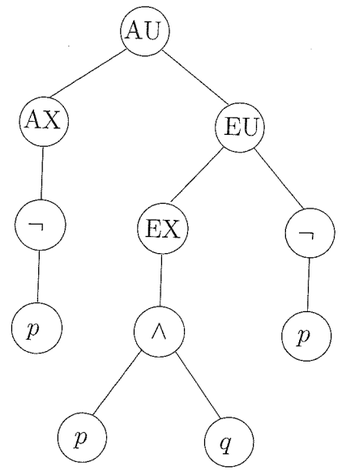

Figure 3.18. The parse tree of a CTL formula without infix notation.

Notice that we use square brackets after the A or E, when the paired operator is a U. There is no strong reason for this; you could use ordinary round brackets instead. However, it often helps one to read the formula (because we can more easily spot where the corresponding close bracket is). Another reason for using the square brackets is that SMV insists on it.

The reason $\mathrm{A}[(r\ \mathrm{U}\ q) \wedge (p\ \mathrm{U}\ r)]$ is not a well-formed formula is that the syntax does not allow us to put a boolean connective (like $\wedge$) directly inside A[ ] or E[ ]. Occurrences of A or E must be followed by one of G, F, X or U; when they are followed by U, it must be in the form $\mathrm{A}[\phi\ \mathrm{U}\ \psi]$. Now, the ϕ and the ψ *may* contain $\wedge$, since they are arbitrary formulas; so $\mathrm{A}[(p \wedge q)\ \mathrm{U}\ (\neg r \rightarrow q)]$ *is* a well-formed formula.

Observe that AU and EU are binary connectives which mix infix and prefix notation. In pure infix, we would write $\phi_1\ \mathrm{AU}\ \phi_2$, whereas in pure prefix we would write $\mathrm{AU}(\phi_1, \phi_2)$.

As with any formal language, and as we did in the previous two chapters, it is useful to draw parse trees for well-formed formulas. The parse tree for $\mathrm{A}[\mathrm{AX}\,\neg p\ \mathrm{U}\ \mathrm{E}[\mathrm{EX}\,(p \wedge q)\ \mathrm{U}\ \neg p]]$ is shown in Figure 3.18.

Definition 3.14 A subformula of a CTL formula ϕ is any formula ψ whose parse tree is a subtree of ϕ's parse tree.

3.4.2 Semantics of computation tree logic

CTL formulas are interpreted over transition systems (Definition 3.4). Let $\mathcal{M} = (S, \rightarrow, L)$ be such a model, $s \in S$ and ϕ a CTL formula. The definition of whether $\mathcal{M}, s \vDash \phi$ holds is recursive on the structure of ϕ, and can be roughly understood as follows:

- If ϕ is atomic, satisfaction is determined by L.
- If the top-level connective of ϕ (i.e., the connective occurring top-most in the parse tree of ϕ) is a boolean connective ($\wedge$, $\vee$, $\neg$, $\top$ etc.) then the satisfaction question is answered by the usual truth-table definition and further recursion down ϕ.
- If the top level connective is an operator beginning A, then satisfaction holds if all paths from s satisfy the 'LTL formula' resulting from removing the A symbol.
- Similarly, if the top level connective begins with E, then satisfaction holds if some path from s satisfy the 'LTL formula' resulting from removing the E.

In the last two cases, the result of removing A or E is not strictly an LTL formula, for it may contain further As or Es below. However, these will be dealt with by the recursion.

The formal definition of $\mathcal{M}, s \vDash \phi$ is a bit more verbose:

Definition 3.15 Let $\mathcal{M} = (S, \rightarrow, L)$ be a model for CTL, s in S, ϕ a CTL formula. The relation $\mathcal{M}, s \vDash \phi$ is defined by structural induction on ϕ:

