Meta-extract: Using Existing Facts in Meta-reasoning

Matt Kaufmann (UT Austin)
Sol Swords (Centaur Technology)

ACL2 Workshop 2017

OUTLINE

INTRODUCTION

- ACL2 supports two kinds of user-defined, verified proof routines:
  - `:meta` rule class: `term` → `term`, invoked by the rewriter,
  - `:clause-processor` rule class: `clause` → `clauses`, invoked by hints.
- Previously could extract facts from the world and use built-in proof tools, but could not assume them correct.
- Now (post-2012) these facts/tools may be assumed correct via *meta-extract hypotheses* when proving soundness of metafunctions.
  - ⋆⋆⋆ At run time, a metafunction may use facts that were not available when it was proved correct! ⋆⋆⋆

THIS TALK

- reviews meta reasoning
- gives two simple examples to illustrate meta-extract hypotheses
- discusses a nice shortcut
- summarizes some applications

OUTLINE

REVIEW OF :Meta RULES

Canonical example of a `:meta` rule:
`cancel_plus-equal` (from "books/meta/meta-plus-equal.lisp") cancels like terms from the equality of two sums.

```
ACL2 !>:trans (equal (+ x y x z) (+ x z z z))

(EQUAL (BINARY-+ X (BINARY-+ Y (BINARY-+ X Z)))
       (BINARY-+ X (BINARY-+ Z (BINARY-+ Z Z))))

=> *

ACL2 !>(cancel_plus-equal
        '(EQUAL (BINARY-+ X (BINARY-+ Y (BINARY-+ X Z)))
                (BINARY-+ X (BINARY-+ Z (BINARY-+ Z Z)))))
(EQUAL (BINARY-+ Y X) (BINARY-+ Z Z))
```

Review of :Meta Rules (2)

Key events:

- Define an evaluator:

```
(defevaluator ev-plus-equal ...)
   (ev-plus-equal term alist) --> value
```

- Define the metafunction:

```
(defun cancel_plus-equal (x) ...)
```

- Prove the metafunction correct w.r.t. the evaluator:

```
(defthm cancel_plus-equal-correct
  (equal
   (ev-plus-equal x a)
   (ev-plus-equal (cancel_plus-equal x) a))
  :rule-classes ((:meta :trigger-fns (equal))))
```

Let's see this rule used in a proof.

REVIEW OF :Meta RULES (2)

```
ACL2 !>(include-book "meta/meta-plus-equal" :dir :system)
....
ACL2 !>(trace$ cancel_plus-equal)
 ((CANCEL_PLUS-EQUAL))
ACL2 !>(thm (implies (and (acl2-numberp z)
                          (equal (+ x y x z) (+ x z z z)))
                     (equal z (/ (+ x y) 2)))
            :hints (("Goal" :in-theory (disable (tau-system))))
Goal'
1> (CANCEL_PLUS-EQUAL
     (EQUAL (BINARY-+ X (BINARY-+ X (BINARY-+ Y Z)))
            (BINARY-+ X (BINARY-+ Z (BINARY-+ Z Z)))))
<1 (CANCEL_PLUS-EQUAL (EQUAL (BINARY-+ X Y) (BINARY-+ Z Z)))
....
Proof succeeded.
ACL2 !>
```

OUTLINE

EXAMPLE 1: USING GLOBAL FACTS

Goal: Rewrite stobj `(accessor (updater val foo$))` terms without either:

- proving n^2 individual rules per stobj
- enabling accessors/updaters to expand to `nth`/`update-nth`

An approach: `nth-update-nth-ev-meta-fn` checks that `accessor` is defined as a call of `nth` and `updater` is defined as a call of `update-nth` and rewrites accordingly.

EXAMPLE 1: USING GLOBAL FACTS

- Can look up function definitons from the world.
- But: how can we prove this correct?
- Before meta-extract we'd need to somehow verify that the definitions found in the world were correct
  - E.g., have a hypothesis metafunction that produces the corresponding assumption.
- Meta-extract lets you assume this while proving your metafunction correct.
- Accessor & updater functions don't need to be known by evaluator
  - Can prove it operates correctly even on functions that haven't been defined yet!

EXAMPLE 1: USING GLOBAL FACTS

```
; demos/nth-update-nth-meta-extract.lisp
(defthm nth-update-nth-meta-rule-st
  (implies
   (and (nth-update-nth-ev ; (f (update-g val st))
         (meta-extract-global-fact
          (list :formula (car term)) state)
         (meta-extract-alist term a state))
        ...)
   (equal (nth-update-nth-ev term a)
          (nth-update-nth-ev
           (nth-update-nth-meta-fn term mfc state)
           a)))
  :hints ...
  :rule-classes ((:meta :trigger-fns ...)))
```

Example 1: Meta-extract Hypothesis

`Meta-extract-global-fact:`

- Returns various terms expressing known facts.
- Only produces terms that are known true.
- Meta rule/clause processor theorems are allowed to assume the terms it produces evaluate to true as a special hypothesis.

