

Chapter 14

Finite state machines and model checking

14.1 Finite state machines

Suppose we have a system that is similar to a discrete-time Markov chain: at any (discrete) instant in time n , the system is in a particular state $i \in \mathcal{S}$. As with a DTMC, the state at time $n + 1$ is based only on the state at time n . However, instead of making this decision based on probability, it is made based on a non-deterministic choice (made by a user, or some other entity beyond our control). We call this a *finite state machine* (FSM), as we assume the set \mathcal{S} is finite.

Like Markov chains, we can draw a FSM as a directed graph. The graph edges are unweighted, or can be weighted with the name of an action (in case we want to distinguish between decisions). For now, we will consider unweighted edges. As such, a FSM can be represented by a boolean matrix \mathbf{E} , where

$$\mathbf{E}[i, j] = \begin{cases} 1 & \text{if the system can go to state } j \text{ after state } i \\ 0 & \text{otherwise} \end{cases}$$

We assume that every state has at least one outgoing arc, possibly only to itself. This means that every execution path is infinitely long, which simplifies later discussions about CTL model checking. The set of possible behaviors of a FSM is *more general* than a Markov chain, because the non-deterministic choices are not bound by any rules (such as the laws of probability).

Example 14.1

The following FSM could be used to describe a (simple) CD player:



The meaning of each state is as follows:

E : The CD player is empty (and closed)

O : The CD player is open

S : The CD player contains a CD, and is stopped

P : The CD player is playing a CD

The transitions between states correspond to user actions (e.g., the user presses “play”).

In the above finite state machine, note that:

1. It is possible to go from state O to state P (because there is a path from state O to state P).
2. It is possible to start in state O and *never reach* state P (because there is an infinite execution sequence, starting in O , that does not include P).

If the above model were instead a DTMC, with all the shown edges having non-zero probability, then (1) holds but (2) does not: the DTMC *will* eventually reach state P starting in state O (to see this, make state P absorbing. . .).

Since a finite state machine does not have probabilities or rates on the edges, we cannot draw any meaningful conclusions about the probability of reaching a particular state, or the expected time to reach a particular state. However, we can perform two simple operations that form the core of CTL model checkers.

Post-image computation

Given a set of states $\mathcal{X} \subseteq \mathcal{S}$, what states can the finite state machine be in, in the next “step” (i.e., after one transition occurs)? This question can be answered using an appropriate data structure for \mathcal{X} , and performing a graph search over the edges \mathbf{E} . A simple way to do this, consistent with our previous discussions about Markov chains, is to use a boolean vector \mathbf{x} of dimension $|\mathcal{S}|$ to represent \mathcal{X} :

$$\mathbf{x}[i] = 1 \quad \Leftrightarrow \quad i \in \mathcal{X}$$

The set of states \mathcal{Y} that can be reached in one step from \mathcal{X} , is also represented as a boolean vector \mathbf{y} . This set can be constructed using the operation

$$\mathbf{y} = \mathbf{x} \cdot \mathbf{E}$$

where scalar addition and multiplication correspond to “logical” boolean operations (i.e., $1+1 = 1$).

Example 14.2

For the simple CD-player FSM, if the FSM is currently in either state O or P , what states can the FSM be in after one transition?

Using the vector-matrix multiplication operation, with states in the order $\{E, O, S, P\}$, we have

$$\begin{aligned} \mathbf{y} &= [0, 1, 0, 1] \cdot \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \\ &= [1, 0, 1, 0] + [0, 0, 1, 0] \\ &= [1, 0, 1, 0] \end{aligned}$$

Therefore, the FSM can be in state E or S .

Pre-image computation

Given a set of states $\mathcal{X} \subseteq \mathcal{S}$, what states could the finite state have been in, in the previous step? We could compute this by reversing the edges in the graph, and then performing a pre-image computation. However, this can be reduced to

$$\mathbf{y} = \mathbf{x} \cdot \mathbf{E}^T = \mathbf{E} \cdot \mathbf{x}$$

to avoid the transpose operation.

