A Profiler for Ltac
CoqPL Workshop 2015

Tobias Tebbi and Jason Gross

We present a simple profiler for the Ltac tactic language of the Coq Proof Assistant. It measures the time spent in invocations of primitive tactics as well as tactics defined in Ltac and their inner invocations. The profiler is available at [www.ps.uni-saarland.de/ttebbi/ltacprof/](http://www.ps.uni-saarland.de/ttebbi/ltacprof/). The primary use is the development of complex tactics, which tend to be so slow as to impede interactive usage. The reasons for the performance degradation can be intricate, like a slowly performing Ltac match or a sub-tactic whose performance only degrades in certain situations. The profiler generates a call tree and indicates the time spent in a tactic depending on its calling context. Thus it allows locating the part of a tactic definition that contains the performance bug.

Let us consider an example. Suppose we want to solve the following large goal.

\[
\forall x_1, x_1 = 0 \rightarrow \forall x_2, x_2 = x_1 \rightarrow \ldots
\]
\[
\ldots \forall x_{200}, x_{200} = x_{199} \rightarrow x_{200} = 0.
\]

Let us compare the following two automation tactics. The first one rewrites naively, while the second one always picks the right equation with a custom Ltac match.

Ltac arew1 := repeat match goal with
  [H : _ \vdash _] ⇒ rewrite H
end; reflexivity.

Ltac arew2 := repeat match goal with
  [H : (?x = _) \vdash ?x = _] ⇒ rewrite H
end; reflexivity.

The profiler is controlled using Vernacular commands. So we can have an interactive session with the following commands. The Coq command **Restart** is just used to prove the same goal twice.

Start Profiling.
simpl. intros. arew1. reflexivity.
Restart.
simpl. intros. arew2. reflexivity.
Show Profile.

\[1\]To create this goal, just write `Goal imps 200 0.` with the definition

Fixpoint imps n x := match n with
  0 ⇒ x = 0 | S n' ⇒ ∀ y, y = x → imps n' y end.
The last command prints the following result. The first table accumulates over all
invocations, while the second table differentiates according to call tree location. Tactics
that need less than 2% of the time are not displayed.

<table>
<thead>
<tr>
<th>tactic</th>
<th>self</th>
<th>total</th>
<th>calls</th>
<th>max</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>21.4%</td>
<td>85.9%</td>
<td>1</td>
<td>19.980s</td>
</tr>
<tr>
<td>arew1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>rewrite</td>
<td>68.2%</td>
<td>68.2%</td>
<td>40600</td>
<td>0.180s</td>
</tr>
<tr>
<td>H</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>arew2</td>
<td>5.3%</td>
<td>9.1%</td>
<td>1</td>
<td>2.108s</td>
</tr>
<tr>
<td>intros</td>
<td>4.9%</td>
<td>4.9%</td>
<td>2</td>
<td>0.588s</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>tactic</th>
<th>self</th>
<th>total</th>
<th>calls</th>
<th>max</th>
</tr>
</thead>
<tbody>
<tr>
<td>arew1</td>
<td>21.4%</td>
<td>85.9%</td>
<td>1</td>
<td>19.980s</td>
</tr>
<tr>
<td>¬ rewrite H</td>
<td>64.5%</td>
<td>64.5%</td>
<td>40400</td>
<td>0.008s</td>
</tr>
<tr>
<td>arew2</td>
<td>5.3%</td>
<td>9.1%</td>
<td>1</td>
<td>2.108s</td>
</tr>
<tr>
<td>¬ rewrite H</td>
<td>3.7%</td>
<td>3.7%</td>
<td>200</td>
<td>0.180s</td>
</tr>
<tr>
<td>intros</td>
<td>4.9%</td>
<td>4.9%</td>
<td>2</td>
<td>0.588s</td>
</tr>
</tbody>
</table>

Note that arew1 is much slower than arew2, but spends a smaller portion of its time
with the Ltac match (21.4% out of 85.9% instead of 5.3% out of 9.1%). So the complex
match pattern in arew2 was not for free. Furthermore, the time spend in intros is
surprisingly high.

We have implemented the profiler as a patch to the Coq sources, since we needed to
instrument the tactic engine. The patch currently only works with Coq 8.4, since the
updated tactic engine in the upcoming Coq 8.5 affected the error backtrace functionality
of Ltac, which we instrument. We would like to have the instrumentation incorporated
as hooks in the Coq trunk, allowing for the profiler to be developed as a plugin.

In the future, we would like to address the following missing features:

- A user-friendly and configurable interface to visualize the results. Especially, it
  should be possible to aggregate selected parts of the call tree.
- A reusable output format like CSV.
- Aggregated results over multiple files.

For the workshop, we propose to do a live demonstration of the profiler, presenting
possible application cases. Afterwards, we would like to discuss use cases, profiling
experiences and feature requests.