Part of the definition:

```
(case-match obj
  ((':formula name)
   (meta-extract-formula name st))
  ...)
```

META-EXTRACT-GLOBAL-FACT

Supports:

- Theorem bodies, function definitions, and constraints (`meta-extract-formula`)
- Rewrite rules from functions' `lemmas` properties
- Evaluation of ground function calls (`magic-ev-fncall`).

Outline

EXAMPLE 2: USING CONTEXTS

Consider this metafunction:

```
(defun nth-symbolp-metafn (term mfc state)
  (declare (xargs :stobjs state))
  (case-match term
    (('nth n x)
     (if (equal (mfc-ts n mfc state :forcep nil)
                *ts-symbol*)
         (list 'car x)
       term))
    (& term)))
```

Approximately: "If term is `(nth n x)` and `n` is known to be a symbol in the current context, rewrite term to `(car x)`."

EXAMPLE 2: USING CONTEXTS

- How can we prove this correct?
- Before meta-extract we'd need to somehow verify that `mfc-ts` was "telling the truth"
  - E.g., have a hypothesis metafunction that produces the corresponding assumption.
- Meta-extract lets you assume this while proving your metafunction correct.

EXAMPLE 2: USING CONTEXTS

Correctness theorem for `nth-symbolp-metafn`:

```
; workshops/2017/kaufmann-swords/support/intro.lisp
(defthm nth-symbolp-meta
  (implies
   ;; Meta-extract hypothesis:
   (nthmeta-ev (meta-extract-contextual-fact
                 `(:typeset ,(cadr term))
                 mfc
                 state)
               a)
   ;; Standard meta rule conclusion:
   (equal (nthmeta-ev term a)
          (nthmeta-ev (nth-symbolp-metafn
                        term mfc state)
                       a)))
  :rule-classes ((:meta :trigger-fns (nth))))
```

EXAMPLE 2: META-EXTRACT HYPOTHESIS

`Meta-extract-contextual-fact:`

- Returns various terms expressing facts known under a given context.
- Only produces terms that are known true.
- Meta rule theorems are allowed to assume the terms it produces evaluate to true.

Part of the definition:

```
(case-match obj
  (('(:typeset term . &) ; mfc-ts produces correct result
   `(typespec-check
     ',(mfc-ts term mfc state :forcep nil :ttreep nil)
     ,term))
```

META-EXTRACT-CONTEXTUAL-FACT

Supports:

- Typeset reasoning (`mfc-ts`)
- Rewriting (`mfc-rw`, `mfc-rw+`, `mfc-relieve-hyp`)
- Linear arithmetic (`mfc-ap`)

OUTLINE

A Nice Shortcut

```
(my-evl (meta-extract-contextual-fact obj mfc state) a)
(my-evl (meta-extract-global-fact obj state) alist)
```

The above meta-extract hyps are accepted with *any term* in place of `obj` and `alist`.

```
(defchoose my-evl-contextual-badguy (obj) (a mfc state)
  (not (my-evl (meta-extract-contextual-fact
                 obj mfc state)
                a)))
```

- Means: “If there is an obj such that the evaluation of the meta-extract is false, return one”
- Using this as the `obj` implies the hyp for all `obj`.
- → At most two meta-extract hyps cover all uses.

A NICE SHORTCUT

Community book "clause-processors/meta-extract-user" defines event-generating macro `def-meta-extract`, which produces:

- bad guy functions for a given evaluator
- macros for meta-extract hyps using bad-guys
- theorems showing how these hyps imply the correctness of various tools/facts.

E.g.,

```
(defthm my-evl-meta-extract-formula
   (implies (and (my-evl-meta-extract-global-facts)
                 (equal (w st) (w state)))
            (my-evl (meta-extract-formula name st) a)))
```

Outline

SOME APPLICATIONS

- The GL symbolic interpreter uses meta-extract hypotheses to call functions, use rewrite rules, etc., without additional proof obligations
- The community book `centaur/misc/bound-rewriter.lisp` provides a tool for solving certain inequalities
- A meta rule for context-sensitive rewriting (like Greve's "nary" framework) is defined in `centaur/misc/context-rw.lisp`
- Others....

OUTLINE

Conclusion

Some concluding thoughts....

- This talk is just an introduction; meta reasoning is a bit complex to absorb in real time!
- The paper develops the ideas from this talk more thoroughly, with more illustrative examples.
- If you use GL then you are already taking advantage of meta-extract.