Towards Proof Refactoring

Iain Whiteside
Mathematical Reasoning Group
University of Edinburgh

May 20, 2010

Outline of talk

- Motivate the need for refactoring of proof scripts;
- Outline the approach;
- Describe simple proof script language and semantics;
- Show how we propose to define refactorings and prove correct.

Refactoring

A ***proof refactoring*** is a proof script transformation that supports the design, evolution, and reuse of a proof script while still preserving the semantics. A refactoring may be pervasive, but it should be routine.

Why is it necessary?

- In the paper on his proof of the Four Colour Theorem, Gonthier speaks of ***several months*** spent refactoring.
- Refactoring provides a structured way to make definitional changes.
- With sophisticated tactics, even swapping independent lemmas can break a proof.

Motivation

Consider the following Isabelle script:

```
abbreviation sym_diff :: "'a set ⇒ 'a set ⇒ 'a set" (infixl "SymDiff" 70) where
  "A SymDiff B ≡ (A Un B) - (A Int B)"

notation (xsymbols)
  sym_diff (infixl "△" 70)

lemma lemma1 [simp]: "A △ B = (A - B) Un (B -A)"
  by blast

lemma lemma2 [simp]:
  assumes "a ∈ A"
  shows "A △ {a} = A - {a}"
proof(subst lemma1)
  have "{a} - A = {}" using assms by blast
  from this show " A - {a} ∪ ({a} - A) = A - {a}"  by blast
qed
```

Motivation 2

```
lemma sym_diff_union_diff [simp]: "A △ B = (A - B) Un (B - A)"
   by blast

lemma sym_diff_singleton [simp]:
  assumes "a ∈ A"
  shows "A △ {a} = A - {a}"
proof (subst sym_diff_union_diff)
  have "{a} - A = {}" using assms by blast
  from this show " A - {a} ∪ ({a} - A) = A - {a}"  by blast
qed
```

Our approach

Some of the key features:

- **Generic:** community is small and fragmented: users often stick to their 'first love'.
- **Correct:** must not inadvertently change the meaning of a lemma.
- **Iterative:** refactoring is difficult. We start with a simple model of proof scripts.
- **Compositional:** can all refactorings be built from a set of primitives?

Proof scripts

We give a very simple proof script language, consisting only tactics and lemmas:

$proofscript$	::=	$(tacdef \mid lemma)$ (; $proofscript)^*$	lemmas and tactics
$lemma$	::=	**lemma** $name$: $goal$ **proof** prf **qed**	lemmas
$tacdef$	::=	**tacdef** $tacname := t$	tactic definition

Here t and prf are the tactic language and the proof language.

Evaluation semantics

We introduce a *proof script context*: (T, L) where:

- T is a set of pairs (t_{name}, t): tactic names and definitions;
- L is a set of triples (l_{name}, γ, prf): lemma names, goal statements and proof.

We define an evaluation relation $\langle (T, L), proofscript \rangle \Downarrow \langle (T', L') \rangle$:

$$\frac{isWellFormed(t) \qquad t_{name} \notin getTacticNames(T)}{\langle (T, L), \textbf{tacdef}\ tacname := t \rangle \Downarrow \langle (T \cup \{(tacname, t)\}, L) \rangle}\ (\textsc{B-Scr-Tactic})$$

$$\frac{\langle \gamma, prf \rangle \Downarrow_p \langle s, [] \rangle \qquad name \notin getLemmaNames(L)}{\langle (T, L), \textbf{lemma}\ name\textbf{:}\ \gamma\ \textbf{proof}\ prf\ \textbf{qed} \rangle \Downarrow \langle (T, L \cup \{(name, \gamma, prf)\}) \rangle}\ (\textsc{B-Scr-Lemma})$$

Refactoring formalised

Using the formalism for proof scripts, we can define a refactoring R as a map:

$$R : proofscript \rightarrow proofscript$$

with the property that, if $R(ps_1) = ps_2$ and

$$\langle (\{\}, \{\}), ps_1 \rangle \Downarrow \langle (T, L) \rangle$$

and

$$\langle (\{\}, \{\}), ps_2 \rangle \Downarrow \langle (T', L') \rangle$$

then

$$\forall \gamma \in getgoals(L).\ \exists l \in L'.\ getgoal(l) = \gamma$$

That is, at least the same things are proved.

Example: renaming a lemma

We define transformations based on the proof script objects. Suppose we want to rename *lemma1* to *sym_diff_union_diff* (as in example):

$$\frac{t' := \mathit{rename\_lemma\_in\_tac}(\mathit{old}, \mathit{new}, t)}{\langle \textbf{tacdef}\ \mathit{tacname} := t \rangle \Rightarrow \langle \textbf{tacdef}\ \mathit{tacname} := t' \rangle}$$

$$\langle \textbf{lemma}\ \mathit{lemma1} \textbf{:}\ \gamma\ \textbf{proof}\ \mathit{prf}\ \ \textbf{qed} \rangle \Rightarrow \langle \textbf{lemma}\ \mathit{sym\_diff\_union\_diff} \textbf{:}\ \gamma\ \textbf{proof}\ \mathit{prf}\ \ \textbf{qed} \rangle$$

$$\frac{\mathit{prf}' := \mathit{rename\_lemma\_in\_lemma}(\mathit{old}, \mathit{new}, \mathit{prf})}{\langle \textbf{lemma}\ \mathit{name} \textbf{:}\ \gamma\ \textbf{proof}\ \mathit{prf}\ \ \textbf{qed} \rangle \Rightarrow \langle \textbf{lemma}\ \mathit{name} \textbf{:}\ \gamma\ \textbf{proof}\ \mathit{prf}'\ \textbf{qed} \rangle}$$

We also have preconditions, which I haven't talked about: new name must be fresh (and some others).

An outline proof

In order to prove correctness:

- We show all the refactored tactics are well formed;
- We show that the refactored lemmas are all still proved;
- We can then show the same context (except with different prfs) has been produced.

Summary

We have:

- Motivated the requirement for proof refactoring;
- Described a simple proof script;
- Shown how refactorings can be defined and correctness stated.

Work supported by Microsoft Research

Any Questions?