Example 14.3

If the CD-player FSM is currently in either state O or P , what states could it have been in previously?

Using the matrix-vector multiplication operation, we have

$$\begin{aligned} \mathbf{y} &= \begin{bmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} 0 \\ 1 \\ 0 \\ 1 \end{bmatrix} \\ &= \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} \\ &= \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \end{bmatrix} \end{aligned}$$

Thus, the FSM could have been in state E or S .

14.2 Computation Tree Logic

To ask (and answer) more interesting questions, we need a formal way to ask these questions. This means we need to define formal operators, and give rules for how to combine the operators. These formal rules will allow us to express interesting (and complex) questions as logical formulas without ambiguity. In this section, we will cover the operators and basic concepts that are used by Computation Tree Logic, or CTL.

14.2.1 State formulas

Definition 14.1 *A state formula is one that either holds, or fails to hold, for a particular state. If a formula f holds in state s , we say s satisfies f , written $s \models f$; otherwise we say s does not satisfy f , written $s \not\models f$.*

State formulas include a special type of formula called “atomic propositions”, which are given or are very easy to determine from the FSM. In addition, state formulas may be combined using the usual boolean logic operators \wedge, \vee, \neg :

$$\begin{aligned} s \models f \wedge g & \text{ iff } (s \models f) \wedge (s \models g) \\ s \models f \vee g & \text{ iff } (s \models f) \vee (s \models g) \\ s \models \neg f & \text{ iff } (s \not\models f) \end{aligned}$$

In particular, we will use the special atomic proposition *true* to denote a formula which all states satisfy.

Example 14.4

For the CD-player FSM, let the atomic proposition *f* equal “the CD player is closed”, and let the atomic proposition *g* equal “the CD player contains a CD”. Then we have

$$\begin{aligned} E & \models f \\ E & \not\models g \\ E & \models f \vee g \\ E & \not\models f \wedge g \\ E & \models f \wedge \neg g \end{aligned}$$

14.2.2 Path formulas

Definition 14.2 A path formula is one that either holds, or fails to hold for a path through the FSM.

A path through the FSM is a (usually, infinite) sequence of states such that pairs of successor states in the sequence are connected by an edge in the FSM. The same state may appear several times in a path.

Example 14.5

One possible path through the CD-player FSM is

$$E \rightarrow O \rightarrow S \rightarrow P \rightarrow S \rightarrow P \rightarrow S \rightarrow P \rightarrow S \rightarrow O \rightarrow \dots$$

14.2.3 Temporal operators

Temporal operators are used to express properties about paths. Important operators used in CTL are the following.

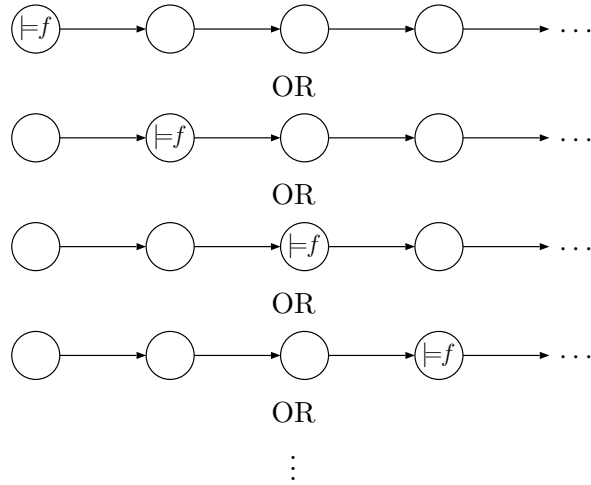
X operator (neXt state)

Xf is true for a given path if the second state in the path satisfies *f*:



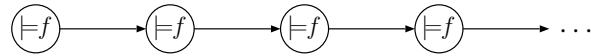
F operator (Future)

Ff is true for a given path if some state in the path satisfies f :



G operator (Globally)

