Online Appendix

We present here the details of our HRL implementation and the simulations briefly described in the main text. For clarity, we begin by describing our implementation of non-hierarchical RL, which was used in the simulations including only primitive actions. This will then be extended, in the next section, to the hierarchical case. All simulations were run using Matlab (The Mathworks, Natick, MA). Code is available for download at www.princeton.edu/~matthewb.

Basic Actor-Critic Implementation

Task and representations. Following the standard RL approach (see Sutton & Barto, 1998), tasks were represented by four elements: a set of states S, a set of actions A, a reward function R assigning a real-valued number to every state transition, and a transition function T giving a new state for each pairing of a state with an action. In our simulations, S contained the set of location tiles in the layout depicted in Figure 4A; A contained eight single-step movements, following the principle compass directions; R yielded a reward of 100 on transitions to the goal state indicated with a G in Figure 4A, otherwise zero; and T was deterministic. All actions were available in every state, and actions yielded no change in state if a move into a wall was attempted. Our choice to use deterministic actions was for simplicity of exposition, and does not reflect a limitation of either the RL or HRL framework.

Architecture. The basic RL agent comprised actor and critic components. The actor maintained a set (matrix) of real-valued strengths (W) for each action in each state. The critic maintained a vector V of values, attaching a real number to each state.

Training. At the outset of training, action strengths and state values were initialized to zero; the state was initialized to the start location indicated in Figure 4A; and a time index t was initialized at zero. On each step of processing, t, an action was selected probabilistically according to the softmax equation:

Eq. 1 $$P(a) = \frac{e^{W(s_t,a)/\tau}}{\sum_{a' \in A} e^{W(s_t,a')/\tau}}$$

where $P(a)$ is the probability of selecting action a at step t; $W(s_t, a)$ is the weight for action a in the current state; and τ is a temperature parameter controlling the tendency toward exploration in action selection (10 in our simulations). The next state (s_{t+1}) was then determined based on the transition function T, and the reward for the transition (r_{t+1}) based on R. Using these, the temporal-difference (TD) prediction error (δ) was computed as

Eq. 2 $$\delta = r_{t+1} + \gamma V(s_{s+1}) - V(s_t)$$

where γ is a discount factor (0.9 in our simulations). The TD prediction error was then used to update both the value function and the strength for the action just completed:

Eq. 3 $V(s_t) \leftarrow V(s_t) + \alpha_C \delta$

Eq. 4 $W(s_t, a) \leftarrow W(s_t, a) + \alpha_A \delta$

The learning rate parameters α_C and α_A were set to 0.2 and 0.1, respectively. Following these updates, *t* was incremented and a new action was selected. The cycle was repeated until the goal state was reached, at which point the agent was returned to the start state, *t* was reinitialized, and another episode was run.

HRL Implementation

Our implementation of HRL was based on the options framework described by Sutton et al. (1999), but adapted to the actor-critic framework.

Task and Representations. The set of available actions was expanded to include options in addition to primitive actions. Each option was associated with (1) an initiation set, indicating the states where the option could be selected; (2) a termination function, returning the probability of terminating the option in each state; and (3) a set of option-specific strengths W_o, containing one weight for each action (primitive or abstract) at each state.

For the four-rooms simulations, two options could be initiated in each room, each terminating deterministically at one of the room's two doors. Each option also had a pseudo-reward function, yielding a pseudo-reward of 100 at the option's termination state. For simplicity, each option was associated with strengths only for primitive actions (i.e., not for other options). That is, option policies were only permitted to select primitive actions. As indicated in the main text, options are ordinarily permitted to select other options. This more general arrangement is compatible with the implementation described here.

Architecture. In addition to the option-specific strengths just mentioned, the actor maintained a 'root' set of strengths, used for action selection when no option was currently active. The critic maintained a root-level value function plus a set of option-specific value functions V_o.

Training. Since primitive actions can be thought of as single-step options, we shall henceforth refer to primitive actions as 'primitive options' and temporally abstract actions as 'abstract options,' using the term 'option' to refer to both at once. The model was initialized as before, with all option strengths and state values initialized to zero. On each successive step, an option o was selected according to

Eq. 5 $$P(o) = \frac{e^{W_{o_{ctrl}}(s_t,o)/\tau}}{\sum_{o' \in O} e^{W_{o_{ctrl}}(s_t,o')/\tau}}$$

where O is the set of available options, including primitive options; o_{ctrl} is the option currently in control of behavior (if any); and $W_{o_{ctrl}}(s_t, o)$ is the option-specific — i.e., o_{ctrl}-specific — strength for option o (or the root strength for o in the case where no option is currently in control). Following identification of the next state and of the reward (including pseudo-reward) yielded by the transition, the prediction error was calculated for all *terminating* options, including primitive options, as

Eq. 6 $$\delta = r_{cum} + \gamma^{t_{tot}} V_{o_{ctrl}}(s_{t+1}) - V_{o_{ctrl}}(s_{init})$$

where t_{tot} is the number of time-steps elapsed since the relevant option was selected (one for primitive actions); $s_{t_{init}}$ is the state in which the option was selected; o_{ctrl} is the option whose policy selected the option that is now terminating (or the root value function if the terminating option was selected by the root policy); and r_{cum} is the cumulative discounted reward for the duration of the option:

Eq. 7 $$r_{cum} = \sum_{i=1}^{t_{tot}} \gamma^{i-1} r_{t_{init}+i}$$

Note that $r_{t_{init}+i}$ incorporated pseudo-reward only if $s_{t_{init}+i}$ was a subgoal state for o_{ctrl}. Thus, pseudo-reward was used to compute prediction errors 'within' an option, i.e., when updating the option's policy, but not 'outside' the option, at the next level up. It should also be remarked that, at the termination of non-primitive options, two TD prediction

errors were computed, one for the last primitive action selected under the option and one for the option itself (see Figure 3).

Following calculation of each δ, value functions and option strengths were updated:

Eq. 8 $$V_{o_{ctrl}}(s_{t_{init}}) \leftarrow V_{o_{ctrl}}(s_{t_{init}}) + \alpha_C \delta$$

Eq. 9 $$W_{o_{ctrl}}(s_{t_{init}}, o) \leftarrow W_{o_{ctrl}}(s_{t_{init}}, o) + \alpha_A \delta$$

The time index was then incremented and a new option/action selected, with the entire cycle continuing until the top-level goal was reached.

In our simulations, the model was first pre-trained for a total of 50000 time-steps without termination or reward delivery at *G*. This allowed option-specific action strengths and values to develop, but did not lead to any change in strengths or values at the root level. Thus, action selection at the top level was random during this phase of training. In order to obtain the data displayed in Figure 4 C, for clarity of illustration, training with pseudo-reward only was conducted with a small learning rate ($\alpha_A = 0.01$, $\alpha_C = 0.1$), reinitializing to a random state whenever the relevant option reached its subgoal.