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:

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

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 $\bigwedge$ and $\bigvee$; and after that come $\rightarrow$, 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:

- AG ($q \rightarrow$ EGr), note that this is not the same as AG$q \rightarrow$ EGr, for according to Convention 3.13, the latter formula means (AGq) $\rightarrow$ (EGr)
- EF E[r U q]
- A[p U EF r]
- EF EG$p \rightarrow$ AFr, again, note that this binds as (EF EGp) $\rightarrow$ AFr, not EF (EG$p \rightarrow$ AFr) or EF EG($p \rightarrow$ AF r)
- A[p_1 U A[p_2 U p_3]]
- E[A[p_1 U p_2] U p_3]
- AG($p \rightarrow$ A[p U ($\neg p \bigwedge$ A[$\neg p$ 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:

- EF Gr
- A$\neg$G $\neg_p$
- F [r U q]
- EF (r U q)
- AEF r
- A[(r U q) $\bigwedge$ (p U r)].

It is especially worth understanding why the syntax rules don't allow us to construct these. For example, take EF (r 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 EF E[r U q] or EF A[r U q].

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 A[(r U q) $\bigwedge$ (p U r)] is not a well-formed formula is that the syntax does not allow us to put a boolean connective (like $\bigwedge$) 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 $\bigwedge$, since they are arbitrary formulas; so A[($p \bigwedge q$) 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 ϕ_1 AU ϕ_2, whereas in pure prefix we would write AU(ϕ_1, ϕ_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 A[AX ¬*p* U E[EX (*p* $\wedge$ *q*) U ¬*p*]] is shown in Figure 3.18.

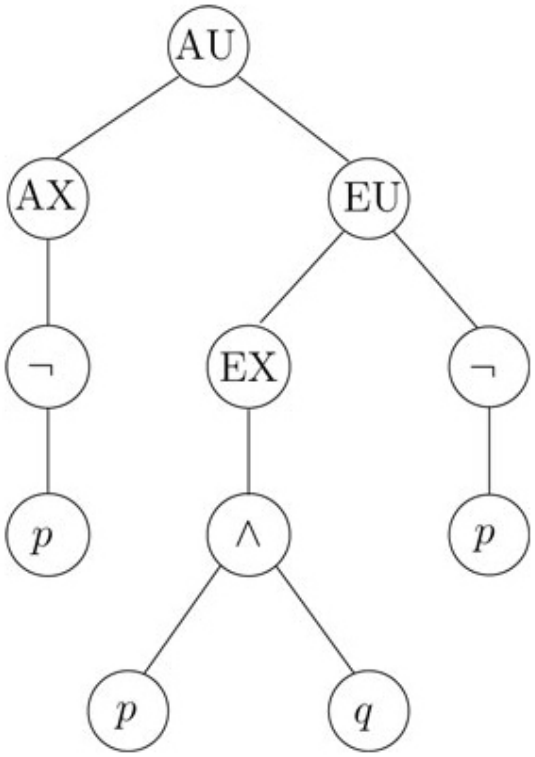


Figure 3.18. The parse tree of a CTL formula without infix notation.

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 $\mathcal{M}, s \models \phi$ 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 $\mathcal{M}, s \models \phi$ 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 $\mathcal{M}, s \models \phi$ is defined by structural induction on ϕ:

1. $\mathcal{M}, s \models \top$ and $\mathcal{M}, s \models \bot$
2. $\mathcal{M}, s \models p$ iff $p \in L(s)$
3. $\mathcal{M}, s \models \neg\phi$ iff $\mathcal{M}, s \models \phi$
4. $\mathcal{M}, s \models \phi_1 \wedge \phi_2$ iff $\mathcal{M}, s \models \phi_1$ and $\mathcal{M}, s \models \phi_2$