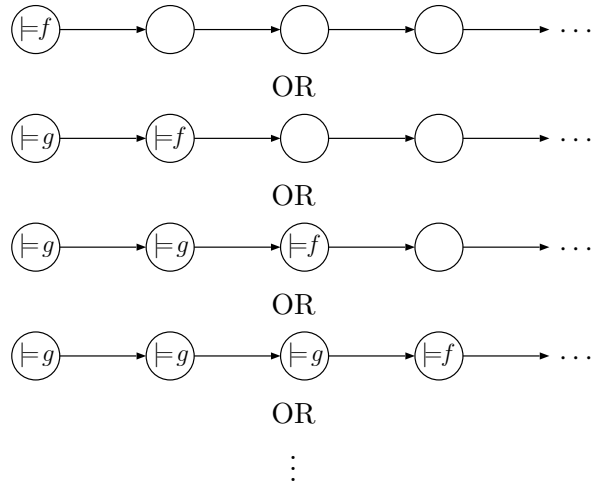
Gf is true for a given path if every state in the path satisfies f :



Note $Gf \equiv \neg F\neg f$

U operator (Until)

gUf is true for a given path if some state in the path satisfies f , and every state before that satisfies g :



Note $Ff \equiv true U f$.

$U^{\leq t}$ operator (Bounded until)

$gU^{\leq t}f$ is similar to gUf , except the number of states satisfying g , before reaching a state that satisfies f , is at most t .

R operator (Release)

The R operator is the logical dual of U:

$$g R f \equiv \neg(\neg g U \neg f)$$

For completeness, we can also define the *time reversed* versions of the above operators, where the above pictures would have the arcs reversed. Thus, for the “ordinary” operators, paths are infinitely-long “to the right”: each path has a starting state, but no final state. Conversely, the time-reversed operators use paths that are infinitely-long “to the left”: each path has a final state, but no starting state. The time reversed operators are:

\bar{X} or Y for time-reversed X

\bar{F} or P (Past) for time-reversed F

\bar{G} or H for time-reversed G

\bar{U} or S (Since) for time-reversed U

Since the time reversed operators are equivalent to reversing the arcs in the finite state machine (i.e., transposing the edge matrix **E**) and then applying the corresponding forward-time operator, we will not discuss these operations any further.

14.2.4 Path quantifiers

We can specify that a path formula holds *for all* paths or *for at least one* path. These are specified by path quantifiers:

A operator: for all paths

E operator: for at least one path (i.e., “there exists a path”)

Note $A(p) \equiv \neg E(\neg p)$

14.2.5 CTL rules: putting it all together

Different temporal logics have different rules about how the above operators may be combined in a formula. In CTL, the rules can be stated as follows:

1. Atomic propositions are state formulas.
2. If f and g are state formulas, then so are $f \wedge g$, $f \vee g$, and $\neg f$.
3. If f and g are state formulas, then Xf , Ff , Gf , and gUf are path formulas.
4. If p is a “forward-time” path formula, then Ap and Ep are state formulas, where Ap holds for a state if p holds for all paths starting in that state. Similarly, Ep holds for a state if there exists a path where p holds, starting in that state.
5. A CTL property is expressed as a state formula, and it is true for a FSM if the state formula holds for (one of) the initial state(s).

Note that these rules imply that nesting of the operators X, F, G, U, A, E, is allowed, but only in the pairs AX, EX, AF, EF, AG, EG, AU, EU.

14.3 Limits of CTL

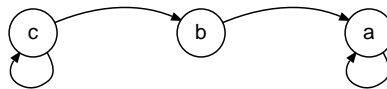
Let p be an atomic proposition. There is no way to express the property $A(\text{FG } p)$ in CTL. This is a legal expression in other temporal logics (in particular, linear temporal logic, or LTL). The obvious property to try in CTL is $\text{AF}(\text{AG } p)$, but this is *stronger than* $A(\text{FG } p)$:

$A(\text{FG } p)$ means: for all paths, in the future a state is reached from which p holds globally.

