We would like to thank the reviewers for their comments, and in particular appreciate the insightful suggestions for future work by R1 and R2. We start by focusing on addressing significant misunderstandings and unfounded assertions, then describe some additional experiments, and finally discuss some of the other comments and questions.

Misunderstandings:

- **R4:** *"The proposed method may suffer low learning efficiency. As reward comes from downstream tasks, supervision on early actions could be fairly weak."* This is unfounded, and it appears the reviewer has misunderstood critical details. A central idea in GFSA is to analytically solve for the marginal distribution over halting behavior, and to directly differentiate through the solution with implicit differentiation (Sec 3.2). There is no reason why supervision on early actions would be weak. Table 1 shows that GFSA has excellent sample efficiency, and Fig 1 shows an example of many-step behavior. The LASTREAD policy shown there takes an average of 35 actions before accepting.
- **R4:** *"It is difficult to see how GFSA could support multiple edge addition."* It's unclear if the reviewer means multiple edges per graph (GFSA does this), multiple edges per start node (GFSA does this, but we will add a sentence to emphasize this), or multiple edge types (GFSA does this using automaton states).
- **R4:** *"The action 'backtrack' could be problematic. While this action enables the model to handle the cases where no edge addition is needed or discovered, it could also make the model run into endless loops."* It is not problematic. As long as the policy places positive probability on STOP and/or ADDEDGEANDSTOP, the Markov chain will terminate with probability 1. In practice, when solving for marginals, we treat backtracking as termination and then correct for backtracking as a post-processing step (See L569-576 in Appendix).
- **R4:** *"How does the final reward guide the model to minimize the chance of triggering 'backtrack'?"* We emphasize that the GFSA layer is formulated in terms of a POMDP, but we are not using RL and have no reward function. The GFSA layer is just a deterministic, differentiable layer used within a supervised learning architecture. The end-to-end loss encourages useful ADDEDGEANDSTOP actions, so no explicit "don't backtrack" signal is needed.

Additional experiments and differences from RL approaches: At a fundamental level, the GFSA layer and RL approaches have different types of output. The GFSA layer produces a full distribution (represented as a continuous-valued vector) that can be transformed nonlinearly (e.g. $f(\mathbb{E}[\tau])$ where f is the downstream model and loss and τ are edge additions from trajectories). In contrast, RL approaches produce stochastic discrete samples. As such, it is not possible to "drop in" a standard RL approach instead of GFSA; one must first reformulate the model, task, and training loop in terms of expected reward and discrete latent variables ($\mathbb{E}[f(\tau)]$). Nevertheless, we ran an experiment inspired by the "Go For a Walk" paper suggested by R3, training a standard RL agent with the same parameterization as GFSA on modified versions of our tasks. For the program analysis tasks, we replace the cross-entropy loss with a reward of +1 for adding a correct edge (or correctly not adding any) and 0 otherwise. We train using REINFORCE with 20 rollouts per start node and a leave-one-out control variate. It suffers from high variance, and the best version underperforms GFSA. For 100k examples, 1x size: 94.2% v.s. 100%, 96.7% v.s. 99.6%, 98.1% v.s. 99.5%. Because edges are added by single trajectories rather than marginals over trajectories (as in GFSA), these agents are unable to learn to add multiple edges per start node. For the variable misuse task, we use the final classification log-likelihood as a reward. Simply computing this reward requires a full downstream model forward pass, so we run only one rollout per example with a learned scalar reward baseline. The best model with these RL-based edges performs similarly to the model on the base AST graph alone, and does not learn to add useful edges. We will describe these experiments in more detail in the next revision.

Prior work: We would like to thank R2, R3, and R4 for their additional citations, and especially R2 for the Relation-Aware Transformer paper, which indeed appears to be equivalent to the "RelAtt" model. We will include these in the next revision, along with a discussion of GFSA v.s. transformer attention as suggested by R1.

Experimental gains: For the variable misuse task, although the gains of our method are only a few percent, even the hand-engineered edges from previous work lead to only a few percent improvement over the base graph after a thorough hyperparameter search. We have thus shown that an end-to-end-trained model can outperform hand-engineered edges for these tasks, and verified that these differences are statistically significant on our test set.

Future work: Thanks R1 and R2 for interesting discussions and suggestions about how to take this work forward. We agree with R1 that applying GFSA ideas to other domains is an exciting next step. Similarly, R2's suggestion to consider pairing GFSA with code-based self-supervised pre-training is a really interesting suggestion. We hadn't thought of framing the start point-specific observations as a form of variable capture, but that's a nice way of thinking about it. Conditioning on the initial node is straightforward because we can still marginalize out the agent's history while solving for the distribution. Modifying the language based on agent history would make this harder, but might be tractable if we could enumerate the possible "captures" and solve them jointly. A related idea we've considered is adding a stack of states, corresponding to context-free grammars; we hope to explore this and similar ideas in the future. We also hope to explore methods for encouraging sparsity; we ran some experiments with entropy regularization after the paper submission but unfortunately this decreased performance on the downstream task.