Meta-extract: Using Existing Facts in Meta-reasoning

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

ACL2 Workshop 2017
OUTLINE

INTRODUCTION

REVIEW OF: Meta RULES

EXAMPLE 1: USING GLOBAL FACTS

EXAMPLE 2: USING CONTEXTS

A NICE SHORTCUT

SOME APPLICATIONS

CONCLUSION
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
INTRODUCTION

REVIEW OF: Meta RULES

EXAMPLE 1: USING GLOBAL FACTS

EXAMPLE 2: USING CONTEXTS

A NICE SHORTCUT

SOME APPLICATIONS

CONCLUSION
 Canonical example of a \texttt{:meta} rule:
\begin{verbatim}
cancel_plus-equal (from "books/meta/meta-plus-equal.lisp")
cancels like terms from the equality of two sums.
\end{verbatim}

\begin{verbatim}
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))
\end{verbatim}
**Review of :Meta Rules (2)**

Key events:

- Define an evaluator:
  
  ```lisp
  (defevaluator ev-plus-equal ...)  
  (ev-plus-equal term alist) --> value
  ```

- Define the metafunction:
  
  ```lisp
  (defun cancel_plus-equal (x) ...)  
  ```

- Prove the metafunction correct w.r.t. the evaluator:
  
  ```lisp
  (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

INTRODUCTION

REVIEW OF :Meta RULES

EXAMPLE 1: USING GLOBAL FACTS

EXAMPLE 2: USING CONTEXTS

A NICE SHORTCUT

SOME APPLICATIONS

CONCLUSION
Example 1: Using Global Facts

Goal: Rewrite stobj \((\text{accessor } (\text{updater val foo}))\) terms without either:

- proving \(n^2\) individual rules per stobj
- enabling accessors/updaters to expand to \(\text{nth/update-nth}\)

An approach: \(\text{nth-update-nth-ev-meta-fn}\) checks that accessor is defined as a call of \(\text{nth}\) and updater is defined as a call of \(\text{update-nth}\) and rewrites accordingly.
Example 1: Using Global Facts

- Can look up function definitions 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))
  ...)

▶ 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.
**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**

**Introduction**

**Review of Meta Rules**

**Example 1: Using Global Facts**

**Example 2: Using Contexts**

**A Nice Shortcut**

**Some Applications**

**Conclusion**
**Example 2: Using Contexts**

Consider this metafunction:

```lisp
(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 \texttt{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** \texttt{nth-symbolp-metafn}:

```lisp
(defun nth-symbolp-meta (term mfc state a)
  (equal (nth-symbolp-metafn term mfc state a)
          (nth-meta-ev (nth-extract-contextual-fact'
                           (:typeset , (cadr term))
                           mfc state)
                      a))

:rule-classes ((:meta :trigger-fns (nth)))
```

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

```lisp
(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 \((\text{mfc-ts})\)
- Rewriting \((\text{mfc-rw, mfc-rw+, mfc-relieve-hyp})\)
- Linear arithmetic \((\text{mfc-ap})\)
OUTLINE

INTRODUCTION

REVIEW OF :Meta RULES

EXAMPLE 1: USING GLOBAL FACTS

EXAMPLE 2: USING CONTEXTS

A NICE SHORTCUT

SOME APPLICATIONS

CONCLUSION
A Nice Shortcut

\[(\text{my-evl } (\text{meta-extract-contextual-fact } \text{obj mfc state}) \text{ a})\]
\[(\text{my-evl } (\text{meta-extract-global-fact } \text{obj state}) \text{ alist})\]

The above meta-extract hyps are accepted with \textit{any term} in place of \texttt{obj and alist}.

\[(\text{defchoose my-evl-contextual-badguy (obj) (a mfc state)}\]
\[(\text{not (my-evl (meta-extract-contextual-fact obj mfc state)}\]
\[\text{a}))\]

\begin{itemize}
  \item Means: “If there is an obj such that the evaluation of the meta-extract is false, return one”
  \item Using this as the \texttt{obj} implies the hyp for all \texttt{obj}.
  \item \(\rightarrow\) At most two meta-extract hyps cover all uses.
\end{itemize}
A NICE SHORTCUT

Community book “clause-processors/meta-extract-user” defines event-generating macro \texttt{def-meta-extract}, which produces:

\begin{itemize}
\item bad guy functions for a given evaluator
\item macros for meta-extract hyps using bad-guys
\item theorems showing how these hyps imply the correctness of various tools/facts.
\end{itemize}

E.g.,

\begin{verbatim}
(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)))
\end{verbatim}
**OUTLINE**

- **Introduction**
- **Review of Meta Rules**
- **Example 1: Using Global Facts**
- **Example 2: Using Contexts**
- **A Nice Shortcut**
- **Some Applications**
- **Conclusion**
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

INTRODUCTION

REVIEW OF : Meta RULES

EXAMPLE 1: USING GLOBAL FACTS

EXAMPLE 2: USING CONTEXTS

A NICE SHORTCUT

SOME APPLICATIONS

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.