Inside PRL

Reading between the parentheses

(/blog/index.html)

Bullets are good for your Coq proofs

2017-02-21 :: coq (/blog/tags/coq.html), by Gabriel Scherer (/blog/tags/by-Gabriel-Scherer.html)

I believe that bullets are one of the most impactful features of recent versions of Coq, among those that non-super-expert users can enjoy. They had a big impact on the maintainability of my proofs. Unfortunately, they are not very well-known, due to the fact that some introductory documents have not been updated to use them.

Bullets are a very general construction and there are several possible ways to use them; I have iterated through different styles. In this post I will give the general rules, and explain my current usage style.

Why bullets

While you are doing a proof, Coq shows a list of subgoals that have to be proved before the whole proof is complete. Most proof steps will operate on the current active subgoal, changing the hypotheses or the goal to prove, but some proof steps will split it into several subgoals (growing the total list of goals), or may terminate the proof of the current subgoal and show you the next active subgoal.

Before bullets, a typical proof script would contain the proofs of each subgoal, one after another.

```
induction foo. (* this creates many subgoal *)

proof of first subgoal.

proof of second subgoal.
```

There are many ways to structure this to make the structure more apparent: people would typically have a comment on each subgoal, or make disciplined use of indentation and blank lines. But, in my experience, a major problem with this style was maintainability in the face of changes to the definitions or parts of automation. It could be very hard of what was happening when a proof suddenly broke after a change before in the file:

- If a proof step now proves *less* things, then what used to be the end of a subgoal may not be anymore. Coq would then start reading the proof of the next subgoal and try to apply it to the unfinished previous goals, generating very confusing errors (you believe you are in the second subgoal, but the context talks about a leaf case of the first goal).

- If a proof step now proves *more* things, it is also very bad! The next proof steps, meant for the first subgoal (for example), would then apply to the beginning of the second subgoal, and you get very confusing errors again.

What we need for robustness is a way to indicate our *intent* to Coq, when we think that a subgoal is finished and that a new subgoal starts, so that Coq can fail loudly at the moment where it notices that this intent does not match reality, instead of at an arbitrary later time.

(The `S*Case` tactics used in (older versions of) Software Foundations can solve this problem if used in a carefully, systematic way, and additionally provides naming.

Alexandre Pilkiewicz implemented an even more powerful cases (https://github.com/pilki/cases) plugin. Bullets are available in standard Coq since 8.4 (released in 2012), and can be used with no effort.)

There is not much discussion of bullets around; see the documentation (https://coq.inria.fr/distrib/8.6/refman/Reference-Manual009.html#sec326) in the Coq manual. I learned a lot from Arthur Azevedo de Amorim's Bullets.v (https://github.com/arthuraa/poleiro/blob/master/theories/Bullets.v) file.

Finally, some people don't use bullets, because they systematically use so much automation that they never see subgoals — each lemma has a one-line proof. This is also a valid style. (I have been going to Adam Chlipala's Formal Reasoning about Programs (https://frap.csail.mit.edu/main) 2017 class, where Adam ignores bullets because that is his usual style.) Because I am not crushy enough to do this from the start, my proofs tend to start with cases and subgoals, and then I refine them to add more automation for robustness. I found bullets very useful for the first step, and during the refinement process.

Bullets

Bullets are actually a combination of two features, braces `{ ... }` and actual list bullets — `-`, `+`, `*`, or homogeneous repetitions of those, for example `--` or `***`.

Braces

The opening brace `{` focuses the proof on the current subgoal. If you finish the proof of the subgoal, the following subgoal will not become accessible automatically; you have to use the closing brace `}` first. (If you finish the goal

earlier than you think, you get an error.) Conversely, `}` fails if the subgoal is not complete. (If you fail to finish, you get an error.)

