at state *i* in a behaviour *s* if ϕ is true in state *i* and every succeeding state in *s*
- $\Diamond$ – **Eventually**: $\Diamond\phi$ is true at state *i* in a behaviour *s* if ϕ is true at state *i* or at some succeeding state in *s*

Always ($\Box\phi$)

Definition (Always)

$s_i \models \Box\phi$ iff $\forall j \geq i \bullet s_j \models \phi$

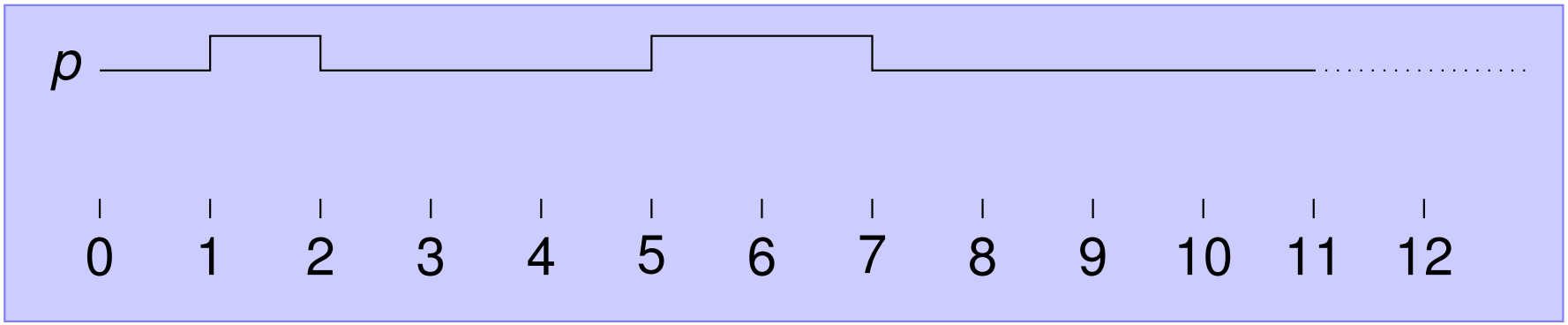


- $s_3 \models \Box p$
- $s_0 \not\models \Box p$

Eventually ($\Diamond\phi$)

Definition (Eventually)

$s_i \models \Diamond\phi$ iff $\exists j \geq i \bullet s_j \models \phi$



- $s_0 \models \Diamond p$
- $s_6 \models \Diamond p$
- $s_7 \not\models \Diamond p$

Examples of temporal formulas

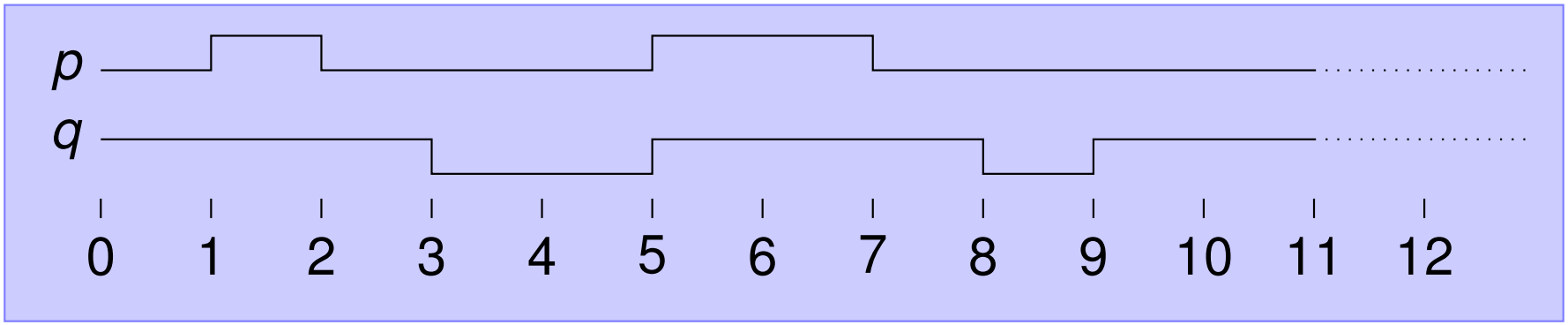
Now we can make precise claims about the behaviours of our hour clock, e.g.

"The value of `hr` is always a natural number between 1 and 12"
$\Box(hr \in 1..12)$

"The value of `hr` is eventually 7"
$\Diamond(hr = 7)$

Exercises

- Which of the following claims are true of the trace shown in the diagram below?
  1. $s_9 \models \Box q$
  2. $s_9 \models \Box \neg p$
  3. $s_0 \models \Box q$
  4. $s_0 \models \Box (p \Rightarrow q)$
  5. $s_9 \models \Diamond p$
  6. $s_0 \models \Diamond q$
  7. $s_0 \models \Diamond \neg (p \Rightarrow q)$



Actions

TLA$^+$ allows us to make statements about the relationship of successive states in a behaviour

It uses *primed* variables to do this, e.g.
$x' = x + 1$
says that the value of x in the next state is equal to the value of x in the current state plus 1

In TLA$^+$, any ordinary mathematical expression thta includes primed and unprimed variables is a *transition expression*