1. $\mathcal{M}, s \vDash \top$ and $\mathcal{M}, s \nvDash \bot$
2. $\mathcal{M}, s \vDash p$ iff $p \in L(s)$
3. $\mathcal{M}, s \vDash \neg\phi$ iff $\mathcal{M}, s \nvDash \phi$
4. $\mathcal{M}, s \vDash \phi_1 \wedge \phi_2$ iff $\mathcal{M}, s \vDash \phi_1$ and $\mathcal{M}, s \vDash \phi_2$
5. $\mathcal{M}, s \vDash \phi_1 \vee \phi_2$ iff $\mathcal{M}, s \vDash \phi_1$ or $\mathcal{M}, s \vDash \phi_2$
6. $\mathcal{M}, s \vDash \phi_1 \rightarrow \phi_2$ iff $\mathcal{M}, s \nvDash \phi_1$ or $\mathcal{M}, s \vDash \phi_2$.
7. $\mathcal{M}, s \vDash \mathrm{AX}\,\phi$ iff for all s_1 such that $s \rightarrow s_1$ we have $\mathcal{M}, s_1 \vDash \phi$. Thus, AX says: 'in every next state.'
8. $\mathcal{M}, s \vDash \mathrm{EX}\,\phi$ iff for some s_1 such that $s \rightarrow s_1$ we have $\mathcal{M}, s_1 \vDash \phi$. Thus, EX says: 'in some next state.' E is dual to A – in exactly the same way that $\exists$ is dual to $\forall$ in predicate logic.
9. $\mathcal{M}, s \vDash \mathrm{AG}\,\phi$ holds iff for all paths $s_1 \rightarrow s_2 \rightarrow s_3 \rightarrow \ldots$, where s_1 equals s, and all s_i along the path, we have $\mathcal{M}, s_i \vDash \phi$. Mnemonically: for All computation paths beginning in s the property ϕ holds Globally. Note that 'along the path' includes the path's initial state s.
10. $\mathcal{M}, s \vDash \mathrm{EG}\,\phi$ holds iff there is a path $s_1 \rightarrow s_2 \rightarrow s_3 \rightarrow \ldots$, where s_1 equals s, and for all s_i along the path, we have $\mathcal{M}, s_i \vDash \phi$. Mnemonically: there Exists a path beginning in s such that ϕ holds Globally along the path.

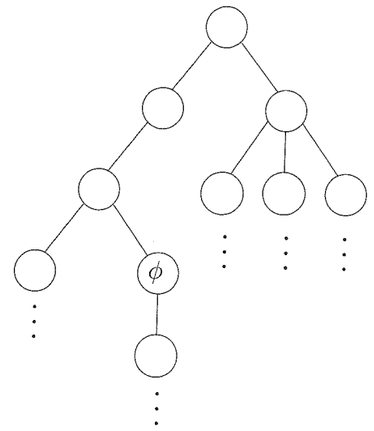

Figure 3.19. A system whose starting state satisfies EF ϕ.

11. $\mathcal{M}, s \vDash \mathrm{AF}\,\phi$ holds iff for all paths $s_1 \to s_2 \to \ldots$, where s_1 equals s, there is some s_i such that $\mathcal{M}, s_i \vDash \phi$. Mnemonically: for All computation paths beginning in s there will be some Future state where ϕ holds.
12. $\mathcal{M}, s \vDash \mathrm{EF}\,\phi$ holds iff there is a path $s_1 \to s_2 \to s_3 \to \ldots$, where s_1 equals s, and for some s_i along the path, we have $\mathcal{M}, s_i \vDash \phi$. Mnemonically: there Exists a computation path beginning in s such that ϕ holds in some Future state;
13. $\mathcal{M}, s \vDash \mathrm{A}[\phi_1\ \mathrm{U}\ \phi_2]$ holds iff for all paths $s_1 \to s_2 \to s_3 \to \ldots$, where s_1 equals s, that path satisfies $\phi_1\ \mathrm{U}\ \phi_2$, i.e., there is some s_i along the path, such that $\mathcal{M}, s_i \vDash \phi_2$, and, for each $j < i$, we have $\mathcal{M}, s_j \vDash \phi_1$. Mnemonically: All computation paths beginning in s satisfy that ϕ_1 Until ϕ_2 holds on it.
14. $\mathcal{M}, s \vDash \mathrm{E}[\phi_1\ \mathrm{U}\ \phi_2]$ holds iff there is a path $s_1 \to s_2 \to s_3 \to \ldots$, where s_1 equals s, and that path satisfies $\phi_1\ \mathrm{U}\ \phi_2$ as specified in 13. Mnemonically: there Exists a computation path beginning in s such that ϕ_1 Until ϕ_2 holds on it.

