We thank the reviewers for their positive feedback. Please find below a reply to the individual comments.

Reviewer 1.
Perhaps the discussion of related work could mention and compare with theorem proving in algebra with ATPs [...]
Thank you for the comment. We will correct this in the updated manuscript, and add references to papers that learn guidance of elementary inference rules.

Some of the ideas could be explained in a more high-level way. For example, the intuition behind the reward function.
Our choice of the reward function has a simple interpretation: a positive reward is given whenever the chosen action results in a bound improvement. That is, if the initial bound is $p(x) \leq 20$, and the chosen action has allowed to establish $p(x) \leq 19$, the agent is rewarded 1. This reward scheme has the benefit of being dense (as rewards are given gradually), in addition to being unsupervised (no ground truth bound required). The specifics of how we compute the bound at each step is described in Sec. 4.2. We will complement the current description in the paper with a more intuitive explanation in Sec. 4.2.

The evaluation could also be made more convincing by including more problems than just stable set.
Please see answer to Reviewer 3.

Reviewer 2.
Are there situations for which [the dynamic approach] takes much longer than the static approach? If yes, is it possible to identify these situations?
Thank you for raising this important point. In general, the dynamic approach cannot take much longer than the static approach; in fact, if we set the proof length of the dynamic method to be sufficiently large, all the proof generators (of the form $\mathbf{x}^{\alpha}(1-\mathbf{x})^{\beta}$) will eventually be added to the memory, provided the degree of intermediate polynomials is not restricted. In this case, the LPs solved by the static approach and the dynamic approach are the same, hence overall the dynamic approach will only take slightly more time due to intermediate computations. We will add this point to the updated manuscript.

Does the proposed dynamic approach always find a solution (e.g. a fully verifiable proof)? Could it happen that the approach fails completely?
The completeness of the proof system guarantees that a proof of finite length exists for any positive polynomial. Hence, if we run the dynamic approach for long enough, a proof will be found. We note that in our experiment, we restrict the degree of the intermediate lemmas to 2. In this case, we cannot provably guarantee that the method will find the optimal bound all the time, however the bounds obtained in practice are much better those obtained by the static method.

Reviewer 3.
Have the authors tried their method on other problems and found improvements?
Thank you for raising this point. We first note that the stable set problem is sufficiently general in that many combinatorial problems (e.g., max matching, SAT, etc...) can be cast as specific instances of the stable set problem, via a simple reduction. We have done some preliminary experiments using our prover (trained on random graphs) to see how our method performs on such problems. Specifically, we evaluated the prover on the maximum matching problem on random graphs of size 30, where we used the property that the maximum matching corresponds to the maximum stable set of the associated line graph. Our results show that the proposed dynamic approach significantly outperforms the static approach, even though these problems come from a distribution that is different from what is seen during training. In particular, while the dynamic approach obtained an average upper bound of 13.8 with an LP of size ≤ 200, the static approach only matched this upper bound by going to the 4th level of the hierarchy, with an LP of size $\approx 1.1 \times 10^{6}$. We therefore strongly believe our proposed approach can go well beyond the stable set problem, and can tackle other combinatorial problems via such reductions.
It should be noted that other (potentially more efficient and direct) ways of modeling such problems using polynomials also exist. While we did not try to train an agent specifically with these formulations, we see this as an exciting avenue of future research.

How essential is it that the agent uses a neural network? How much of the improvement is due to using the reinforcement learning framework (perhaps with a simple classifier), vs. due to specifically using a neural network?
In our setting, the neural network is used to decide the next proof step from the current proof state. To model this complex procedure, it is crucial to use a highly expressive and *nonlinear* function class. In addition, as argued in the paper, the Q-value function approximator has to incorporate important symmetries about the problem. It can be shown that if one uses a *linear* function, incorporating the "variable relabeling" symmetry significantly constrains the function and drastically reduces the degrees of freedom of the linear function. In contrast, neural networks have much larger expressivity, and were shown to be very successful value function approximators with good generalization properties in previous works.