CS103 Handout 12
Fall 2020 October 2, 2020

Discrete Structures Proofwriting Checklist

Now that we're transitioning to writing proofs about discrete structures like binary relations, functions, and graphs, there are a few specific points we'd like to you to check for in your proofs in addition to the general rules set out in the earlier Proofwriting Checklist.

As you're working through the problems on Problem Set Three and beyond, take a few minutes per proof to read over what you've written and check to make sure that your proofs obey the following criteria:

□ ***Appropriately call back to first-order definitions.***

□ ***Don't use quantifiers or connectives in your proofs.***

□ ***Don't repeat definitions; use them instead.***

□ ***Type-check your proofs.***

As before, ***we will be applying this checklist to the proofs that you submit***, starting with Problem Set Three, as part of how we grade, so catching these sorts of issues before you submit will help us give you even better targeted feedback on your proofs.

The remainder of this handout goes into more detail about what each of these rules mean.

Appropriately Call Back to First-Order Definitions

One of the main differences between the proofs that you'll be writing on Problem Set Three and the proofs you did on Problem Set One and Problem Set Two is that we now have symbolic definitions of key terms and definitions expressed in the language of first-order logic. From a proofwriting perspective, this is quite useful – it means that you know exactly what it is that you'll need to prove at each point in time. At the same time, this means that you need to be extremely careful when setting up your proofs to make sure that you're making the right assumptions and ultimately trying to prove the right results.

For example, let's suppose that you're trying to prove that some binary relation R over a set A is transitive. This means that you need to prove that

$$\forall x \in A.\ \forall y \in A.\ \forall z \in A.\ (xRy \land yRz \to xRz).$$

Given that this is the statement you want to prove, you should ***not*** start your proof off like this:

> ***Incorrect!*** ***Proof:*** Consider any x, $y \in A$ where xRy. Since xRy, we know that […] ■

The issue with this proof is that the structure of what's written above doesn't match the structure of the first-order statement in question. Specifically, since in this proof we've picked an arbitrary x and y from A and assumed that xRy, we're essentially writing a proof that matches the following first-order logic fragment:

$$\forall x \in A.\ \forall y \in A.\ (xRy \to [\ldots])$$

Take a minute to confirm that you see exactly why this is the case.

As you're working through problems on discrete structures, we strongly recommend that, before you spend any mental energy trying to actually tackle the problem, you make sure that you have clearly identified what it is that you're assuming and what it is that you need to prove. The Guide to Proofs on Discrete Structures contains a number of example proof templates that you can use for some common cases, but more generally you'll want to get as much practice as possible going back and forth between first-order logic and proof setup.

There's another way that we often see proofs fail to engage with first-order definitions, and that's to operate at far too high of a level. As an example, if you look on Problem Set Three, you'll see a binary relation E_R defined in terms of a relation R that's reflexive and transitive:

$$xE_Ry \quad \text{if} \quad xRy \land yRx.$$

You're asked to prove that E_R is always an equivalence relation. Here's an (incorrect!) proof of this.

> ***Incorrect!*** ***Proof:*** The E_R relation is defined in terms of the R relation. Since the R relation is an reflexive and transitive and E_R is defined in terms of R, we see that E_R itself must be reflexive and transitive. This means that we just need to show that E_R is symmetric. To see this, note that […] ■

This proof doesn't establish that E_R is reflexive and transitive, and instead relies on a high-level appeal to intuition that if you start with a reflexive, transitive relation and hit it with the Math Hammer to form a new relation, you are guaranteed to end up with a reflexive and transitive relation. The problem here is that this argument essentially takes a much narrower statement ("E_R is an equivalence relation"), which we aren't sure is true, and tries to prove it by appealing to a much broader statement ("any binary relation that can be made out of a reflexive, transitive relation must be reflexive and transitive.") The catch is that we haven't proved that broader statement, and in fact, since it's so broad, we should be even *more* skeptical of it than we are of the claim that the single relation E_R is reflexive and transitive.

In fact, you can show that the statement given above isn't true. Let's consider this related binary relation S_R, which has a similar definition to E_R but with a slight twist:

$$xS_Ry \quad \text{if} \quad xRy \land y\not{R}x.$$

This is a relation that's defined in terms of a reflexive, transitive relation, but it's not reflexive. You can check this by noting that aS_Ra could only happen if aRa and $a\not{R}a$ were true, but that's not possible.

There are two lessons to take away from this example. First, ***be wary about going off-script*** in the course of writing a proof. If you're asked to prove that something is a strict order, or an equivalence relation, or a bijection, etc., you have a formal definition you can call back to, and chances are that it's probably best to prove that the object has the given type by explicitly appealing to each of the parts of this definition. If you decide not to do that and to prove the result through some other mechanism, you should be very suspicious about the line of reasoning you're following.