$\text{AF}(\text{AG } p)$ means: for all paths, in the future a state is reached from which, for all paths leaving from that state, p holds globally.

Example 14.6

Suppose we have the following finite state machine with initial state c :



and suppose we have an atomic proposition p where a and c satisfy p , but b does not. This model is simple enough that we can imagine all possible paths through the state machine. Every path visits state c one or more times, then visits state b , then remains in state a . However, it is also possible to never leave state c . If we remain in state c , then property p always holds, and for this path $\text{FG } p$ holds. Otherwise, we eventually (in the Future) reach state a , and then property p always holds. Thus, for this finite state machine, $A(\text{FG } p)$ holds.

Now consider the CTL property $\text{AF}(\text{AG } p)$. Since $\text{AG } p$ is a state formula, we can determine which states satisfy it. Clearly, $\text{AG } p$ holds only for state a : only from state a is it true that on all paths, p holds globally. Thus, for this finite state machine, the property $\text{AF}(\text{AG } p)$ is equivalent to the state formula $\text{AF } a$. But which states satisfy this formula? Again, a satisfies the formula, as does b (because there is only one path from b , and that is to state a). But c does not, because it is possible to remain in state c forever. Thus, not *all* paths from state c can reach state a . As such, for this finite state machine, $\text{AF}(\text{AG } p)$ does not hold.

14.4 CTL model checking

The model checking problem is as follows: given a finite state machine (as an edge matrix \mathbf{E}) and the initial state (or states), and a CTL formula f , does the formula hold for the initial state? As the previous example suggests, a complex CTL formula can be computed recursively, by determining which states satisfy each sub-expression (which is a state formula). As such, to perform CTL model checking, we need algorithms to implement each of the state formula operators, namely \wedge, \vee, \neg and the operator pairs EX, AX, EF, AF, EG, AG, EU, AU.

14.4.1 The logical operators

Suppose we have evaluated state formulas f and g , and have stored the sets of states that satisfy these formulas (at least, conceptually) as boolean vectors \mathbf{f} and \mathbf{g} . Then we need to implement the operations $f \wedge g$, $f \vee g$, and $\neg f$. In other words, we need to be able to construct a boolean vector \mathbf{h} of states satisfying the above formulas. These operations can be described in terms of element-wise logical operations on the vectors \mathbf{f} and \mathbf{g} :

$f \wedge g$: We can compute the vector for $f \wedge g$ by

$$\mathbf{h}[i] = \mathbf{f}[i] \wedge \mathbf{g}[i] \quad \forall i$$

$f \vee g$: We can compute the vector for $f \vee g$ by

$$\mathbf{h}[i] = \mathbf{f}[i] \vee \mathbf{g}[i] \quad \forall i$$

which is equivalent to $\mathbf{f} + \mathbf{g}$ (for “logical” $+$).

$\neg f$: We can compute the vector for $\neg f$ by

$$\mathbf{h}[i] = \neg \mathbf{f}[i] \quad \forall i$$

14.4.2 The operator pairs

Similar to the implementation for logical operators, we must be able to construct a boolean vector \mathbf{h} of states satisfying the formulas $AX f$, $EX f$, $AF f$, $EF f$, $AG f$, $EG f$, $Ag U f$, and $Eg U f$. However, some of these operations can be expressed in terms of other operations. In fact, we can express *all* of the above operations in terms of only the three operations $EX f$, $Eg U f$, and $EG f$:

$$\begin{aligned} AX f &= \neg EX \neg f \\ AF f &= \neg EG \neg f \\ EF f &= E \text{ true } U f \\ AG f &= \neg EF \neg f = \neg E \text{ true } U \neg f \\ Ag U f &= \neg E[\neg f U (\neg f \wedge \neg g)] \wedge \neg EG \neg f \end{aligned}$$

Since we already know how to combine state formulas using \neg and \wedge , we only need to determine how to implement the three operators EX , EU , and EG .

