To prove equivalence (3.2), suppose first that a path satisfies $\phi \,\mathrm{U}\, \psi$. Then, from clause 11, we have $i \geq 1$ such that $\pi^i \vDash \psi$ and for all $j = 1, \ldots, i-1$ we have $\pi^j \vDash \phi$. From clause 12, this proves $\phi \,\mathrm{W}\, \psi$, and from clause 10 it proves $\mathrm{F}\, \psi$. Thus for all paths π, if $\pi \vDash \phi \,\mathrm{U}\, \psi$ then $\pi \vDash \phi \,\mathrm{W}\, \psi \wedge \mathrm{F}\, \psi$. As an exercise, the reader can prove it the other way around.

Writing W in terms of U is also possible: W is like U but also allows the possibility of the eventuality never occurring:

$$\phi \,\mathrm{W}\, \psi \equiv \phi \,\mathrm{U}\, \psi \vee \mathrm{G}\, \phi. \tag{3.3}$$

Inspection of clauses 12 and 13 reveals that R and W are rather similar. The differences are that they swap the roles of their arguments ϕ and ψ; and the clause for W has an $i-1$ where R has i. Therefore, it is not surprising that they are expressible in terms of each other, as follows:

$$\phi \,\mathrm{W}\, \psi \equiv \psi \,\mathrm{R}\, (\phi \vee \psi) \tag{3.4}$$

$$\phi \,\mathrm{R}\, \psi \equiv \psi \,\mathrm{W}\, (\phi \wedge \psi). \tag{3.5}$$

3.2.5 Adequate sets of connectives for LTL

Recall that $\phi \equiv \psi$ holds iff any path in any transition system which satisfies ϕ also satisfies ψ, and vice versa. As in propositional logic, there is some redundancy among the connectives. For example, in Chapter 1 we saw that the set $\{\bot, \wedge, \neg\}$ forms an adequate set of connectives, since the other connectives $\vee$, $\rightarrow$, $\top$, etc., can be written in terms of those three.

Small adequate sets of connectives also exist in LTL. Here is a summary of the situation.

- X is completely orthogonal to the other connectives. That is to say, its presence doesn't help in defining any of the other ones in terms of each other. Moreover, X cannot be derived from any combination of the others.
- Each of the sets $\{\mathrm{U}, \mathrm{X}\}$, $\{\mathrm{R}, \mathrm{X}\}$, $\{\mathrm{W}, \mathrm{X}\}$ is adequate. To see this, we note that
  - R and W may be defined from U, by the duality $\phi \,\mathrm{R}\, \psi \equiv \neg(\neg\phi \,\mathrm{U}\, \neg\psi)$ and equivalence (3.4) followed by the duality, respectively.
  - U and W may be defined from R, by the duality $\phi \,\mathrm{U}\, \psi \equiv \neg(\neg\phi \,\mathrm{R}\, \neg\psi)$ and equivalence (3.4), respectively.
  - R and U may be defined from W, by equivalence (3.5) and the duality $\phi \,\mathrm{U}\, \psi \equiv \neg(\neg\phi \,\mathrm{R}\, \neg\psi)$ followed by equivalence (3.5).

Sometimes it is useful to look at adequate sets of connectives which do not rely on the availability of negation. That's because it is often convenient to assume formulas are written in negation-normal form, where all the negation symbols are applied to propositional atoms (i.e., they are near the leaves

of the parse tree). In this case, these sets are adequate for the fragment without X, and no strict subset is: $\{\mathrm{U}, \mathrm{R}\}$, $\{\mathrm{U}, \mathrm{W}\}$, $\{\mathrm{U}, \mathrm{G}\}$, $\{\mathrm{R}, \mathrm{F}\}$, $\{\mathrm{W}, \mathrm{F}\}$. But $\{\mathrm{R}, \mathrm{G}\}$ and $\{\mathrm{W}, \mathrm{G}\}$ are not adequate. Note that one cannot define G with {U,F}, and one cannot define F with {R,G} or {W,G}.