This brings us to the second lesson here – ***exercise skepticism about broad claims in your proofs***. If you're making some claim about how all equivalence relations behave, or what every function looks like, etc., your immediate response should be to ask whether what you're saying is actually true. If so, why? Go and prove it. If not, why not? Go disprove it. In the course of doing so, you'll either discover a flaw in your original line of reasoning, which is great (it means that you've spotted an error in your proof and can go and try to address it), or you'll end up with a more robust proof because you'll have justified a critical and non-obvious claim you've made.

Don't Use Quantifiers or Connectives in Your Proofs

The title of this section says it all – the convention in proofwriting is to not use quantifiers ($\forall$ or $\exists$) or connectives ($\land$, $\lor$, $\to$, etc.) in the course of writing up a mathematical proof. There's no fundamental reason why proofs couldn't be written using these symbols. It's just the way that the mathematical community decided to do things. Since part of the goal of this class is to get you to write proofs in way that matches established guidelines, we'll be enforcing this rule pretty strictly.

The good news is that many violations of this rule are quite easy to fix. For example, suppose that we're trying to prove that some function $f : A \to B$ is surjective. Here's one possible not-so-great way to start off a proof to that effect:

Incorrect! ***Proof:*** We will prove that the function f is surjective. To do so, we need to prove that $\forall b \in B.\ \exists a \in A.\ f(a) = b$. So consider $\forall b \in B$. We will prove that $\exists a \in A$ where $f(a) = b$. [...] ■

Here, the first-order logic notation that's present in the proof is just a restatement of the formal definition of surjectivity, and from context it looks like the author of the proof may have included it to make clear that they know what the definition of surjectivity is and to remind the author of that definition.

As you saw in the original Proofwriting Checklist, it's almost never a good idea to restate definitions in the abstract ("Don't restate definitions; use them instead"). You can assume that whoever is reading your proof knows what a surjective function is, so restating the definition of a surjective function doesn't actually accomplish anything here. We can safely delete that entire sentence from the proof, leaving us with this much simpler proof setup:

Proof: We will prove that the function f is surjective. To do so, consider an arbitrary $b \in B$. We will prove that there is some $a \in A$ where $f(a) = b$. [...] ■

This proof contains all of the important details of the original proof, but without any quantifiers or connectives.

Another common case where we see people using first-order logic in proofs is in the context of working with discrete structures (often, binary relations) that are themselves defined in first-order logic. Stealing an example from the Guide to Proofs on Discrete Structures, let's suppose that you have the following binary relation R defined over the set $\mathbb{Z}$:

$$xRy \quad \text{if} \quad \exists k \in \mathbb{N}.\ (k \neq 0 \ \land\ x + k = y).$$

Imagine that you want to prove that R is transitive. Here's a not-so-great way to do this:

Incorrect! ***Proof:*** Consider any $x, y, z \in \mathbb{Z}$ where xRy and yRz. We need to prove that xRz.

Since xRy, we know that $\exists k \in \mathbb{N}.\ (k \neq 0 \ \land\ x + k = y)$. Similarly, since yRz, we know that $\exists r \in \mathbb{N}.\ (r \neq 0 \land y + r = z)$. Since $x + k = y$ and $y + r = z$, we see that

$$x + k + r = y + r = z. \tag{1}$$

Additionally, since k and r are natural numbers where $k \neq 0$ and $r \neq 0$, we know that $k > 0$ and that $r > 0$, and so $k + r > 0$. Therefore, $k + r \neq 0$. Consequently, we see that $\exists s \in \mathbb{N}.\ (s \neq 0 \ \land\ x + s = z)$, since we can pick $s = k + r$. Therefore, xRz, as required. ■

The logic of the above proof is entirely correct, and so *conceptually* there's nothing wrong with the proof. The issue here is that this just isn't the way that you're supposed to write proofs about definitions given in first-order logic. Instead of using the formal first-order logic notation, the convention is to render the equivalents of these statements in plain English. Here's a better way to write up the above proof:

Proof: Consider any $x, y, z \in \mathbb{Z}$ where xRy and yRz. We need to prove that xRz.

Since xRy, we know that there is a nonzero natural number k where $x + y = k$. Similarly, since yRz, we know that there is a nonzero natural number r where $y + r = z$. Since $x + k = y$ and $y + r = z$, we see that

$$x + k + r = y + r = z. \tag{1}$$

Additionally, since k and r are natural numbers where $k \neq 0$ and $r \neq 0$, we know that $k > 0$ and that $r > 0$, and so $k + r > 0$. Therefore, $k + r \neq 0$. Consequently, we see that there is a natural number s (namely, k+r) such that $s \neq 0$ and $x + s = z$. Therefore, xRz, as required. ■

All we've done here is replace denser first-order logic notation with plain English. The reasoning here is completely the same.

