Hint Orchestration Using ACL2's Simplifier

Sol Swords
Centaur Technology, Inc.
ACL2 Workshop 2018

This talk is for hint abusers

This might be you if:

- You can’t be bothered to figure out a good rewriting strategy
- You just don’t know the right lemma to prove
- Your proofs are all done by luck and hackery
- You are a bad ACL2 user

Awful Hints

A Short Compendium of Common Abominations

Awful Hints: The Unstable Subgoal

```
:hints (("Goal" :induct t)
        ("Subgoal *1/2"
         "Subgoal *1/2.1"
         "Subgoal *1/2.2"
         "Subgoal *1/1.2"
         "Subgoal *1/1.3.2" ...))
```

Awful Hints: The Unreliable :expand

```
:hints (("Goal" :induct (foo x y z)
                :expand ((foo x   y z)
                         (foo nil y z)
                         (foo t   y z))))
```

Awful Hints: The Unwieldy :use

```
:use ((:instance my-lemma
       (a (MV-NTH 0 (FOOBAR X
                            (MV-NTH 1 (BIZBAZ-WITNESS X Z))
                            (BAR (BUZ Y) Z))))
       (b (MV-NTH 2 (FOOBAR X
                            (MV-NTH 1 (BIZBAZ-WITNESS X Z))
                            (BAR (BUZ Y) Z))))))
```

Awful Hints: The Untypable Translated Term

```
(and stable-under-simplificationp
     (member-equal '(not (equal (tag$inline x) ':g-call)) clause)
     '(...))
```

use-termhint

Solution to all these problems and more

use-termhint

~~Solution to all these problems and more~~

Hack that works around some of these problems

use-termhint

~~Solution to all these problems and more~~

Hack that works around some of these problems

(if you can't be bothered to do things the RIGHT way.)

How to use `use-termhint`

- Write a term that produces the hints you want in the cases you want
  - There are a few nifty features to be aware of
- Give a hint (`use-termhint my-term`)

- That's it

Offline Demo

```
(defun-sk nat-list-bounded-by-x (x y)
  (forall z
          (implies (member (nfix z) y)
                   (<= (nfix z) (nfix x)))))

(in-theory (disable nat-list-bounded-by-x
                    nat-list-bounded-by-x-necc))

(defthm nat-list-bounded-by-x-of-nfix
  (equal (nat-list-bounded-by-x (nfix x) y)
         (nat-list-bounded-by-x x y))
  :hints (...))
```

Handwavy Hand Proof

Two cases:
→: `(implies (nat-list-bounded-by-x (nfix x) y) (nat-list-bounded-by-x x y))`
←: `(implies (nat-list-bounded-by-x x y) (nat-list-bounded-by-x (nfix x) y))`

→: assume `(not (nat-list-bounded-by-x x y))`, expand it to get a witness z such that `(member (nfix z) y)` and `(not (<= (nfix z) (nfix x)))`. Then this implies `(not (nat-list-bounded-by-x (nfix x) y))` by `nat-list-bounded-by-x-necc`, since `(nfix (nfix x)) = (nfix x)`. □

←: same, swapping `(nfix x)` and `x`. □

Without `use-termhint`

```
(defthm nat-list-bounded-by-x-of-nfix
  (equal (nat-list-bounded-by-x (nfix x) y)
         (nat-list-bounded-by-x x y))
  :hints (("goal"
           :use ((:instance nat-list-bounded-by-x-necc
                  (z (nat-list-bounded-by-x-witness (nfix x) y))
                  (x x))
                 (:instance nat-list-bounded-by-x-necc
                  (z (nat-list-bounded-by-x-witness x y))
                  (x (nfix x))))
           :in-theory (enable nat-list-bounded-by-x))))
```

Make it a challenge?

- Break the proof into the two natural cases →, ←
- Only :use each instance in the case where it's needed
- Don't enable `nat-list-bounded-by-x`, expand where needed

  Artificial handicap for a small example, but practical for more complicated/expensive proofs…

  Also produces a proof that's easier to follow (if anyone cares).