We finally state and prove a useful equivalence about U.

Theorem 3.10 The equivalence $\phi \,\mathrm{U}\, \psi \equiv \neg(\neg\psi \,\mathrm{U}\, (\neg\phi \wedge \neg\psi)) \wedge \mathrm{F}\, \psi$ holds for all LTL formulas ϕ and ψ.

PROOF: Take any path $s_0 \rightarrow s_1 \rightarrow s_2 \rightarrow \ldots$ in any model.

First, suppose $s_0 \vDash \phi \,\mathrm{U}\, \psi$ holds. Let n be the smallest number such that $s_n \vDash \psi$; such a number has to exist since $s_0 \vDash \phi \,\mathrm{U}\, \psi$; then, for each $k < n$, $s_k \vDash \phi$. We immediately have $s_0 \vDash \mathrm{F}\, \psi$, so it remains to show $s_0 \vDash \neg(\neg\psi \,\mathrm{U}\, (\neg\phi \wedge \neg\psi))$, which, if we expand, means:
$(*)$ for each $i > 0$, if $s_i \vDash \neg\phi \wedge \neg\psi$, then there is some $j < i$ with $s_j \vDash \psi$.
Take any $i > 0$ with $s_i \vDash \neg\phi \wedge \neg\psi$; $i > n$, so we can take $j \stackrel{\text{def}}{=} n$ and have $s_j \vDash \psi$.
Conversely, suppose $s_0 \vDash \neg(\neg\psi \,\mathrm{U}\, (\neg\phi \wedge \neg\psi)) \wedge \mathrm{F}\, \psi$ holds; we prove $s_0 \vDash \phi \,\mathrm{U}\, \psi$. Since $s_0 \vDash \mathrm{F}\, \psi$, we have a minimal n as before. We show that, for any $i < n$, $s_i \vDash \phi$. Suppose $s_i \vDash \neg\phi$; since n is minimal, we know $s_i \vDash \neg\psi$, so by $(*)$ there is some $j < i < n$ with $s_j \vDash \psi$, contradicting the minimality of n. □

3.3 Model checking: systems, tools, properties

3.3.1 Example: mutual exclusion

Let us now look at a larger example of verification using LTL, having to do with *mutual exclusion*. When concurrent processes share a resource (such as a file on a disk or a database entry), it may be necessary to ensure that they do not have access to it at the same time. Several processes simultaneously editing the same file would not be desirable.

We therefore identify certain *critical sections* of each process' code and arrange that only one process can be in its critical section at a time. The critical section should include all the access to the shared resource (though it should be as small as possible so that no unnecessary exclusion takes place). The problem we are faced with is to find a *protocol* for determining which process is allowed to enter its critical section at which time. Once we have found one which we think works, we verify our solution by checking that it has some expected properties, such as the following ones:

Safety: Only one process is in its critical section at any time.

```
MODULE one-bit-chan(input)
VAR
   output : boolean;
   forget : boolean;
ASSIGN
   next(output) := case
                     forget : output;
                     1:       input;
                   esac;
FAIRNESS running
FAIRNESS input & !forget
FAIRNESS !input & !forget

MODULE two-bit-chan(input1,input2)
VAR
   forget : boolean;
   output1 : boolean;
   output2 : boolean;
ASSIGN
   next(output1) := case
                      forget : output1;
                      1:       input1;
                    esac;
   next(output2) := case
                      forget : output2;
                      1:       input2;
                    esac;
FAIRNESS running
FAIRNESS input1 & !forget
FAIRNESS !input1 & !forget
FAIRNESS input2 & !forget
FAIRNESS !input2 & !forget
```

Figure 3.16. The two modules for the two ABP channels in SMV.

contraints of the form 'infinitely often p implies infinitely often q', which would be more satisfactory here, but is not allowed by SMV.

