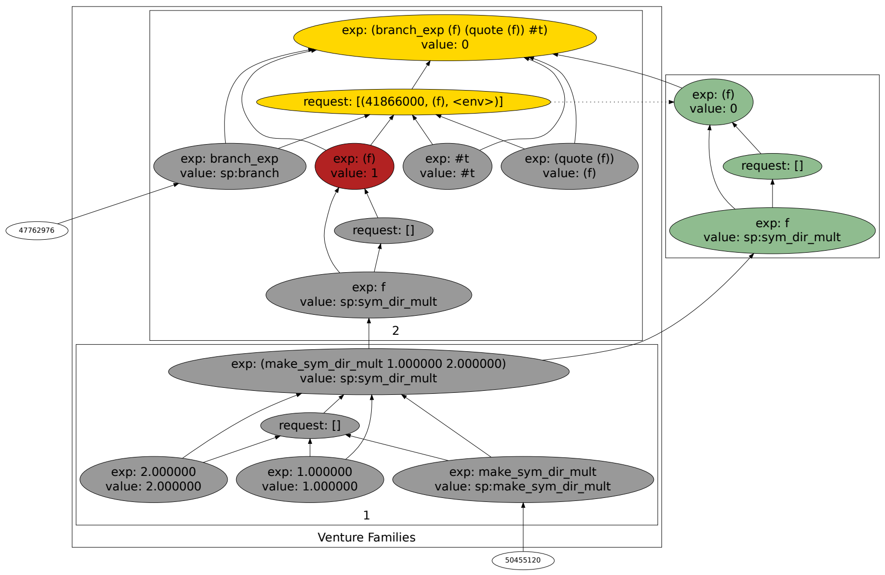
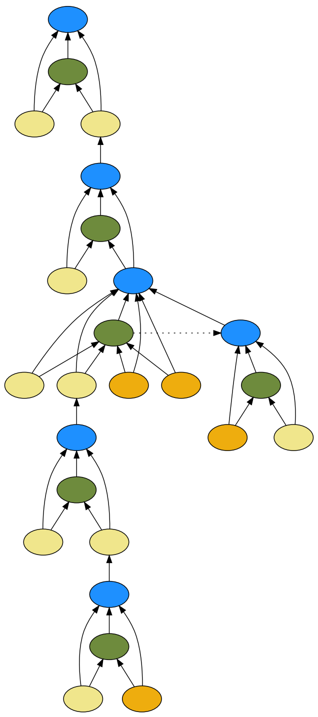
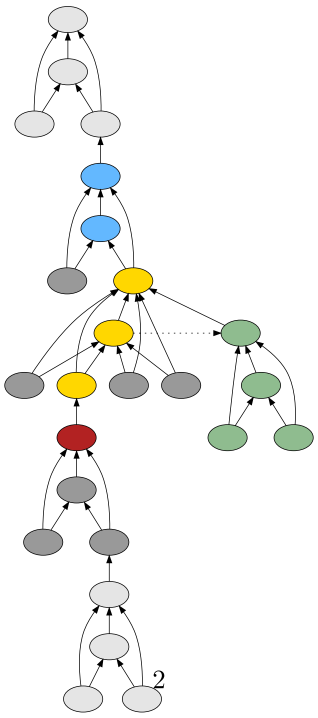


(a) The result of makeSymDirMult is An SP whose applications are exchangeably coupled. Here the result of one application determines whether or not another application is made. Thus the second application is in the brush of the scaffold generated by the first application. If we do not detach the brush first, we may propose to the first application conditioned on an application that will not exist in the new trace.

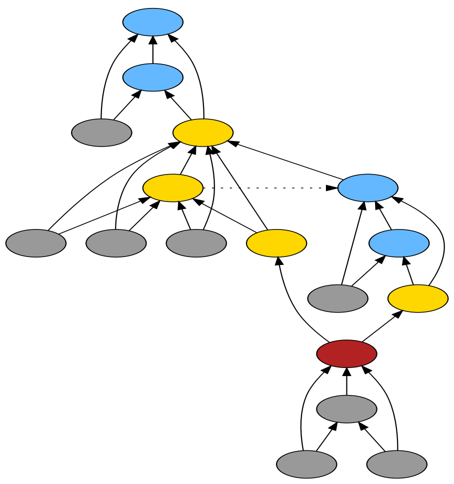
(b)
TODO
show
code

Figure 1: The need to detach first

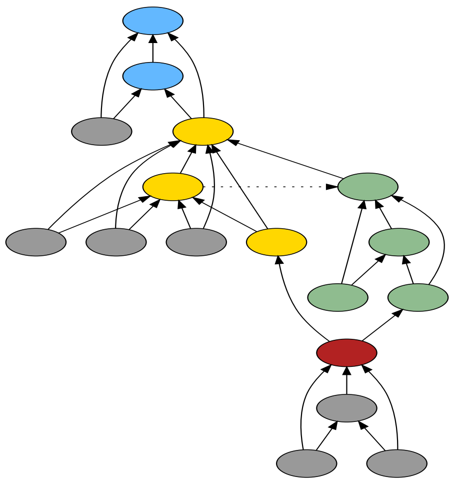

(a) A trace.