EX

The set of states satisfying $EX f$ is exactly the set of states that can reach a state satisfying f in one step. Thus, given the states satisfying f , we can perform the pre-image computation to obtain the states satisfying $EX f$:

$$\mathbf{h} = \mathbf{E} \cdot \mathbf{f}$$

where \mathbf{E} is the matrix of edges of the finite state machine.

EU

We can build the set of states satisfying $Eg \cup f$ by starting with states satisfying f , and then adding any state that satisfies g and can reach any state that satisfies the property. Thus, we can use the following iteration.

$$\begin{aligned}\mathbf{h}_0 &= \mathbf{f} \\ \mathbf{h}_{n+1} &= \mathbf{h}_n \vee ((\mathbf{E} \cdot \mathbf{h}_n) \wedge \mathbf{g})\end{aligned}$$

Note that $\mathbf{E} \cdot \mathbf{h}_n$ gives the set of states that can reach \mathbf{h}_n in one step, and then we intersect¹ with \mathbf{g} . Therefore, \mathbf{h}_n represents the set of states from which there is a path of n or fewer states satisfying g , and then a state satisfying f . Clearly, we have $\mathbf{h}_n \subseteq \mathbf{h}_{n+1}$ for all n , and since the set \mathcal{S} is finite, we will eventually have $\mathbf{h}_{m+1} = \mathbf{h}_m$ for some m . When this occurs, we may stop the iterations, and the result is exactly the set satisfying $Eg \cup f$.

To instead build the set of states satisfying $Eg \cup^{\leq t} f$, we can use the above iterations and stop once we obtain \mathbf{h}_t , since this represents the set of states from which there is a path of t or fewer states satisfying g , followed by a state satisfying f .

EG

We can build the set of states satisfying EGf iteratively, by starting with the set satisfying f , and at each iteration, looking at all the states that can reach states satisfying the property, and removing those that do not satisfy f . Thus, we can use the following iteration².

$$\begin{aligned}\mathbf{h}_0 &= \mathbf{f} \\ \mathbf{h}_{n+1} &= (\mathbf{E} \cdot \mathbf{h}_n) \wedge \mathbf{f}\end{aligned}$$

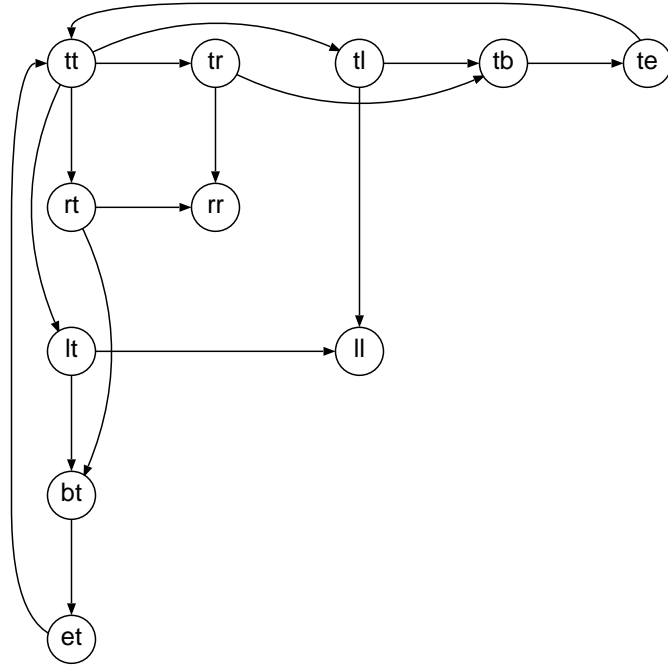
Note that \mathbf{h}_{n+1} represents the set of states satisfying f , that can reach a state satisfying \mathbf{h}_n . Thus, \mathbf{h}_n represents the set of states from which there is a path of length $n + 1$ states all satisfying f . Therefore, we have $\mathbf{h}_{n+1} \subseteq \mathbf{h}_n$ for all n , and we will eventually have $\mathbf{h}_{m+1} = \mathbf{h}_m$ for some m . When this occurs, we may stop the iterations, and the result is exactly the set satisfying EGf .