Don't Repeat Definitions; Use Them Instead

"Hey!," you're probably saying. "Isn't this something covered in the original Proofwriting Checklist?" Yes. Yes it is. As we transition to proofs on discrete structures, this rule will become even more relevant, and so we thought it was worth revisiting.

On Problem Set One and Problem Set Two, we gave you the advice to avoid restating definitions purely in the abstract and to instead use them in a way that demonstrates how that definition is applied. For example, we recommended replacing statements like the ones on the left with one like what's on the right:

We know that $x \in A$. Since $A \subseteq B$, we know that every element of A is an element of B. Thus we see that $x \in B$.

We know that $x \in A$. Since $A \subseteq B$, we know that every $x \in A$ satisfies $x \in B$. Therefore, we see that $x \in B$.

We know that $x \in A$. Since $A \subseteq B$, we know that every $z \in A$ satisfies $z \in B$. Therefore, we see that $x \in B$.

Since $x \in A$ and $A \subseteq B$, we see that $x \in B$.

There are a few reasons why it's wise to avoid repeating definitions in the abstract. First, you can assume that the reader knows all of the relevant terms and definitions that are needed in your proofs. Your job as a proofwriter is not to convince the reader of what the definitions are, but to show how those definitions interact with one another to build into some result. In that sense, repeating a definition in the abstract, like what's done above and to the left, doesn't actually contribute anything to the argument you're laying out. The reader already knows the definition, so that sentence is fully redundant.

Second, restating definitions in the abstract risks violating other checklist items. Let's go one at a time through the three options on the left that we advise against. The first one is far too general ("every element of A is an element of B") and therefore breaks our advice of making specific claims about specific variables. The second one ("every $x \in A$ satisfies $x \in B$") is a variable scoping error – is x the specific value referred to in the first sentence, or is it a placeholder? The third one is making specific claims about the variable z and doesn't have a scoping error, but in that case z is purely a placeholder – it doesn't refer to any value. In each of those cases, you can safely delete things.

And finally, restating definitions in the abstract just makes things longer. Compare the three options to the left to the one on the right. All three of those proof fragments are significantly longer than the more concise and direct version shown to the right.

Type-Check Your Proofs

Type checking has been a recurring theme in this class – you've seen this in the set theory translations from Problem Set One and the first-order logic translations from Problem Set Two – and that trend continues forward as you're talking about discrete structures. With the introduction of new terminology from discrete structures, there are more opportunities to make type errors, and so we thought we'd quickly survey some of the common mistakes.

Let's imagine that R is a binary relation over a set A and that x and y are elements of A. There is a distinction between the binary relation R (which you can think of as a sort of predicate) and the statement xRy, which says that x is related to y via the relation R. In other words:

R is of type "binary relation" $\qquad$ xRy is of type "proposition"

Certain adjectives can only be used to describe binary relations, not propositions. For example, only a binary relation can be reflexive, so it makes sense to say something like "R is reflexive" or "R is not reflexive" because R has type "binary relation." However, it is ***not*** correct to say something like "xRx is reflexive" or "xRx is not reflexive," since xRx has type "proposition" and propositions cannot be reflexive or irreflexive. Similarly, if $x \in A$ and R is a binary relation over a set A, it would ***not*** be correct to say "x is reflexive" in place of writing out xRx, since x itself can't be reflexive (unless, of course, A happens to be a set of binary relations, and R is a relation on top of other relations!)

Similarly, let's imagine that $f : A \to B$ is a function and that x is an element of A. There's similarly a difference between the function f (a general transformation that can be applied to elements of A) and the object $f(x)$, which is a specific object contained in the set B. In other words:

f is of type "function" $\qquad$ $f(x)$ is of type "element of B"

This means, for example, that you could say something like "f is injective" or "f is a bijection," but you probably shouldn't write something like "$f(x)$ is injective" or "$f(x)$ is a bijection," since $f(x)$ refers specifically to what you get when you plug x into f. As a note, in practice, mathematicians tend to sometimes be a lot looser with the distinction between f and $f(x)$ and often use one in place of the other, though for the purposes of CS103 we'll expect you to keep the notation distinct to convince us that you understand why they don't represent the same thing.

Both binary relations and functions operate on elements that are drawn from some particular set (the underlying set for a binary relation; the domain for a function). If R is a binary relation over a set A, then you should be careful not to write something like xRy unless you're sure that x and y are elements of the set A. After all, the whole point of having binary relations operate on a specific set is so that we don't have to worry about cases like ★ ⊆ or △ < , and the reason we define domains for our functions is so that we don't end up evaluating expressions like $n^3 - 4n^2$ on inputs like .