(b) The scaffold partitions the trace into five groups: the nodes that will definitely still exist in the proposal trace but whose values may change (drg), the nodes that we will definitely compute likelihoods at (absorbing), the nodes that may no longer exist (the brush),

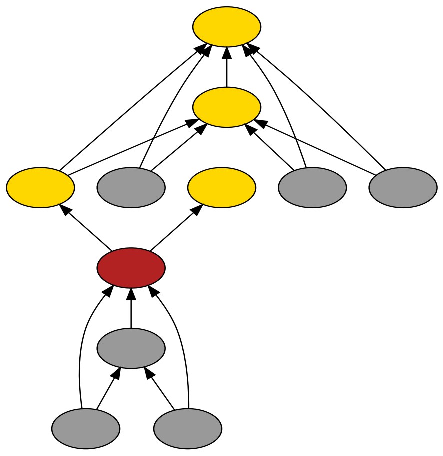


(a) To construct a scaffold, we first walk downstream from the principal node, and color gold every node whose value may change, and blue every node at which we can absorb.

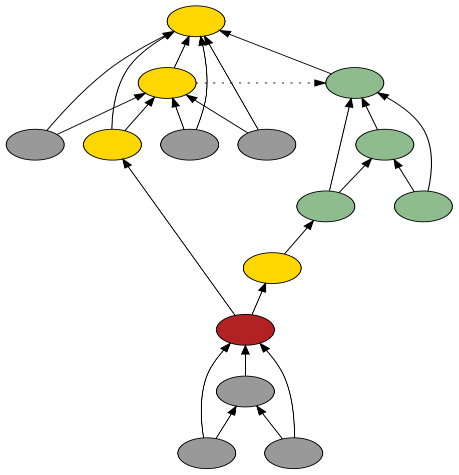


(b) Next, we color green every node that may no longer exist once the gold nodes are resampled. At this point, the red and gold nodes constitute the definite regeneration graph, the blue nodes constitute the absorbing border, and the green nodes constitute the brush.

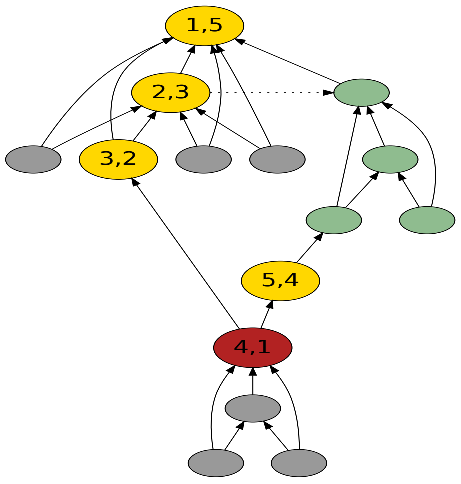
Figure 3: The two stages of constructing a scaffold



(a) A torus with two border nodes. Suppose we regenerate the higher one first, and one of the nodes regenerated makes a simulation request. Regen then hands over control to eval in order to evaluate the expression. (TODO mark the three nodes that have values, and space-permitting have an extra figure to start that just shows the scaffold.)

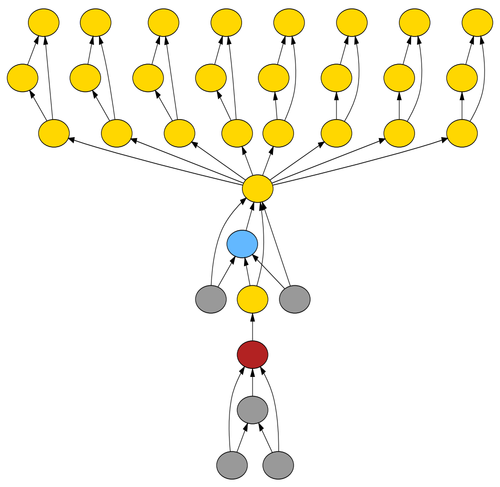


(b) Evaluating the expression might involve referencing other nodes in the trace, for example to resolve a variable lookup. Those nodes may be in the drg and may not have been regenerated yet, so eval must hand over control to regen to guarantee that all values have been regenerated before they are used.

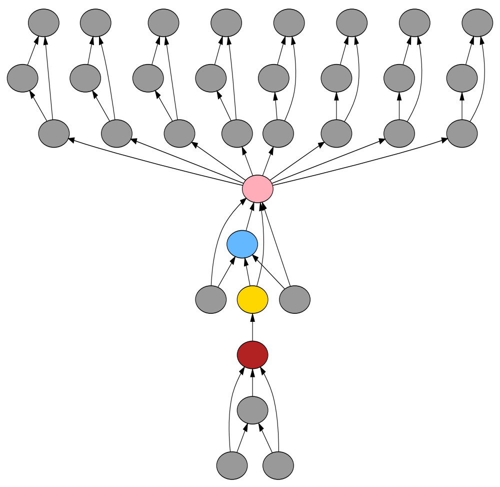


(c) The same trace as figure with the first-visit order and regeneration order displayed.