5. $\mathcal{M}, s \models \phi_1 \vee \phi_2$ iff $\mathcal{M}, s \models \phi_1$ or $\mathcal{M}, s \models \phi_2$
6. $\mathcal{M}, s \models \phi_1 \rightarrow \phi_2$ iff $\mathcal{M}, s \models \phi_1$ or $\mathcal{M}, s \models \phi_2$.
7. $\mathcal{M}, s \models$ AX ϕ iff for all s_1 such that $s \rightarrow s_1$ we have $\mathcal{M}, s_1 \models \phi$. Thus, AX says: 'in every next state.'
8. $\mathcal{M}, s \models$ EX ϕ iff for some s_1 such that $s \rightarrow s_1$ we have $\mathcal{M}, s_1 \models \phi$. Thus, EX says: 'in some next state.' E is dual to A – in exactly the same way that ∃ is dual to ∀ in predicate logic.
9. $\mathcal{M}, s \models$ AG ϕ 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 $\mathcal{M}, s_i \models \phi$. 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. $\mathcal{M}, s \models$ EG ϕ 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 $\mathcal{M}, s_i \models \phi$. Mnemonically: there Exists a path beginning in s such that ϕ holds Globally along the path.
11. $\mathcal{M}, s \models$ AF ϕ holds iff for all paths $s_1 \rightarrow s_2 \rightarrow \ldots$, where s_1 equals s, there is some s_i such that $\mathcal{M}, s_i \models \phi$. Mnemonically: for All computation paths beginning in s there will be some Future state where ϕ holds.
12. $\mathcal{M}, s \models$ EF ϕ holds iff there is a path $s_1 \rightarrow s_2 \rightarrow s_3 \rightarrow \ldots$, where s_1 equals s, and for some s_i along the path, we have $\mathcal{M}, s_i \models \phi$. Mnemonically: there Exists a computation path beginning in s such that ϕ holds in some Future state;
13. $\mathcal{M}, s \models$ A[ϕ_1 U ϕ_2] holds iff for all paths $s_1 \rightarrow s_2 \rightarrow s_3 \rightarrow \ldots$, where s_1 equals s, that path satisfies ϕ_1 U ϕ_2, i.e., there is some s_i along the path, such that $\mathcal{M}, s_i \models \phi_2$, and, for each $j < i$, we have $\mathcal{M}, s_j \models \phi_1$. Mnemonically: All computation paths beginning in s satisfy that ϕ_1 Until ϕ_2 holds on it.
14. $\mathcal{M}, s \models$ E[ϕ_1 U ϕ_2] holds iff there is a path $s_1 \rightarrow s_2 \rightarrow s_3 \rightarrow \ldots$, where s_1 equals s, and that path satisfies ϕ_1 U ϕ_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.

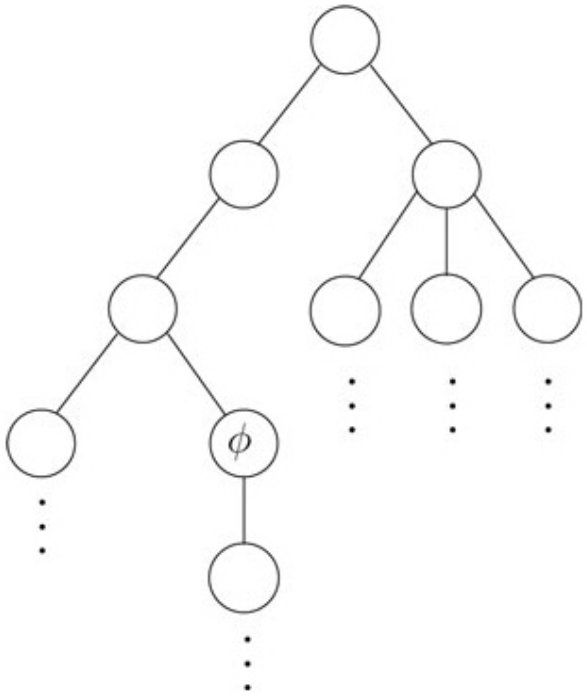


Figure 3.19. A system whose starting state satisfies EF ϕ.