Finally, we tie it all together with the module `main` (Figure 3.17). Its role is to connect together the components of the system, and giving them initial values of their parameters. Since the first control bit is 0, we also initialise the receiver to expect a 0. The receiver should start off by sending 1 as its


```
MODULE main
VAR
   s : process sender(ack_chan.output);
   r : process receiver(msg_chan.output1,msg_chan.output2);
   msg_chan : process two-bit-chan(s.message1,s.message2);
   ack_chan : process one-bit-chan(r.ack);
ASSIGN
   init(s.message2) := 0;
   init(r.expected) := 0;
   init(r.ack)      := 1;
   init(msg_chan.output2) := 1;
   init(ack_chan.output) := 1;

LTLSPEC  G (s.st=sent & s.message1=1 -> msg_chan.output1=1)
```

Figure 3.17. The `main` ABP module.

acknowledgement, so that `sender` does not think that its very first message is being acknowledged before anything has happened. For the same reason, the output of the channels is initialised to 1.

The specifications for ABP. Our SMV program satisfies the following specifications:

Safety: If the message bit 1 has been sent and the correct acknowledgement has been returned, then a 1 was indeed received by the receiver: `G (S.st=sent & S.message1=1 -> msg_chan.output1=1)`.

Liveness: Messages get through eventually. Thus, for any state there is inevitably a future state in which the current message has got through. In the module `sender`, we specified `G F st=sent`. (This specification could equivalently have been written in the main module, as `G F S.st=sent`.) Similarly, acknowledgements get through eventually. In the module `receiver`, we write `G F st=received`.

3.4 Branching-time logic

In our analysis of LTL (*linear-time temporal logic*) in the preceding sections, we noted that LTL formulas are evaluated on *paths*. We defined that a *state* of a system satisfies an LTL formula if *all paths* from the given state satisfy it. Thus, LTL implicitly quantifies universally over paths. Therefore, properties which assert the existence of a path cannot be expressed in LTL. This problem can partly be alleviated by considering the negation of the property in question, and interpreting the result accordingly. To check whether there

exists a path from s satisfying the LTL formula ϕ, we check whether all paths satisfy $\neg\phi$; a positive answer to this is a negative answer to our original question, and vice versa. We used this approach when analysing the ferryman puzzle in the previous section. However, as already noted, properties which *mix* universal and existential path quantifiers cannot in general be model checked using this approach, because the complement formula still has a mix.

Branching-time logics solve this problem by allowing us to quantify explicitly over paths. We will examine a logic known as *Computation Tree Logic*, or CTL. In CTL, as well as the temporal operators U, F, G and X of LTL we also have quantifiers A and E which express 'all paths' and 'exists a path', respectively. For example, we can write:

- There is a reachable state satisfying q: this is written $\mathrm{EF}\, q$.
- From all reachable states satisfying p, it is possible to maintain p continuously until reaching a state satisfying q: this is written $\mathrm{AG}\,(p \to \mathrm{E}[p\ \mathrm{U}\ q])$.
- Whenever a state satisfying p is reached, the system can exhibit q continuously forevermore: $\mathrm{AG}\,(p \to \mathrm{EG}\, q)$.
- There is a reachable state from which all reachable states satisfy p: $\mathrm{EF}\,\mathrm{AG}\, p$.

3.4.1 Syntax of CTL

Computation Tree Logic, or CTL for short, is a *branching-time* logic, meaning that its model of time is a tree-like structure in which the future is not determined; there are different paths in the future, any one of which might be the 'actual' path that is realised.

As before, we work with a fixed set of atomic formulas/descriptions (such as $p, q, r, \ldots$, or $p_1, p_2, \ldots$).

Definition 3.12 We define CTL formulas inductively via a Backus Naur form as done for LTL:

$$\phi ::= \bot \mid \top \mid p \mid (\neg\phi) \mid (\phi \wedge \phi) \mid (\phi \vee \phi) \mid (\phi \to \phi) \mid \mathrm{AX}\,\phi \mid \mathrm{EX}\,\phi \mid \mathrm{AF}\,\phi \mid \mathrm{EF}\,\phi \mid \mathrm{AG}\,\phi \mid \mathrm{EG}\,\phi \mid \mathrm{A}[\phi\ \mathrm{U}\ \phi] \mid \mathrm{E}[\phi\ \mathrm{U}\ \phi]$$

where p ranges over a set of atomic formulas.