Figure 4: Tracking the regen recursions

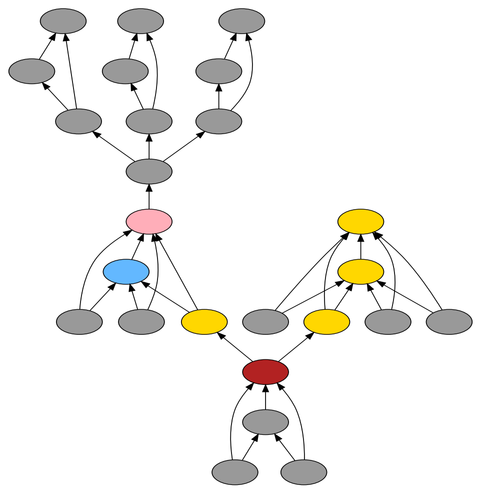


(a) A large scaffold for sampling a hyperparameter.

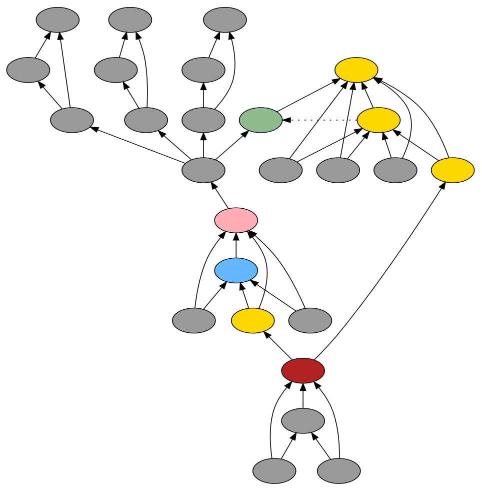


(b) The application of the maker SP computes the log density of all of its applications for us. We say that the maker SP "absorbs at applications".

Figure 5: Absorbing at applications (AAA)

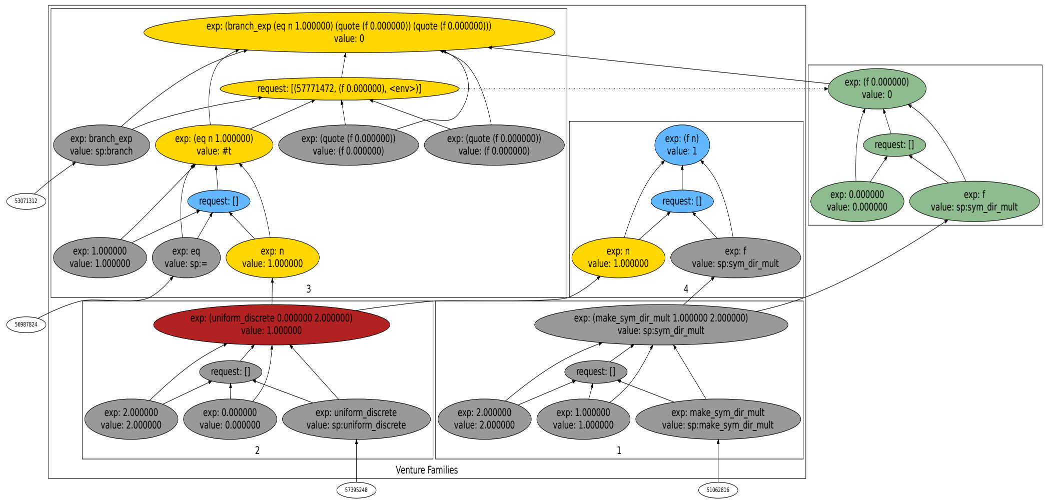


(a) A scaffold with three border nodes, one of which is absorbing at applications. The child of the aaa node is a reference to it, and would normally be in the drg. (TODO the extended drg and the extended scaffold should refer to the semantic drg/scaffold, before the aaa shortcut, even though it is never constructed. We should abandon the old use of that term)



(b) A simulation request may lookup that child before the aaa node has been regenerated, and call regen on it. Regen is normally a no-op for nodes that are not in the drg, but even though the child is not in the drg, it and its parent must be regenerated nonetheless. Thus we need to add the child to the drg dynamically.

Figure 6: Challenges with absorbing at applications



(a) A procedure with partial-exchangeable coupling is applied in both the absorbing border and the brush. The probability of the absorbing nodes depends on the order in which we regenerate. As we are detaching, we need to compute the probability of the absorbing nodes that we would have calculated if we had proposed this trace by regenerating along the scaffold. In particular, if we would have visited the absorbing application first during regen, then we must visit it last during detach. We solve this problem by having detach always detach all nodes in the opposite order from how regen would have generated them. (TODO make this a finite collapsed hmm)

Figure 7: The need to detach in the opposite order from how we would have regenerated