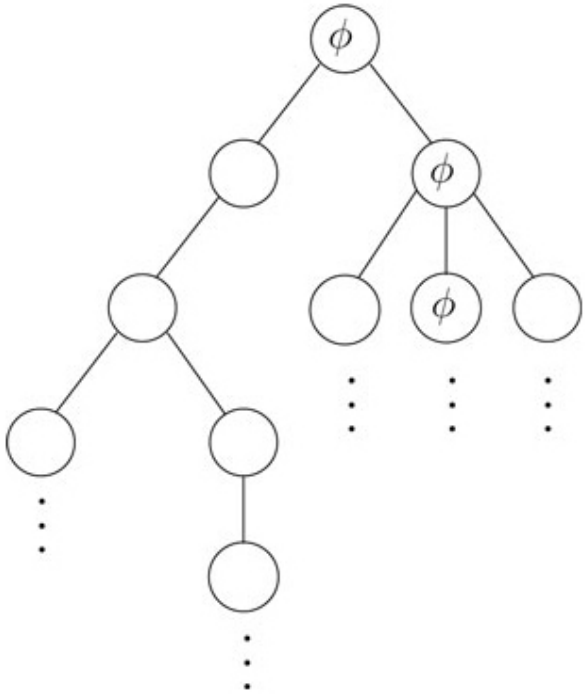


Figure 3.20. A system whose starting state satisfies EG ϕ.

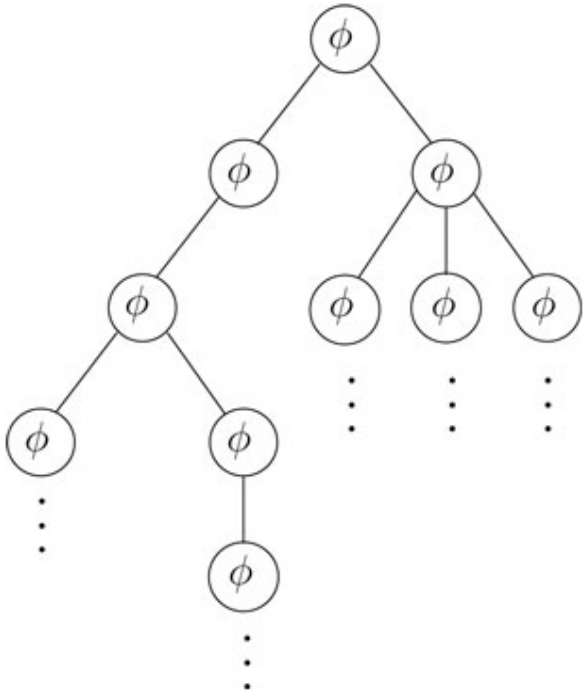


Figure 3.21. A system whose starting state satisfies AG ϕ.

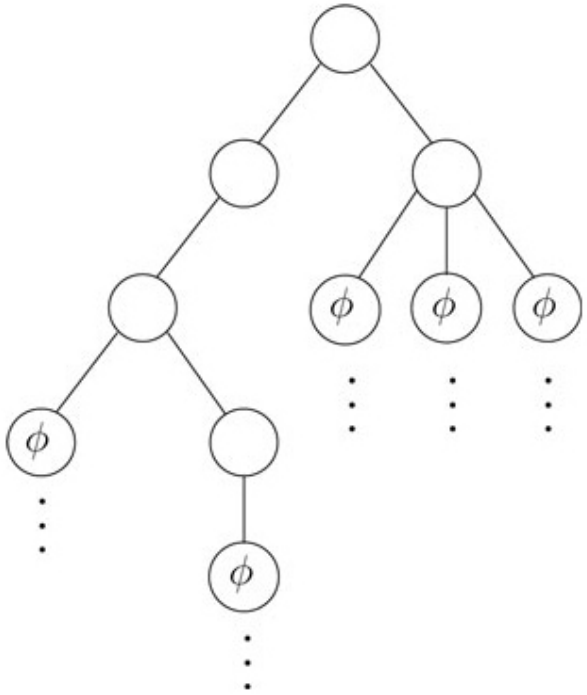


Figure 3.22. A system whose starting state satisfies AF ϕ.