Without `use-termhint`

```
(defthm nat-list-bounded-by-x-of-nfix
  (equal (nat-list-bounded-by-x (nfix x) y)
         (nat-list-bounded-by-x x y))
  :hints (("goal" :cases ((nat-list-bounded-by-x (nfix x) y)))
          (and stable-under-simplificationp
               (let ((lit (assoc 'nat-list-bounded-by-x clause)))
                 `(:expand (,lit)
                   :use ((:instance nat-list-bounded-by-x-necc
                          (z (nat-list-bounded-by-x-witness . ,(cdr lit)))
                          (x ,(if (eq (second lit) 'x) '(nfix x) 'x)))))))))
```

With `use-termhint`

```
(defthm nat-list-bounded-by-x-of-nfix
  (equal (nat-list-bounded-by-x (nfix x) y)
         (nat-list-bounded-by-x x y))
  :hints ((use-termhint
           (b* (((mv bounding-x other-x)
                 (if (nat-list-bounded-by-x (nfix x) y)
                     (mv (nfix x) x)  ;; →
                   (mv x (nfix x))))  ;; ←
                (witness (nat-list-bounded-by-x-witness other-x y)))
             `(:expand ((nat-list-bounded-by-x ,(hq other-x) y))
               :use ((:instance nat-list-bounded-by-x-necc
                      (x ,(hq bounding-x))
                      (z ,(hq witness)))))))))
```

Comparison

- Termhint version is a little longer, but just because I chose a bad example
- Termhint version is in the “object language” -- same kind of term as the goal itself
- Non-termhint version is in the “meta language” -- analyzing the representation of the goal
- Termhint version kind of describes how the proof works
- Non-termhint version says what to do based on the syntax of the clause.

- What is that HQ thing?
  - Stands for Hint Quote
  - Just some function
  - We treat it like QUOTE when we want to -- more later

```
Goal'
(IMPLIES
 (USE-TERMHINT-HYP
    (MV-LET (BOUNDING-X OTHER-X)
            (IF (NAT-LIST-BOUNDED-BY-X (NFIX X) Y)
                (LIST (NFIX X) X)
                (LIST X (NFIX X)))
            (LET ((WITNESS (NAT-LIST-BOUNDED-BY-X-WITNESS OTHER-X Y)))
                 (LIST :EXPAND (LIST (LIST 'NAT-LIST-BOUNDED-BY-X
                                           (HQ OTHER-X)
                                           'Y))
                       :USE (LIST (LIST :INSTANCE 'NAT-LIST-BOUNDED-BY-X-NECC
                                        (LIST 'X (HQ BOUNDING-X))
                                        (LIST 'Z (HQ WITNESS)))))))))
 (EQUAL (NAT-LIST-BOUNDED-BY-X (NFIX X) Y)
        (NAT-LIST-BOUNDED-BY-X X Y))).
```

```
Subgoal 2'
(IMPLIES
 (AND
  (NAT-LIST-BOUNDED-BY-X (NFIX X) Y)
  (USE-TERMHINT-HYP
   (LIST
        :EXPAND (LIST (LIST* 'NAT-LIST-BOUNDED-BY-X
                             (HQ X)
                             '(Y)))
        :USE (LIST (LIST :INSTANCE 'NAT-LIST-BOUNDED-BY-X-NECC
                         (LIST 'X (HQ (NFIX X)))
                         (LIST 'Z
                               (HQ (NAT-LIST-BOUNDED-BY-X-WITNESS X Y))))))))
 (NAT-LIST-BOUNDED-BY-X X Y)).
```

```
   (LIST
        :EXPAND (LIST (LIST* 'NAT-LIST-BOUNDED-BY-X
                             (HQ X)
                             '(Y)))
        :USE (LIST (LIST (LIST :INSTANCE 'NAT-LIST-BOUNDED-BY-X-NECC
                               (LIST 'X (HQ (NFIX X)))
                               (LIST 'Z
                                     (HQ (NAT-LIST-BOUNDED-BY-X-WITNESS X Y))))))
```

