Generating Mutually Inductive Theorems From Concise Descriptions

Sol Swords
Centaur Technology, Inc.

ACL2 Workshop 2020

Paper: http://acl2-2020.info/papers/generating-mutually-inductive-theorems.pdf

Proofs about Mutual Recursions Aren't Hard

What gets in the way?

- Induction schemes aren't provided by ACL2 as with singly-recursive functions
  - Needs some extra work, but this can be automated
- Usually need a (slightly different) theorem about each function in the clique
  - Tedious to list all the theorems by hand when the clique is large

Contribution: `defret-mutual-generate`

- Generates a mutually-inductive clique of theorems from a set of rules
- Rules based on info recorded by `defines`: input/output names and types
- Wraps around existing macro `defret-mutual`
    - Which itself wraps flag defthm macros generated by `make-flag`

Case Study: FGL Rewriter

- 49-function clique defined in centaur/fgl/interp.lisp
- 22 sets of theorems generated using `defret-mutual-generate`
  - 17 mutual inductions, 5 per-function corollaries
- Average 41 lines per `defret-mutual-generate` form
  - Dominated by one 430-line form
  - Average 23 lines omitting the one outlier
  - The simplest of these produce ~300 lines of defthm forms (not counting local helper events).
- Keeps DRY and maintainable
  - Most changes to rewriter require only small changes to few of the theorem forms.

Quick background: `make-flag`

Generates a *flag function* from a mutual recursion for use as an induction scheme

```
(defun-nx subst-term-flag (flag x alist)
  (cond ((equal flag 'subst-term)
         (cond …
               (t (cons (car x)
                        (subst-term-flag 'subst-termlist (cdr x) alist)))))
        (t ;; (equal flag 'subst-termlist)
         (if (atom x)
             nil
           (cons (subst-term-flag 'subst-term (car x) alist)
                 (subst-term-flag 'subst-termlist (cdr x) alist))))))
```

Quick background: `make-flag`

Also a *flag defthm macro*:

```
(defthm-subst-term-flag
  (defthm ev-term-of-subst-term
    (equal (ev-term (subst-term x alist) env)
           (ev-term x (ev-alist alist env)))
    :flag subst-term)
  (defthm ev-termlist-of-subst-termlist
    (equal (ev-termlist (subst-termlist x alist) env)
           (ev-termlist x (ev-alist alist env)))
    :flag subst-termlist))
```

Quick background: `make-flag`

```
(defthm flag-lemma-for-ev-term-of-subst-term
  (cond ((equal flag 'subst-term)
         (equal (ev-term (subst-term x alist) env)
                (ev-term x (ev-alist alist env))))
        (t ;; subst-termlist
         (equal (ev-termlist (subst-termlist x alist) env)
                (ev-termlist x (ev-alist alist env)))))
  :hints (("goal" :induct (subst-term-flag flag x alist))
          …)
  :rule-classes nil)
```

Quick background: `define`, `defines`

Define/defines are like defun/mutual-recursion but allow specifying/storing some extra info. Relevant to us: types of formals, types and names of return values

```
(defines fgl-interp
  (define fgl-interp-term ((x pseudo-termp)
                           (interp-st interp-st-bfrs-ok)
                           state)
    :returns (mv
              (xobj fgl-object-p)
              new-interp-st new-state)
    ...)
  ...)
```

Quick background: `defret`, `defret-mutual`

Mostly like `defthm` but creates hidden bindings of return names to their values

```
(defret interp-st-scratch-isomorphic-of-<fn>
  (interp-st-scratch-isomorphic new-interp-st (double-rewrite interp-st))
  :fn fgl-interp-term)
→
(defthm interp-st-scratch-isomorphic-of-fgl-interp-term
  (b* (((mv ?xobj ?new-interp-st ?new-state)
        (fgl-interp-term x interp-st state)))
    (interp-st-scratch-isomorphic new-interp-st
                                  (double-rewrite interp-st))))
```