Clauses 9–14 above refer to computation paths in models. It is therefore useful to visualise all possible computation paths from a given state s by unwinding the transition system to obtain an infinite computation tree, whence 'computation tree logic.' The diagrams in Figures 3.19–3.22 show schematically systems whose starting states satisfy the formulas EF ϕ, EG ϕ, AG ϕ and AF ϕ, respectively. Of course, we could add more ϕ to any of these diagrams and still preserve the satisfaction – although there is nothing to add for AG. The diagrams illustrate a 'least' way of satisfying the formulas.

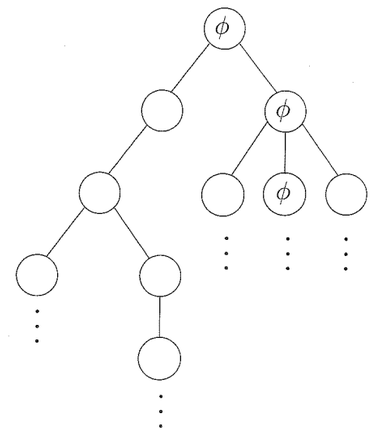

Figure 3.20. A system whose starting state satisfies EG ϕ.

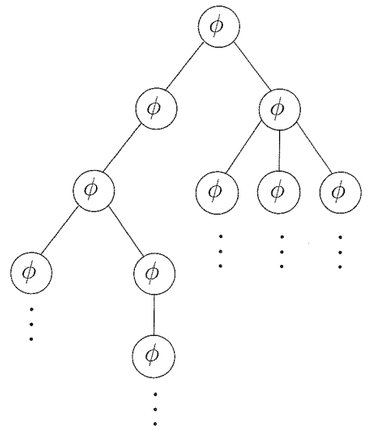

Figure 3.21. A system whose starting state satisfies AG ϕ.

Recall the transition system of Figure 3.3 for the designated starting state s_0, and the infinite tree illustrated in Figure 3.5. Let us now look at some example checks for this system.

1. $\mathcal{M}, s_0 \vDash p \wedge q$ holds since the atomic symbols p and q are contained in the node of s_0.
2. $\mathcal{M}, s_0 \vDash \neg r$ holds since the atomic symbol r is *not* contained in node s_0.

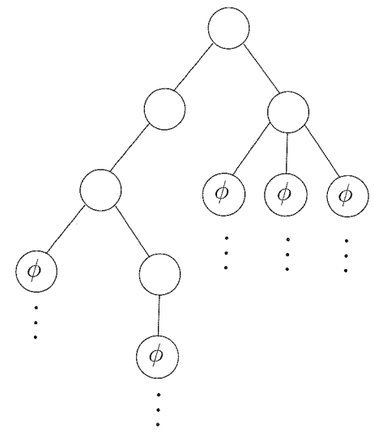

Figure 3.22. A system whose starting state satisfies AF ϕ.

