The function $V_t^{\mu}(\mathbf{x})$ gives the expected reward starting in state $\mathbf{x}$ at time t over the remainder of the truncated horizon $t, t+1, \ldots, T$. We call this truncated problem the ***t-subproblem.*** The principle of optimality then states the following:

THEOREM D.2 *If* $\mu^* = \{\mu_1^*, \mu_2^*, \ldots, \mu_T^*\}$ *is an optimal policy for the problem (D.1), then the truncated policy* $\{\mu_t^*, \mu_{t+1}^*, \ldots, \mu_T^*\}$ *is optimal for the* ***t-subproblem.*** *That is,*

$$V_t^{\mu^*}(\mathbf{x}) = \sup_{\mu \in \mathcal{M}} V_t^{\mu}(\mathbf{x}) \quad \forall \mathbf{x} \in S_t.$$

We omit a formal proof of this fact, but it is easy to see why it holds. Indeed, suppose $\{\mu_t^*, \mu_{t+1}^*, \ldots, \mu_T^*\}$ was not optimal for the ***t-subproblem.*** and another policy, $\hat{\mu}$, yields strictly greater expected reward. If this were true, then the policy

$$\{\mu_1^*, \ldots, \mu_{t+1}^*, \hat{\mu}_t, \hat{\mu}_{t+1}, \ldots, \hat{\mu}_T\}$$

would produce a strictly greater expected reward than does the policy μ^* for the original problem, which contradicts the optimality of μ^*. Hence, μ^* must be optimal for the ***t-subproblem.***

The Dynamic Programming Recursion

The principle of optimality leads naturally to a recursive procedure for finding the optimal policy. First, for all $\mathbf{x} \in S_t$ and all $t = 1, \ldots, T$, define the optimal reward-to-go, called the *value function,* by

$$V_t(\mathbf{x}) = \sup_{\mu \in \mathcal{M}} V_t^{\mu}(\mathbf{x}).$$

The value function gives the optimal expected reward from time t onward given that we are in state $\mathbf{x}$ at time t. Note that $V_1(\mathbf{x})$ is the optimal expected reward for the original problem with initial state $\mathbf{x}$. The principle of optimality leads to the following recursive procedure for determining the value function:

PROPOSITION D.15 *The value function* $V_t(\mathbf{x})$ *is the unique solution to the recursion*

$$V_t(\mathbf{x}) = \max_{\mathbf{u} \in U_t(\mathbf{x})} E\left[g_t(\mathbf{x}, \mathbf{u}, \mathbf{w}(t)) + V_{t+1}(\mathbf{f}_t(\mathbf{x}, \mathbf{u}, \mathbf{w}(t)))\right], \tag{D.2}$$

for all $t = 1, \ldots, T$ *and all* $\mathbf{x} \in S_t$, *with boundary conditions*

$$V_{T+1}(\mathbf{x}) = g_{T+1}(\mathbf{x}), \quad \mathbf{x} \in S_{T+1}.$$

Moreover, if $\mathbf{u}^* = \mu_t^*(\mathbf{x})$ *achieves the maximum in (D.2) for all* t *and* $\mathbf{x} \in S_t$, *then* $\mu^* = \{\mu_1^*, \mu_2^*, \ldots, \mu_T^*\}$ *is an optimal policy.*

We omit a formal proof of this fact, but again the reasoning is quite intuitive—namely, since $V_{t+1}(\mathbf{x}(t+1))$ measures the optimal expected reward given state $\mathbf{x}(t+1)$ in the next time-period, $t+1$, the optimal value of the ***t-subproblem*** should be the result of maximizing the sum of the current expected reward, $E[g_t(\mathbf{x}, \mathbf{u}, \mathbf{w}(t))]$, and the expected reward from the $(t+1)$-subproblem, $E[V_{t+1}(\mathbf{x}(t+1)] = E[V_{t+1}(\mathbf{f}_t(\mathbf{x}, \mathbf{u}, \mathbf{w}(t)))]$. This is precisely what (D.2) does. The result yields the optimal ***t-subproblem*** value function $V_t(\mathbf{x})$ and the process is repeated.

The complexity of this recursion depends on the size of the state space S_t, control space U_t, and number of time-periods T. The worst-case complexity is $\sum_{t=1}^{T} |S_t||U_t|$,