`Defret-mutual-generate`: Minimal example

```
(std::defret-mutual-generate interp-st-scratch-isomorphic-of-<fn>
  :return-concls ((new-interp-st
                   (interp-st-scratch-isomorphic
                    new-interp-st
                    (double-rewrite interp-st))))
  :hints ((fgl-interp-default-hint 'fgl-interp-term id nil world))
  :mutual-recursion fgl-interp)
```

Rules

Rules take the form condition → action, e.g.:

(*condition*) If a function has a return value named `new-interp-st`, then

(*action*) Add the following expression as a conclusion.

Example conditions

- Function's name is `foo`
- Function has a formal of type `fancy-objtype`
- Function has a return value named `blob`

Example actions

- Add `term` as a conclusion or hypothesis
- Add `b*` bindings around the hypotheses and conclusion
- For each formal of type `integerp`, add hypothesis `(natp x)` where `x` is the formal name
- For each return value of type `stringp`, add conclusion `(< (length x) 5)` where `x` is the return value name
- Add a keyword to the defthm form
- Set the the theorem name to the given template

Shortcuts

- `:formal-hyps` generates hypotheses for formals of the given name or type
- `:return-concls` generates conclusions for return values of the given name or type

```
(std::defret-mutual-generate interp-st-bfrs-ok-of-<fn>
  :rules
  ((t (:add-bindings
       ((?new-logicman (interp-st->logicman new-interp-st))
        (?logicman (interp-st->logicman interp-st)))))
   ((or (:fnname fgl-rewrite-try-rules)
        (:fnname fgl-rewrite-try-rule)
        (:fnname fgl-rewrite-try-rewrite)
        (:fnname fgl-rewrite-try-meta)
        (:fnname fgl-rewrite-binder-try-rules)
        (:fnname fgl-rewrite-binder-try-rule)
        (:fnname fgl-rewrite-binder-try-rewrite)
        (:fnname fgl-rewrite-binder-try-meta)
        (:fnname fgl-rewrite-try-rules3))
    (:add-hyp (scratchobj-case (stack$a-top-scratch (double-rewrite (interp-st->stack interp-st)))
                               :fgl-objlist))))
  :formal-hyps    ;; generates hypotheses
  (((interp-st-bfr-p x)           (lbfr-p x logicman))
   ((fgl-object-p x)              (lbfr-listp (fgl-object-bfrlist x) logicman))
   ((fgl-objectlist-p x)          (lbfr-listp (fgl-objectlist-bfrlist x) logicman))
   ((fgl-object-bindings-p x)     (lbfr-listp (fgl-object-bindings-bfrlist x) logicman))
   (interp-st                     (interp-st-bfrs-ok interp-st))
   ((constraint-instancelist-p x) (lbfr-listp (constraint-instancelist-bfrlist x) logicman)))
  :return-concls  ;; generates conclusions
  ((xbfr                          (lbfr-p xbfr new-logicman))
   ((fgl-object-p x)              (lbfr-listp (fgl-object-bfrlist x) new-logicman))
   ((fgl-objectlist-p x)          (lbfr-listp (fgl-objectlist-bfrlist x) new-logicman))
   (new-interp-st                 (interp-st-bfrs-ok new-interp-st)))
  :hints ((fgl-interp-default-hint 'fgl-interp-term id nil world))
  :mutual-recursion fgl-interp)
```

Future possibilities

- Apply same idea to automate theorems on sets of functions that are not all mutually recursive
- Allow annotations of formals and returns and recognize them in rule conditions

Conclusion

- Greatly reduces the size of forms, amount of editing for proving theorems about large cliques.
- Low requirements: use defines, add formal types, return value names/types.
- Documentation: std::defret-mutual-generate
- Mutual recursions are fine! Even big ones. No need to avoid them.
    - Big cliques are preferable to big functions.