An *action* is a boolean-valued transition expression

We can interpret an action, A, as a temporal formula by defining $s \models A$ to be true iff the first two states of s are an A-step, i.e. iff $s_0 \longrightarrow s_1$ is an A-step

A behaviour $s \models \Box A$ iff $s_n \longrightarrow s_{n+1}$, for all natural numbers n

More on temporal formulas with $\Box$

For a behaviour, s, define s^{+n} to be the suffix of s obtained by deleting its first n states

$$s^{+n} \quad \widehat{=} \quad s_n \longrightarrow s_{n+1} \longrightarrow s_{n+2} \cdots$$

So $s_n \longrightarrow s_{n+1}$ is the first step of s^{+n}, and therefore $s \models \Box A$ is true iff $s^{+n} \models A$ is true for all n. Formally,

$$s \models \Box A \quad \Leftrightarrow \quad \forall\, n \in Nat : s^{+n} \models A$$

This can be generalised to any temporal formula F

$$s \models \Box F \quad \widehat{=} \quad \forall\, n \in Nat : s^{+n} \models F$$

Stuttering

So why isn't the specification of our hour clock defined as

$$Spec \quad \widehat{=} \quad Init \wedge \Box Next$$

. . . because this specification would not allow *stuttering steps* and TLA$^+$ requires our specifications to allow stuttering steps

A stuttering step is a step that leaves all of a specification's variables unchanged

$$[Next]_{vars} \quad \widehat{=} \quad Next \vee \text{UNCHANGED } vars$$

So our specification says that any behaviour of the hour clock is one that satisfies *Init* (*Init* is true at s_0) and in which any step $s_n \longrightarrow s_{n+1}$ is either a *Next* step or it leaves the value of *hr* unchanged

We won't go into why insisting on allowing stuttering is a good idea here

Checking temporal properties

We can define a property, *TypeOk*, for the hour clock as

$$TypeOk \quad \widehat{=} \quad hr \in 1 \,..\, 12$$

and then use TLC to check the temporal property $Spec \Rightarrow \Box TypeOk$ — this is equivalent to checking that *TypeOk* is an invariant of *Spec*

TypeOk is an example of a *safety property*, i.e. a property, which if true, shows that some "bad thing" never happens

We can also state a temporal property such as

$$Spec \Rightarrow \Diamond(hr = 12)$$

i.e. eventually the hour clock reaches noon

$Spec \Rightarrow \Diamond(hr = 12)$ is an example of a *liveness property*, i.e. a property, which if true, shows that some "good thing" does eventually happen

Fairness

Demo: Use TLC to check the previous 2 properties

We see that the liveness property is not satisfied because *Spec* can stutter forever, i.e. carry on taking steps in which the *hr* variable remains unchanged

We can require that these behaviours are excluded by adding a *fairness* requirement to the specification

In Pluscal, we do this by adding the word `fair` in front of `algorithm`.

Demo: Try this for the hour clock spec

Fairness (ctd)

In TLA$^+$, we do this by adding $\mathrm{WF}_{vars}(Next)$ to the definition of *Spec*

Let $\langle A \rangle_{vars} \triangleq A \wedge (vars' \neq vars)$, then $\langle A \rangle_{vars}$ is an A-step that changes *vars*, and if *vars* is the tuple of all system variables, then $\langle A \rangle_{vars}$ is a non-stuttering A-step

The *weak fairness* formula $\mathrm{WF}_{vars}(A)$ is then defined by

Definition (Weak fairness)

$\mathrm{WF}_{vars}(A)$ is satisfied by a behaviour $s_0 \longrightarrow s_1 \longrightarrow s_2 \cdots$ iff there is no n such that $s_n \longrightarrow s_{n+1} \cdots$ has no $\langle A \rangle_{vars}$ step but $\langle A \rangle_{vars}$ is enabled in all of its states

Classifying temporal properties

- **Safety** – “Nothing bad happens”
  $\Box\neg(\texttt{in\_cs1} \wedge \texttt{in\_cs2})$
- **Liveness** – “Something good happens”
  $\Box(\texttt{request} \Rightarrow \Diamond\texttt{receive})$
- **Fairness** – “Computation is fair”
  $\Box\Diamond\texttt{coffee\_ordered} \Rightarrow \Box\Diamond\texttt{coffee\_delivered}$
  - This is an example of *strong* fairness

Examples of typical specification patterns

- Invariance ($\Box\neg\, P$)
  $\Box\neg\,(\texttt{systemState} == \texttt{DANGEROUS})$
  $\Box\neg\,((\texttt{crossing} == \texttt{OCCUPIED}) \wedge (\texttt{gate} == \texttt{OPEN}))$
- Bounded Response ($\Box(P \Rightarrow \Diamond Q)$)
  $\Box(\texttt{waitingForService} \Rightarrow \Diamond \texttt{serviceReceived})$
  $\Box(\texttt{facingObstacle} \Rightarrow \Diamond(\texttt{distanceFromObstacle} >= 5))$
- Nearly all of the specifications that you write will be either an *invariance* property or a *bounded response* property.
- More practice in writing temporal specifications to come in the lab sessions.

Acknowledgements

These slides are based on material in Barland, I.; Vardi, M.; Greiner, J. *Model Checking Concurrent Programs*, Connexions Web site. http://cnx.org/content/col10294/1.3/, Oct 27, 2005. and Lamport's hyperbook.