14.4.3 Examples

Consider the following finite state machine model of two dining philosophers:

¹The matrix–vector multiplication operation can be combined with the set intersection operation for efficiency.

²Again, we can combine the vector–matrix multiplication operation with the set intersection operation.



The state of each philosopher is either “thinking” (has neither fork), “has only the left fork”, “has only the right fork”, “has both forks”, and “eating”. Note there are two absorbing states: rr , where each philosopher has their right fork (and so no forks remain on the table), and ll , where each philosopher has their left fork. For verifying CTL properties (below), we will assume that the absorbing states have self-loops.

Example 14.7

For the two dining philosophers model, determine which states satisfy $\text{EF}(\text{Phil.1 is eating})$.

First, we must convert the expression into one of the three supported operator pairs; this gives us:

$$\text{EF}(\text{Phil.1 is eating}) = \text{E}(\text{true} \cup \text{Phil.1 is eating})$$

Next, we determine the set of states that satisfy the atomic proposition “Phil.1 is eating”, and obtain $\{et\}$. Note that the set of states satisfying the atomic proposition true is the entire state space, \mathcal{S} . To compute the set satisfying $\text{E}(\mathcal{S} \cup \{et\})$, we can use the iterative algorithm for EU:

$$\begin{aligned} \mathbf{h}_0 &= \{et\} \\ \mathbf{h}_1 &= \mathbf{h}_0 \cup ((\text{Set of states reaching } \mathbf{h}_0) \cap \mathcal{S}) = \{bt, et\} \\ \mathbf{h}_2 &= \mathbf{h}_1 \cup ((\text{Set of states reaching } \mathbf{h}_1) \cap \mathcal{S}) = \{rt, lt, bt, et\} \\ \mathbf{h}_3 &= \dots = \{tt, rt, lt, bt, et\} \\ \mathbf{h}_4 &= \dots = \{tt, te, rt, lt, bt, et\} \\ \mathbf{h}_5 &= \dots = \{tt, tb, te, rt, lt, bt, et\} \\ \mathbf{h}_6 &= \dots = \{tt, tr, tl, tb, te, rt, lt, bt, et\} \end{aligned}$$

$$\mathbf{h}_7 = \dots = \{tt, tr, tl, tb, te, rt, lt, bt, et\}$$

where, for clarity, vectors are shown as the set of elements equal to one. Since $\mathbf{h}_6 = \mathbf{h}_7$, the set of states satisfying $\text{EF}(\{et\})$ is $\{tt, tr, tl, tb, te, rt, lt, bt, et\}$, namely, all states except the absorbing ones.

Example 14.8

For the two dining philosophers model, determine which states satisfy $\text{AF}(\text{Phil.1 is eating})$.

First, convert the expression into one containing only EX, EG, EU:

$$\begin{aligned} \text{AF}(\text{Phil.1 is eating}) &= \neg\text{EG}(\neg\text{Phil.1 is eating}) \\ &= \neg\text{EG}(\text{Phil.1 is not eating}) \end{aligned}$$

The set of states satisfying the atomic proposition “Phil.1 is not eating” is $\mathcal{S} \setminus \{et\}$. Next, we use the iterative algorithm for EG:

$$\begin{aligned} \mathbf{h}_0 &= \{tt, tr, tl, tb, te, rt, rr, lt, ll, bt\} \\ \mathbf{h}_1 &= (\text{Set of states reaching } \mathbf{h}_0) \wedge (\mathcal{S} \setminus \{et\}) \\ &= \{tt, tr, tl, tb, te, rt, rr, lt, ll\} \\ \mathbf{h}_2 &= (\text{Set of states reaching } \mathbf{h}_1) \wedge (\mathcal{S} \setminus \{et\}) \\ &= \{tt, tr, tl, tb, te, rt, rr, lt, ll\} \end{aligned}$$

Finally, we complement the result from EG, and we obtain that the set of states satisfying $\text{AF}(\text{Phil.1 is eating})$ is $\{bt, et\}$.