After replacing HQ with QUOTE, this evaluates to:

```
(:EXPAND ((NAT-LIST-BOUNDED-BY-X X Y))
         :USE ((:INSTANCE NAT-LIST-BOUNDED-BY-X-NECC (X (NFIX X))
                          (Z (NAT-LIST-BOUNDED-BY-X-WITNESS X Y)))))
```

```
[Note:  A hint was supplied for our processing of the goal below.
Thanks!]

Subgoal 2''
(IMPLIES (NAT-LIST-BOUNDED-BY-X (NFIX X) Y)
         (NAT-LIST-BOUNDED-BY-X X Y)).

We augment the goal with the hypothesis provided by the :USE hint.
The hypothesis can be derived from NAT-LIST-BOUNDED-BY-X-NECC via instantiatio\
n.  We are left with the following subgoal.
...
Subgoal 2'''...
Subgoal 2'4'...

But simplification reduces this to T, using the :definitions
NAT-LIST-BOUNDED-BY-X and NOT, the :executable-counterpart of NOT and
the :type-prescription rule NAT-LIST-BOUNDED-BY-X.
```

Alternatives to Awful Hints

Alternatives: The Unstable Subgoal

```
:hints (("Goal" :induct t)
        ("Subgoal *1/2"
         "Subgoal *1/2.1"
         "Subgoal *1/2.2"
         "Subgoal *1/1.2"
         "Subgoal *1/1.3.2" ...))
```

- `Use-termhint` lets you pick the case in which your hint applies via `if` tests in your term -- no subgoal numbers.

Alternatives: The Unreliable :expand

```
:hints (("Goal" :induct (foo x y z)
                :expand ((foo x   y z)
                         (foo nil y z)
                         (foo t   y z))))

:hints (("Goal" :induct (foo x y z))
        (use-termhint `(:expand ((foo ,(hq x) ,(hq y) ,(hq z))))))
```

- The x in the hint term is simplified similar to the x in the call of `foo`

Alternatives: The Unwieldy :use

```
:use ((:instance my-lemma
       (a (MV-NTH 0 (FOOBAR X
                            (MV-NTH 1 (BIZBAZ-WITNESS X Z))
                            (BAR (BUZ Y) Z))))
       (b (MV-NTH 2 (FOOBAR X
                            (MV-NTH 1 (BIZBAZ-WITNESS X Z))
                            (BAR (BUZ Y) Z))))))
```

```
((use-termhint
  (b* (((mv ?biz baz) (bizbaz-witness x z))
       ((mv a ?b c) (foobar x biz (bar (buz y) z))))
    `(:use ((:instance my-lemma (a ,(hq a)) (b ,(hq c))))))))
```

Alternatives: The Untypable Translated Term

```
(and stable-under-simplificationp
     (member-equal '(not (equal (tag$inline x) ':g-call)) clause)
     '(...))

(use-termhint
  (and (eq (tag x) :g-call)
       `(...)))
```

- Choice of case via case splitting rather than clause membership
- Never need to deal with translated term syntax
- Object language, not metalanguage

Conclusion

When you have to use hints, use-termhint

Solves a few pernicious problems with hints:

- Triggers use of a hint on a particular assumption, not a subgoal number or syntactic property
- Allows binding variables & using those variables in hints to avoid term blowup and stay DRY
- Hint term is simplified so it doesn't need to start in normal form for things like `:expand`
- Never need write a translated term.

Would be nice

- Induct + provide hints for various cases by writing a recursive function that produces hint terms
- Provide hints for goals created by processes other than case splitting
  - Functionally instantiate a theorem and provide hints for functional-instance obligations
  - Instantiate `(:theorem (foo (bar x)))` and give a hint for the proof of `(foo (bar x))`
  - Call a clause processor and give hints for its generated subgoals
- Give hints when not stable-under-simplification

I don't see how to do these by building on `use-termhint` (but prove me wrong!)