Improving Eliminate-Irrelevance for ACL2

Matt Kaufmann
(Joint Work with J Moore)

The University of Texas at Austin

October 14, 2016
Organization of this talk.

1. Review the ACL2 waterfall and its eliminate-irrelevance clause-processor.
   ▶ Section Waterfall
   ▶ Section Eliminate-Irrelevance

2. Present a recent change in its heuristics.
   ▶ Section Example
   ▶ Section Details

3. Remark on considerations when designing and implementing that change.
   ▶ Section Further Considerations
THE ACL2 WATERFALL
**Clause Processors**

Every ACL2 goal is represented as a *clause*: a list that is viewed as a disjunction of terms (called *literals*).

**Example:** A goal and corresponding clause:

\[
\text{(implies (and (p1 x) (p2 x y)) (p3 y))}
\]

\[
\text{((not (p1 x)), (not (p2 x y)), (p3 y))}
\]

Each waterfall step uses a *clause-processor*: a function that maps a clause to a list of clauses (possibly empty). Key property:

*If every result clause is a theorem, then the input clause is a theorem.*

**NOTE:** Converse need not hold!
**Introduction to Eliminate-Irrelevance**

Example from the ACL2 regression suite, in:
books/workshops/2006/cowles-gamboa-euclid/Euclid/fld-u-poly/.

```
(ld "fuproducto.port")
(in-package "FUPOL")
(rebuild "fuproducto.lisp" '*)

; Succeeds:
(thm ; polinomiop-*
  (polinomiop (* p q)))

; Fails:
(thm ; polinomiop-*
  (polinomiop (* p q))
  :hints
  (("Goal"
    :do-not '(eliminate-irrelevance))))
```
From successful proof, after `(set-gag-mode nil)`: 

Subgoal *1/2’5’
(IMPLIES (AND (MONOMIOP P1)
                   (POLINOMIOP P2)
                   (POLINOMIOP V*0))
                   (POLINOMIOP (APPEND (*-MONOMIO P1 Q) V*0))).

We suspect that the term `(POLINOMIOP P2)` is irrelevant to the truth of this conjecture and throw it out. We will thus try to prove 

Subgoal *1/2’6’
(IMPLIES (AND (MONOMIOP P1) (POLINOMIOP V*0))
                   (POLINOMIOP (APPEND (*-MONOMIO P1 Q) V*0))).

Name the formula above *1.1.

...

We will induct according to a scheme suggested by `(POLINOMIOP V*0)`. 

In the failed proof, keeping the literal `(POLINOMIOP P2)`: 

We will induct according to a scheme suggested by `(POLINOMIOP P2)`.
A HEURISTIC

Consider again this goal:

\[
\text{IMPLIES} \ (\text{AND} \ (\text{MONOMIOP} \ P1) \\
\quad \quad (\text{POLINOMIOP} \ P2) \\
\quad \quad (\text{POLINOMIOP} \ V*0)) \\
\quad \quad (\text{POLINOMIOP} \ (\text{APPEND} \ (*-\text{MONOMIO} \ P1 \ Q) \ V*0)))
\]

ACL2 represents this as a clause (disjunction of literals):

\[
\{(\text{NOT} \ (\text{MONOMIOP} \ P1)), \\
\quad (\text{NOT} \ (\text{POLINOMIOP} \ P2)), \\
\quad (\text{NOT} \ (\text{POLINOMIOP} \ V*0)), \\
\quad (\text{POLINOMIOP} \ (\text{APPEND} \ (*-\text{MONOMIO} \ P1 \ Q) \ V*0))\}
\]

The relation of sharing a variable has two components.

\[
\{\{(\text{NOT} \ (\text{MONOMIOP} \ P1)), \\
\quad (\text{NOT} \ (\text{POLINOMIOP} \ V*0)), \\
\quad (\text{POLINOMIOP} \ (\text{APPEND} \ (*-\text{MONOMIO} \ P1 \ Q) \ V*0))\}, \\
\quad \{\text{NOT} \ (\text{POLINOMIOP} \ P2)\}\}
\]

ACL2 drops the component that has a single member.
CHANGING THE HEURISTIC: AN EXAMPLE

J Moore encountered a problem with this heuristic. The following simple example exhibits the problem.