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. $\mathcal{M}, s_0 \models p \wedge q$ holds since the atomic symbols p and q are contained in the node of s_0.
2. $\mathcal{M}, s_0 \models \neg r$ holds since the atomic symbol r is *not* contained in node s_0.
3. $\mathcal{M}, s_0 \models \top$ holds by definition.
4. $\mathcal{M}, s_0 \models$ EX $(q \wedge r)$ holds since we have the leftmost computation path $s_0 \rightarrow s_1 \rightarrow s_0 \rightarrow s_1 \rightarrow \ldots$ in Figure 3.5, whose second node s_1 contains q and r.
5. $\mathcal{M}, s_0 \models \neg$AX $(q \wedge r)$ holds since we have the rightmost computation path $s_0 \rightarrow s_2 \rightarrow s_2 \rightarrow s_2 \rightarrow \ldots$ in Figure 3.5, whose second node s_2 only contains r, but *not* q.
6. $\mathcal{M}, s_0 \models \neg$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. $\mathcal{M}, s_2 \models$ EGr holds since there is a computation path $s_2 \rightarrow s_2 \rightarrow s_2 \rightarrow \ldots$ 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 $\mathcal{M}, s_2 \models$ AGr holds as well.
8. $\mathcal{M}, s_0 \models$ 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. $\mathcal{M}, s_0 \models$ E[$(p \wedge q)$ U r] holds since we have the rightmost computation path $s_0 \rightarrow s_2 \rightarrow s_2 \rightarrow s_2 \rightarrow \ldots$ 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. $\mathcal{M}, s_0 \models$ A[p U r] holds since p holds at s_0 and r holds in any possible successor state of s_0, so p U r is true for all computation paths beginning in s_0 (so we may choose $i = 1$ independently of the path).
11. $\mathcal{M}, s_0 \models$ AG $(p \vee q \vee r \rightarrow$ 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 EGr (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`** $\wedge$ ¬**`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`** → 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`** → 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 **deadlock**ed:
  AF (AG **deadlock**). Note that this formula is stronger than FG **deadlock** considered in section 3.2.3.
- 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** $\bigwedge$ **directionup** $\bigwedge$ **ButtonPressed5** → 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** $\bigwedge$ **idle** $\bigwedge$ **doorclosed** → EG (**floor3** $\bigwedge$ **idle** $\bigwedge$ **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 ($n_1 \rightarrow$ 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 ($c_1 \bigwedge$ E[c_1 U ($\neg c_1 \bigwedge$ 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 $\phi \equiv \psi$.

We have already noticed that A is a universal quantifier on paths and E is the corresponding existential quantifier. Moreover, G and F are also universal and existential quantifiers, ranging over the states along a particular path. In view of these facts, it is not surprising to find that de Morgan rules exist:

$$\begin{aligned} \neg \mathrm{AF}\,\phi &\equiv \mathrm{EG}\,\neg\phi \\ \neg \mathrm{EF}\,\phi &\equiv \mathrm{AG}\,\neg\phi \\ \neg \mathrm{AX}\,\phi &\equiv \mathrm{EX}\,\neg\phi. \end{aligned} \tag{3.6}$$

We also have the equivalences

$$\mathrm{AF}\,\phi \;\equiv\; \mathrm{A}[\top\ \mathrm{U}\ \phi] \qquad \mathrm{EF}\,\phi \;\equiv\; \mathrm{E}[\top\ \mathrm{U}\ \phi]$$

which are similar to the corresponding equivalences in LTL.

