

1 Announcements

Reading assignment for this part of the course: Chapters 8 and 9 of the book.

Informal homework: use ACL2 to prove all of the theorems we've seen in class.

2 Review

Last time we started discussing the internals of the theorem prover, and in particular rewriting. We did that by using the `rev-rev` example.

We saw that the proof ACL2 came up with was not the same as the proof we came up with.

We embarked on the exercise to mimic the proof we gave in class and that led to a discussion about how ACL2 uses lemmas, which led to a discussion of the rewriter.

We saw that ACL2 uses lemmas (theorems) as rewrite rules. Rewrite rules are oriented, i.e., they are applied in one direction. We saw that theorems of the form `(implies ... (equal (foo ...) ...))` are used to rewrite occurrences of `(foo ...)` and that rules are checked in reverse-chronological order (last first) until one that matches is found. If the rule has hypotheses, then we backchain, trying to discharge those hypotheses. If we do, we apply it.

We also saw that ACL2 rewrites in an inside-out manner. This forces to think about how to write theorems so that ACL2 can make effective use of them. One general idea is to put the more complicated part of the equality on the left-hand side. Another is that we should aim to use rewrite rules to drive expressions into a “canonical” form.

Another thing we considered is what to do when proof attempts fail. The rough idea is to look at checkpoints and to use our model of how ACL2 works to come up with a plan that will allow ACL2 to make progress. More on that today.

3 Using ACL2

Let us now try the `rev-rev` example.

Last time we saw that we needed the following lemmas:

```
(defthm true-listp-rev
  (true-listp (rev x)))
```

```
(defthm app-rev
  (equal (rev (app a b))
```

```
(app (rev b) (rev a)))
```

When we try to prove `true-listp-rev`, we notice that ACL2 does an extra induction (*1.1), so we have to see why and to figure out what lemma to prove.

We decide on the following lemma. Why?

```
(defthm true-listp-app
  (equal (true-listp (app a b))
         (true-listp b)))
```

Now `true-listp-rev` does not require extra inductions.

However, `app-rev` fails. Looking at the checkpoint, we see that we need the following lemma.

```
(defthm app-truelistp
  (implies (true-listp x)
           (equal (app x nil)
                  x)))
```

Now, the proof of `app-rev` goes through, but there is an extra induction, so we check to find that the following lemma is suggested.

```
(defthm app-associative
  (equal (app (app a b) c)
         (app a (app b c))))
```

We add it and now the proof of `app-rev` goes through with no extra inductions.

The proof of `rev-rev` also goes through with no inductions. Great.

```
(defthm rev-rev
  (implies (true-listp x)
           (equal (rev (rev x)) x)))
```

Now, let's try proving the following:

```
(defthm trip-rev
  (equal (rev (rev (rev x)))
         (rev x)))
```

It goes through with no problems. What if we just used ACL2's proof of `rev-rev`? It doesn't work. Looking at the failed proof, we see that the lemmas we proved above are needed for the proof.

Now, let's consider a different version `rev`, one that is more efficient, but also more complicated.

```
(defun revt (x acc)
  (if (endp x)
      acc
      (revt (cdr x) (cons (car x) acc))))
```

```
(defun rev2 (x)
  (revt x nil))
```

We would like to work with `rev2` since it is more efficient. But, then we are going to have to reason about it too.

What can we do?

Well, we can prove that `rev2` is `rev`. Here we go.

```
(defthm rev2-is-rev
  (equal (rev2 x)
         (rev x)))
```

But this needs what should be an obvious lemma:

```
(defthm revt-rev
  (equal (revt x acc)
         (app (rev x) acc)))
```

Now, notice that we don't need to worry about `rev2` anymore. Because every occurrence of `(rev2 ...)` will be rewritten to `(rev ...)`! So, all of the theorems we proved about `rev` apply here directly, and we can prove any theorem about `rev2` that we can prove about `rev`. For example, consider the proof of:

```
(thm (implies (true-listp x)
              (equal (rev2 (rev2 x)) x)))
```

Also, when we execute, we execute the efficient version of the code.

So, this is an example of step-wise refinement. The idea is to come up with as simple (and probably inefficient) solution as possible. To prove theorems about that system and to then replace components with equivalent components.