(Automation) Either a graph or its complement is connected.

What would trigger the algorithm to select these moves when given access to this point-and-click system?

I have a few thoughts on how to automate certain parts.

Automate each "unfold" when a new variable in the context has a syntactic match to a different definition of an existing variable.

The question of syntactic matching is a problem other members of the team are working on, so I won't focus on it too much here.

Automate each "tidy" after every move.

I think auto-tidying is what we do in our heads while doing math.

Automate each "rewrite" when it shortens the context.

This is inspired by a conversation I had with Anand and separately with Fabian. We were asking — when are we inspired to assign a name to an expression that appears in the context? They mentioned that if creating a separate expression consolidates the codebase/context, that expression should be factored out and named (a la Dreamcoder).

In the proof, we employed this here: instead of saying u and v are vertices in $V(G)$ and x and y are vertices in $V(G^c)$, it takes less space to say that they are all members of both. So, we do.

Automate each "specialize" when the search space for an existential is bigger than a certain threshold (e.g. exponential).

At a certain point in this example, the algorithm no longer has routine moves available (it has unfolded everything it can), but it's still trying to construct an existential in an exponential search space.

– –

$$\exists \text{ vertices } u, v \in V(G) \text{ with no path of length } \leq 2 \text{ between them in } G$$

$$x, y \in V(G^c)$$

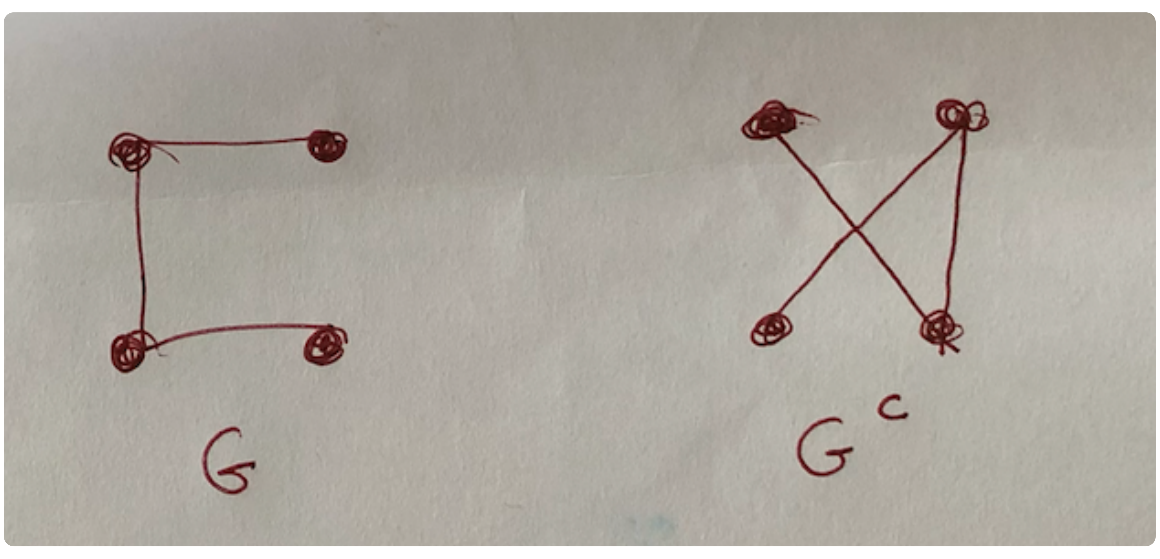
=====================================

$$\exists \text{ path between } x \text{ and } y \text{ in } G^c$$

– –

At this "stuck" point, we can ask it to try to prove the converse: If the graph is connected, is its complement not connected?

Indeed, it would end up disproving the converse, thus **failing**, ideally by considering the graph below where both the graph and its complement are connected.



It should then **learn from failure** — the fact the statement is not bidirectional should signal to the program there is some strengthening that can be done. That is, it should **strengthen to balance bidirectionality** , and ultimately **remove distracting assumptions** (like that no path at all exists between two vertices in G) and **parameterize the construction** (that its sufficient to consider when no path of length 1 or 2 exists between two vertices in G).

But that was all quite abstract…now for slightly more detail:

- Suppose the program finds the counterexample to the converse (as a program like Graffiti likely knows how to do…ideally we would have built in this existing functionality…).
- Then, we'll have some coded meta-reasoning heuristic that says "if the converse isn't true, perhaps the statement should be strengthened."
- Well, there are two ways to strengthen — weaken hypothesis and strengthening target. How might we be guided to parameterize something in the hypothesis?
- Well, I think as I was proving the problem, I actually strengthened the target first (that the complement must have a diameter of at most 3). I think perhaps the program can look at a few examples (as Graffiti does), and try to identify graph properties that (1) a few of the example graphs that are in disproofs of the converse have in common and (2) imply connectedness. Hopefully, it will identify the property "graph diameter is at most 3." Then, it can strengthen the target in that way. I think this is quite akin to how humans might approach this sort of problem.
- Then, using routine moves, the algorithm could reason backward from the target. We see every vertex in the graph goes through at most two other vertices to get to any other. Which two vertices? At this point, syntactic similarity suggests we should plug in the two ones we have in our context — u and v. At this point, hopefully routine moves can prove why this works. I've glossed over details here, but this is the "conflict-guided" automation idea.