3.4.5 Adequate sets of CTL connectives

As in propositional logic and in LTL, there is some redundancy among the CTL connectives. For example, the connective AX can be written ¬EX ¬; and AG, AF, EG and EF can be written in terms of AU and EU as follows: first, write AG ϕ as ¬EF ¬ϕ and EG ϕ as ¬AF ¬ϕ, using (3.6), and then use AF $\phi \equiv$ A[$\top$ U ϕ] and EF $\phi \equiv$ E[$\top$ U ϕ]. Therefore AU, EU and EX form an adequate set of temporal connectives.

Also EG, EU, and EX form an adequate set, for we have the equivalence

$$\mathrm{A}[\phi\ \mathrm{U}\ \psi] \;\equiv\; \neg(\mathrm{E}[\neg\psi\ \mathrm{U}\ (\neg\phi \wedge \neg\psi)] \vee \mathrm{EG}\,\neg\psi) \tag{3.7}$$

which can be proved as follows:

$$\begin{aligned} \mathrm{A}[\phi\ \mathrm{U}\ \psi] &\equiv \mathrm{A}[\neg(\neg\psi\ \mathrm{U}\ (\neg\phi \wedge \neg\psi)) \wedge \mathrm{F}\,\psi] \\ &\equiv \neg\mathrm{E}\neg[\neg(\neg\psi\ \mathrm{U}\ (\neg\phi \wedge \neg\psi)) \wedge \mathrm{F}\,\psi] \\ &\equiv \neg\mathrm{E}[(\neg\psi\ \mathrm{U}\ (\neg\phi \wedge \neg\psi)) \vee \mathrm{G}\,\neg\psi] \\ &\equiv \neg(\mathrm{E}[\neg\psi\ \mathrm{U}\ (\neg\phi \wedge \neg\psi)] \vee \mathrm{EG}\,\neg\psi). \end{aligned}$$

The first line is by Theorem 3.10, and the remainder by elementary manipulation. (This proof involves intermediate formulas which violate the syntactic formation rules of CTL; however, it is valid in the logic CTL* introduced in the next section.) More generally, we have:

Theorem 3.17 A set of temporal connectives in CTL is adequate if, and only if, it contains at least one of {AX , EX }, at least one of {EG , AF , AU } and EU.

This theorem is proved in a paper referenced in the bibliographic notes at the end of the chapter. The connective EU plays a special role in that theorem because neither weak-until W nor release R are primitive in CTL (Definition 3.12). The temporal connectives AR, ER, AW and EW are all definable in CTL:

- A[ϕ R ψ] = ¬E[¬ϕ U ¬ψ]
- E[ϕ R ψ] = ¬A[¬ϕ U ¬ψ]
- A[ϕ W ψ] = A[ψ R ($\phi \vee \psi$)], and then use the first equation above
- E[ϕ W ψ] = E[ψ R ($\phi \vee \psi$)], and then use the second one.

These definitions are justified by LTL equivalences in Sections 3.2.4 and 3.2.5. Some other noteworthy equivalences in CTL are the following:

$$\begin{aligned}
\mathrm{AG}\,\phi &\equiv \phi \wedge \mathrm{AX}\,\mathrm{AG}\,\phi \\
\mathrm{EG}\,\phi &\equiv \phi \wedge \mathrm{EX}\,\mathrm{EG}\,\phi \\
\mathrm{AF}\,\phi &\equiv \phi \vee \mathrm{AX}\,\mathrm{AF}\,\phi \\
\mathrm{EF}\,\phi &\equiv \phi \vee \mathrm{EX}\,\mathrm{EF}\,\phi \\
\mathrm{A}[\phi\ \mathrm{U}\ \psi] &\equiv \psi \vee (\phi \wedge \mathrm{AX}\,\mathrm{A}[\phi\ \mathrm{U}\ \psi]) \\
\mathrm{E}[\phi\ \mathrm{U}\ \psi] &\equiv \psi \vee (\phi \wedge \mathrm{EX}\,\mathrm{E}[\phi\ \mathrm{U}\ \psi]).
\end{aligned}$$

For example, the intuition for the third one is the following: in order to have AF ϕ in a particular state, ϕ must be true at some point along each path from that state. To achieve this, we either have ϕ true now, in the current state; or we postpone it, in which case we must have AF ϕ in each of the next states. Notice how this equivalence appears to define AF in terms of AX and AF itself, an apparently circular definition. In fact, these equivalences can be used to define the six connectives on the left in terms of AX and EX , in a *non-circular* way. This is called the fixed-point characterisation of CTL; it is the mathematical foundation for the model-checking algorithm developed in Section 3.6.1; and we return to it later (Section 3.7).