Notice that each of the CTL temporal connectives is a pair of symbols. The first of the pair is one of A and E. A means 'along All paths' (*inevitably*) and E means 'along at least (there Exists) one path' (*possibly*). The second one of the pair is X, F, G, or U, meaning 'neXt state,' 'some Future state,' 'all future states (Globally)' and Until, respectively. The pair of symbols in $\mathrm{E}[\phi_1\ \mathrm{U}\ \phi_2]$, for example, is EU. In CTL, pairs of symbols like EU are

indivisible. Notice that AU and EU are binary. The symbols X, F, G and U cannot occur without being preceded by an A or an E; similarly, every A or E must have one of X, F, G and U to accompany it.

Usually weak-until (W) and release (R) are not included in CTL, but they are derivable (see Section 3.4.5).

Convention 3.13 We assume similar binding priorities for the CTL connectives to what we did for propositional and predicate logic. The unary connectives (consisting of $\neg$ and the temporal connectives AG, EG, AF, EF, AX and EX) bind most tightly. Next in the order come $\wedge$ and $\vee$; and after that come $\to$, AU and EU.

Naturally, we can use brackets in order to override these priorities. Let us see some examples of well-formed CTL formulas and some examples which are not well-formed, in order to understand the syntax. Suppose that p, q and r are atomic formulas. The following are well-formed CTL formulas:

- $\mathrm{AG}\,(q \to \mathrm{EG}\, r)$, note that this is not the same as $\mathrm{AG}\, q \to \mathrm{EG}\, r$, for according to Convention 3.13, the latter formula means $(\mathrm{AG}\, q) \to (\mathrm{EG}\, r)$
- $\mathrm{EF}\,\mathrm{E}[r\ \mathrm{U}\ q]$
- $\mathrm{A}[p\ \mathrm{U}\ \mathrm{EF}\, r]$
- $\mathrm{EF}\,\mathrm{EG}\, p \to \mathrm{AF}\, r$, again, note that this binds as $(\mathrm{EF}\,\mathrm{EG}\, p) \to \mathrm{AF}\, r$, not $\mathrm{EF}\,(\mathrm{EG}\, p \to \mathrm{AF}\, r)$ or $\mathrm{EF}\,\mathrm{EG}\,(p \to \mathrm{AF}\, r)$
- $\mathrm{A}[p_1\ \mathrm{U}\ \mathrm{A}[p_2\ \mathrm{U}\ p_3]]$
- $\mathrm{E}[\mathrm{A}[p_1\ \mathrm{U}\ p_2]\ \mathrm{U}\ p_3]$
- $\mathrm{AG}\,(p \to \mathrm{A}[p\ \mathrm{U}\ (\neg p \wedge \mathrm{A}[\neg p\ \mathrm{U}\ q])])$.

It is worth spending some time seeing how the syntax rules allow us to construct each of these. The following are not well-formed formulas:

- $\mathrm{EF}\,\mathrm{G}\, r$
- $\mathrm{A}\neg\mathrm{G}\,\neg p$
- $\mathrm{F}\,[r\ \mathrm{U}\ q]$
- $\mathrm{EF}\,(r\ \mathrm{U}\ q)$
- $\mathrm{AEF}\, r$
- $\mathrm{A}[(r\ \mathrm{U}\ q) \wedge (p\ \mathrm{U}\ r)]$.

It is especially worth understanding why the syntax rules don't allow us to construct these. For example, take $\mathrm{EF}\,(r\ \mathrm{U}\ q)$. The problem with this string is that U can occur only when paired with an A or an E. The E we have is paired with the F. To make this into a well-formed CTL formula, we would have to write $\mathrm{EF}\,\mathrm{E}[r\ \mathrm{U}\ q]$ or $\mathrm{EF}\,\mathrm{A}[r\ \mathrm{U}\ q]$.