The previous example can thus be written as follows, and will be more robust:

```
induction foo. (* this creates many subgoal *)
{
  proof of first subgoal.
}
{
  proof of second subgoal.
}
```

If you also want to make sure that an error occurs if the number of subgoals changes (for example if new constructors are added to the inductive type of `foo`), you can use an outer layer of braces:

```
{ induction foo. (* this creates many subgoal *)
  {
    proof of first subgoal.
  }
  {
    proof of second subgoal.
  }
} (* would fail if a new subgoal appeared *)
```

List bullets

A bullet, for example `--`, also focuses on the next subgoal. The difference is that when the subgoal is finished, you do not have a closing construction, you must use the same bullet to move to the next subgoal. (Again, this fails if the first proof step changes to prove too much or too little.) With bullets you would write

```
induction foo. (* this creates many subgoal *)
+ proof of first subgoal.
+ proof of second subgoal.
```

Bullets can be nested, but you must use different bullets for the different nesting levels. For example, if this proof is only one subgoal of a larger proof, you can use:

```
- induction foo. (* this creates many subgoal *)
  + proof of first subgoal.
  + proof of second subgoal.
- (* would fail if a new subgoal appeared *)
  rest of the proof
```

The natural ordering of bullets, I think, is by increasing number of lines: `-`, `+` then `*` (and then multi-character bullets, I guess). You can also mix bullets with braces: the opening brace resets the bullet scope, any bullet can be used again with the subgoal.

This gives a large space of freedom in how you want to use these features. You can use only braces, only bullets, braces and only one level of bullets, etc. My own style evolved with experience using the feature, and I will present the current status below.

My current bullet style

When deciding how to use bullets, one distinguishes the commands that preserve the number of subgoals and those that may create new subgoals. I use some additional distinctions.

Some tactics, for example `assert`, create a number of subgoals that is *statically* known, always the same for the tactic. I then use braces around each sub-proof, except the last one, which I think of as the “rest” of the current proof.

```
assert foo as H.
{ proof of foo. }
rest of the proof using H:foo.
```

(If the proof of `foo` takes several lines, I two-indent them, with the braces alone on their lines.)

Most tactics create a *dynamic* number of subgoals, that depends on the specifics of the objects being operated on; this is the case of `case`, `destruct`, `induction` for example. In this case, I open a brace before the tactic, and use a bullet for each subgoal.

```
{ induction foo; simpl; auto.
- proof of first remaining subgoal.
- proof of second remaining subgoal.
  rest of the proof of the second subgoal.
}
```

(Notice that the subgoal-creating step is vertically aligned with the proof steps: I use both braces and bullets, but take only one indentation level each time.)

As an exception, I may omit the braces if we are at the toplevel of the proof (`Proof .. Qed` serve as braces).

Note that omitting the braces here and using different bullets when you nest is also just fine. In my experience it gives proofs that are a bit more pleasant to read but also a bit more cumbersome to edit and move around.

Finally, a not-uncommon mode of use of “dynamic” tactics in the sense above is to expect all the cases, except one, to be discharged by direct automation (for example they are all absurd except one). When it is my intent that all cases but one be discharged (and not a coincidence), I express it by not using braces (this command preserves the number of subgoals), but marking the remaining subgoal with a new bullet *without* increasing the indentation level.

```
{ induction foo.
- first subgoal.
- second subgoal.
  case blah; discharge all sub-subgoals but one.
+ remaining sub-subgoal of the second subgoal.
  finish the sub-subgoal.
- third subgoal.
}
```

(This is the only time where I use several bullet levels.)

If you are the kind of programmer that is passionate about indentation style, I should now have tricked you to use bullets to propose a different variant. Otherwise, please consider using bullets anyway, for example by following the style above, it will make your life easier in the face of changing developments.

← *Linear Types for Low-level Languages* (/blog/2017/02/28/linear-types-for-low-level-languages/)

Datalog for Static Analysis → (/blog/2017/02/21/datalog-for-static-analysis/)