3.5 CTL* and the expressive powers of LTL and CTL

CTL allows explicit quantification over paths, and in this respect it is more expressive than LTL, as we have seen. However, it does not allow one to select a range of paths by describing them with a formula, as LTL does. In that respect, LTL is more expressive. For example, in LTL we can say 'all paths which have a p along them also have a q along them,' by writing F $p \rightarrow$ F q. It is not possible to write this in CTL because of the constraint that every F has an associated A or E. The formula AF $p \rightarrow$ AF q means something quite different: it says 'if all paths have a p along them, then all paths have a q along them.' One might write AG ($p \rightarrow$ AF q), which is closer, since it says that every way of extending every path to a p eventually meets a q, but that is still not capturing the meaning of F $p \rightarrow$ F q.

CTL* is a logic which combines the expressive powers of LTL and CTL, by dropping the CTL constraint that every temporal operator (X, U, F, G) has to be associated with a unique path quantifier (A, E). It allows us to write formulas such as

- A[(p U r) $\vee$ (q U r)]: along all paths, either p is true until r, or q is true until r.
- A[$X\,p \vee$ X X p]: along all paths, p is true in the next state, or the next but one.
- E[G F p]: there is a path along which p is infinitely often true.

These formulas are *not* equivalent to, respectively, A[($p \vee q$) U r)], AX$p \vee$ AX AXp and EG EFp. It turns out that the first of them can be written as a (rather long) CTL formula. The second and third do not have a CTL equivalent.

The syntax of CTL* involves two classes of formulas:

- *state formulas*, which are evaluated in states:

$$\phi ::= \top \mid p \mid (\neg\phi) \mid (\phi \wedge \phi) \mid \mathrm{A}[\alpha] \mid \mathrm{E}[\alpha]$$

where p is any atomic formula and α any path formula; and

- *path formulas*, which are evaluated along paths:

$$\alpha ::= \phi \mid (\neg\alpha) \mid (\alpha \wedge \alpha) \mid (\alpha\ \mathrm{U}\ \alpha) \mid (\mathrm{G}\,\alpha) \mid (\mathrm{F}\,\alpha) \mid (\mathrm{X}\,\alpha)$$

where ϕ is any state formula. This is an example of an inductive definition which is *mutually recursive*: the definition of each class depends upon the definition of the other, with base cases p and $\top$.

LTL and CTL as subsets of CTL* Although the syntax of LTL does not include A and E, the semantic viewpoint of LTL is that we consider all paths. Therefore, the LTL formula α is equivalent to the CTL* formula A[α]. Thus, LTL can be viewed as a subset of CTL*.

CTL is also a subset of CTL*, since it is the fragment of CTL* in which we restrict the form of path formulas to

$$\alpha ::= (\phi \,\mathrm{U}\, \phi) \mid (\mathrm{G}\, \phi) \mid (\mathrm{F}\, \phi) \mid (\mathrm{X}\, \phi)$$

Figure 3.23 shows the relationship among the expressive powers of CTL, LTL and CTL*. Here are some examples of formulas in each of the subsets shown:

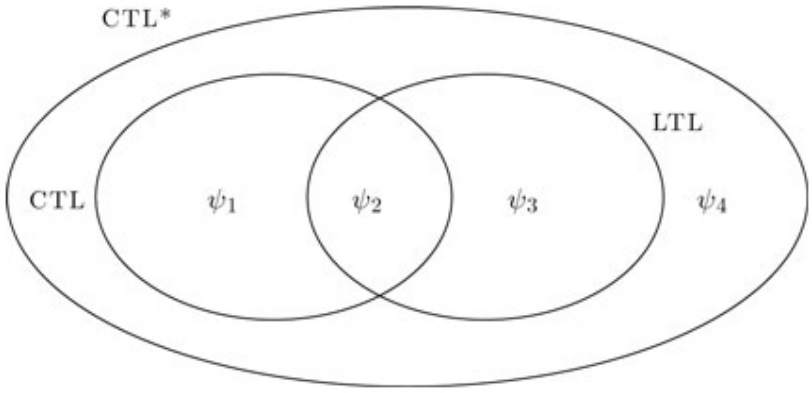
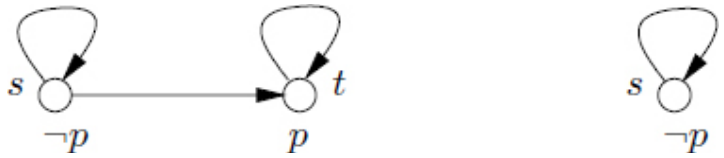


Figure 3.23. The expressive powers of CTL, LTL and CTL*.

In CTL but not in LTL: $\psi_1 \stackrel{\text{def}}{=} \mathrm{AG\,EF}\, p$. This expresses: wherever we have got to, we can always get to a state in which p is true. This is also useful, e.g., in finding deadlocks in protocols.

The proof that AG EF p is not expressible in LTL is as follows. Let ϕ be an LTL formula such that A[ϕ] is allegedly equivalent to AG EF p. Since $\mathcal{M}, s \models$ AG EF p in the left-hand diagram below, we have $\mathcal{M}, s \models$ A[ϕ]. Now let $\mathcal{M}'$ be as shown in the right-hand diagram. The paths from s in $\mathcal{M}'$ are a subset of those from s in $\mathcal{M}$, so we have $\mathcal{M}', s \models$ A[ϕ]. Yet, it is *not* the case that $\mathcal{M}', s \models$ AG EF p; a contradiction.



In CTL*, but neither in CTL nor in LTL: $\psi_4 \stackrel{\text{def}}{=}$ E[G F p], saying that there is a path with infinitely many p.

The proof that this is not expressible in CTL is quite complex and may be found in the papers co-authored by E. A. Emerson with others, given in the references. (Why is it not expressible in LTL?)

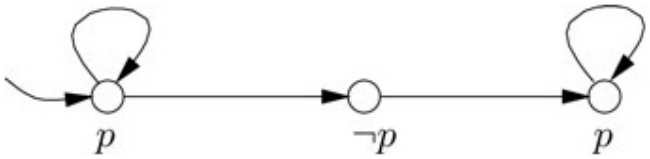
In LTL but not in CTL: $\psi_3 \stackrel{\text{def}}{=} \mathrm{A[G\,F}\, p \rightarrow \mathrm{F}\, q]$, saying that if there are infinitely many p along the path, then there is an occurrence of q.

This is an interesting thing to be able to say; for example, many fairness constraints are of the form 'infinitely often requested implies eventually acknowledged'.

In LTL and CTL: $\psi_2 \stackrel{\text{def}}{=}$ AG ($p \rightarrow$ AF q) in CTL, or G ($p \rightarrow$ F q) in LTL: any p is eventually followed by a q.

Remark 3.18 We just saw that some (but not all) LTL formulas can be converted into CTL formulas by adding an A to each temporal operator. For a positive example, the LTL formula G ($p \rightarrow$ F q) is equivalent to the CTL formula AG ($p \rightarrow$ AF q). We discuss two more negative examples:

- F G p and AF AG p are not equivalent, since F G p is satisfied, whereas AF AG p is not satisfied, in the model



In fact, AF AG p is strictly stronger than F G p.