3. $\mathcal{M}, s_0 \models \top$ holds by definition.
4. $\mathcal{M}, s_0 \models \mathrm{EX}\,(q \wedge r)$ holds since we have the leftmost computation path $s_0 \to s_1 \to s_0 \to s_1 \to \ldots$ in Figure 3.5, whose second node s_1 contains q and r.
5. $\mathcal{M}, s_0 \models \neg\mathrm{AX}\,(q \wedge r)$ holds since we have the rightmost computation path $s_0 \to s_2 \to s_2 \to s_2 \to \ldots$ in Figure 3.5, whose second node s_2 only contains r, but *not* q.
6. $\mathcal{M}, s_0 \models \neg\mathrm{EF}\,(p \wedge r)$ holds since there is no computation path beginning in s_0 such that we could reach a state where $p \wedge r$ would hold. This is so because there is simply no state whatsoever in this system where p and r hold at the same time.
7. $\mathcal{M}, s_2 \models \mathrm{EG}\,r$ holds since there is a computation path $s_2 \to s_2 \to s_2 \to \ldots$ beginning in s_2 such that r holds in all future states of that path; this is the only computation path beginning at s_2 and so $\mathcal{M}, s_2 \models \mathrm{AG}\,r$ holds as well.
8. $\mathcal{M}, s_0 \models \mathrm{AF}\,r$ holds since, for all computation paths beginning in s_0, the system reaches a state (s_1 or s_2) such that r holds.
9. $\mathcal{M}, s_0 \models \mathrm{E}[(p \wedge q)\ \mathrm{U}\ r]$ holds since we have the rightmost computation path $s_0 \to s_2 \to s_2 \to s_2 \to \ldots$ in Figure 3.5, whose second node s_2 ($i = 1$) satisfies r, but all previous nodes (only $j = 0$, i.e., node s_0) satisfy $p \wedge q$.
10. $\mathcal{M}, s_0 \models \mathrm{A}[p\ \mathrm{U}\ r]$ holds since p holds at s_0 and r holds in any possible successor state of s_0, so $p\ \mathrm{U}\ r$ is true for all computation paths beginning in s_0 (so we may choose $i = 1$ independently of the path).
11. $\mathcal{M}, s_0 \models \mathrm{AG}\,(p \vee q \vee r \to \mathrm{EF\ EG}\,r)$ holds since in all states reachable from s_0 and satisfying $p \vee q \vee r$ (all states in this case) the system can reach a state satisfying EG r (in this case state s_2).

3.4.3 Practical patterns of specifications

It's useful to look at some typical examples of formulas, and compare the situation with LTL (Section 3.2.3). Suppose atomic descriptions include some words such as busy and requested.

- It is possible to get to a state where started holds, but ready doesn't:
  EF (started $\wedge$ $\neg$ready). To express impossibility, we simply negate the formula.
- For any state, if a request (of some resource) occurs, then it will eventually be acknowledged:
  AG (requested $\to$ AF acknowledged).
- The property that if the process is enabled infinitely often, then it runs infinitely often, is not expressible in CTL. In particular, it is not expressed by AG AF enabled $\to$ AG AF running, or indeed any other insertion of A or E into the corresponding LTL formula. The CTL formula just given expresses that if every path has infinitely often enabled, then every path is infinitely often taken; this is much weaker than asserting that every path which has infinitely often enabled is infinitely often taken.
- A certain process is enabled infinitely often on every computation path:
  AG (AF enabled).
- Whatever happens, a certain process will eventually be permanently deadlocked:
  AF (AG deadlock).
- From any state it is possible to get to a restart state:
  AG (EF restart).
- An upwards travelling lift at the second floor does not change its direction when it has passengers wishing to go to the fifth floor:
  AG (floor2 $\wedge$ directionup $\wedge$ ButtonPressed5 $\to$ A[directionup U floor5])
  Here, our atomic descriptions are boolean expressions built from system variables, e.g., floor2.
- The lift can remain idle on the third floor with its doors closed:
  AG (floor3 $\wedge$ idle $\wedge$ doorclosed $\to$ EG (floor3 $\wedge$ idle $\wedge$ doorclosed)).
- A process can always request to enter its critical section. Recall that this was not expressible in LTL. Using the propositions of Figure 3.8, this may be written AG ($n_1 \to \mathrm{EX}\,t_1$) in CTL.
- Processes need not enter their critical section in strict sequence. This was also not expressible in LTL, though we expressed its negation. CTL allows us to express it directly: EF ($c_1 \wedge$ E[c_1 U ($\neg c_1 \wedge$ E[$\neg c_2$ U c_1])]).

3.4.4 Important equivalences between CTL formulas

Definition 3.16 Two CTL formulas ϕ and ψ are said to be semantically equivalent if any state in any model which satisfies one of them also satisfies the other; we denote this by $\phi \equiv \psi$.