- While the LTL formulas X F p and F X p are equivalent, and they are equivalent to the CTL formula AX AF p, they are not equivalent to AF AX p. The latter is strictly stronger, and has quite a strange meaning (try working it out).

Remark 3.19 There is a considerable literature comparing linear-time and branching-time logics. The question of which one is 'better' has been debated for about 20 years. We have seen that they have incomparable expressive powers. CTL* is more expressive than either of them, but is computationally much more expensive (as will be seen in Section 3.6). The choice between LTL and CTL depends on the application at hand, and on personal preference. LTL lacks CTL's ability to quantify over paths, and CTL lacks LTL's finer-grained ability to describe individual paths. To many people, LTL appears to be more straightforward to use; as noted above, CTL formulas like AF AX p seem hard to understand.

3.5.1 Boolean combinations of temporal formulas in CTL

Compared with CTL*, the syntax of CTL is restricted in two ways: it does not allow boolean combinations of path formulas and it does not allow nesting of the path modalities X, F and G. Indeed, we have already seen examples of the inexpressibility in CTL of nesting of path modalities, namely the formulas ψ_3 and ψ_4 above.

In this section, we see that the first of these restrictions is only apparent; we can find equivalents in CTL for formulas having boolean combinations of path formulas. The idea is to translate any CTL formula having boolean combinations of path formulas into a CTL formula that doesn't. For example, we may see that E[F $p \wedge$ F q] ≡ EF [$p \wedge$ EF q] $\vee$ EF [$q \wedge$ EF p] since, if we have F $p \wedge$ F q along any path, then either the p must come before the q, or the other way around, corresponding to the two disjuncts on the right. (If the p and q occur simultaneously, then both disjuncts are true.)

Since U is like F (only with the extra complication of its first argument), we find the following equivalence:

$$\begin{aligned}\mathrm{E}[(p_1 \ \mathrm{U}\ q_1) \wedge (p_2 \ \mathrm{U}\ q_2)] \equiv \mathrm{E}[(p_1 \wedge p_2) \ \mathrm{U}\ (q_1 \wedge \mathrm{E}[p_2 \ \mathrm{U}\ q_2])] \\ \vee \mathrm{E}[(p_1 \wedge p_2) \ \mathrm{U}\ (q_2 \wedge \mathrm{E}[p_1 \ \mathrm{U}\ q_1])].\end{aligned}$$

And from the CTL equivalence A[p U q] ≡ ¬(E[¬q U (¬$p \wedge$ ¬q)] $\vee$ EG ¬q) (see Theorem 3.10) we can obtain E[¬(p U q)] ≡ E[¬q U (¬$p \wedge$ ¬q)] $\vee$ EG ¬q. Other identities we need in this translation include E[¬X p] ≡ EX ¬p.

3.5.2 Past operators in LTL

The temporal operators X, U, F, etc. which we have seen so far refer to the future. Sometimes we want to encode properties that refer to the past, such as: 'whenever q occurs, then there was some p in the past.' To do this, we may add the operators Y, S, O, H. They stand for *yesterday*, *since*, *once*, and *historically*, and are the past analogues of X, U, F, G, respectively. Thus, the example formula may be written G ($q \rightarrow O\ p$).

NuSMV supports past operators in LTL. One could also add past operators to CTL (AY, ES, etc.) but NuSMV does not support them.

Somewhat counter-intuitively, past operators do not increase the expressive power of LTL. That is to say, every LTL formula with past operators can be written equivalently without them. The example formula above can be written ¬q W p, or equivalently ¬(¬p U ($q \wedge$ ¬p)) if one wants to avoid W. This result is surprising, because it seems that being able to talk about the past as well as the future allows more expressivity than talking about the future alone. However, recall that LTL equivalence is quite crude: it says that the two formulas are satisfied by exactly the same set of paths. The past operators allow us to travel backwards along the path, but only to reach points we could have reached by travelling forwards from its beginning. In contrast, adding past operators to CTL does increase its expressive power, because they can allow us to examine states not forward-reachable from the present one.