(encapsulate ((p) => *) ((my-app * *) => *))
  (local (defun p () t))
  (local (defun my-app (x y) (append x y)))
  (defthm my-app-def
   (implies (p)
     (equal (my-app x y)
       (append x y)))))

(defun rev (x)
  (if (consp x)
    (my-app (rev (cdr x))
      (cons (car x) nil))
    nil))

(thm (implies (and (p)
       (true-listp x))
       (equal (rev (rev x)) x)))
ACL2 Version 7.2 discards \((P)\): proof then fails!

Subgoal *1/2’5’
\[
\begin{align*}
&\text{(IMPLIES (AND (P) (TRUE-LISTP X2)))} \\
&\quad \text{(EQUAL (REV (APPEND RV (LIST X1)))} \\
&\quad \quad \text{(CONS X1 (REV RV))))}.
\end{align*}
\]

We suspect that the terms (TRUE-LISTP X2) and (P) are irrelevant to the truth of this conjecture and throw them out. We will thus try to prove

Subgoal *1/2’6’
\[
\begin{align*}
&\text{(EQUAL (REV (APPEND RV (LIST X1)))} \\
&\quad \text{(CONS X1 (REV RV)))}.
\end{align*}
\]

Name the formula above *1.1.

But now, ACL2 keeps \((P)\), and the proof succeeds.

We suspect that the term (TRUE-LISTP X2) is irrelevant to the truth of this conjecture and throw it out. We will thus try to prove

Subgoal *1/2’6’
\[
\begin{align*}
&\text{(IMPLIES (P)} \\
&\quad \text{(EQUAL (REV (APPEND RV (LIST X1)))} \\
&\quad \quad \text{(CONS X1 (REV RV))))}.
\end{align*}
\]

9/19
THE CHANGE IN A NUTSHELL

Why does ACL2 now keep the hypothesis \((P)\)?
Technically: Why does ACL2 keep the literal \((\text{NOT } (P))\)?
Recall the theorem exported from our \texttt{encapsulate} event.

\begin{verbatim}
(defthm my-app-def
  (implies (p)
    (equal (my-app x y)
      (append x y))))
\end{verbatim}

- Variables of hypothesis \((p)\): \{\}. 
- Variables of left-hand side \((\text{my-app } x \ y)\): \{x, y\}.

These are disjoint sets! So the function symbol \(p\) is marked as \textit{relevant}, since \((p)\) can be useful for rewriting calls that don’t involve its (empty set of) variables.
The New Heuristic in More Detail

Suppose $p$ is a Boolean and we have two terms, as follows.

- Let $t_1$ be $(\text{FN } v_1 \ldots v_K)$, an application of a function symbol to distinct variables.
- Let $t_2$ be a term whose free variables are disjoint from those of $t_1$.

Then $\text{FN}$ is relevant with parity $p$ whenever $t_1$ or its negation is a hypothesis (perhaps among others), in which case:

- $p = \top$ if $t_1$ is a hypothesis;
- $p = \text{nil}$ if $(\text{not } t_1)$ is a hypothesis.
**Example of “Relevant with Parity”**

Recall our earlier example rewrite rule and the problem goal:

```lisp
(encapsulate (((p) => *) ((my-app * *) => *))
  (local (defun p () t))
  (local (defun my-app (x y) (append x y)))
  (defthm my-app-def
     (implies (p)
       (equal (my-app x y)
             (append x y)))))

(IMPLIES (AND (P) (TRUE-LISTP X2))
  (EQUAL (REV (APPEND RV (LIST X1)))
         (CONS X1 (REV RV))))
```

The “hypothesis” (P) is, internally, the literal (NOT (P)). Parity \(\top\) corresponds to “negated literal should be kept”, so:

ACL2 !>(assoc-eq 'p
     (global-val 'never-irrelevant-fns-alist
               (w state)))

(P . T)
ACL2 !>
Relevance with Parity for Various Rules

Assume that terms $t_1 = (\text{FN} \ V_1 \ldots \ V_K)$ (distinct $V_i$) and $t_2$ have disjoint free variables, where for a rule of the given class:

- **Rule-classes**: REWRITE and DEFINITION: $t_2$ is the rule’s left-hand side.
- **Rule-class**: LINEAR: $t_2$ is a max-term.
- **Rule-class**: TYPE-PRESCRIPTION: $t_2$ is a typed-term.
- **Rule-class**: FORWARD-CHAINING: $t_2$ is the conclusion.

Then $\text{FN}$ is relevant with parity $p$ for such rules when:

- $p = t$ : (implies (and ... $t_1$ ...) ...)
- $p = \text{nil}$: (implies (and ... (not $t_1$) ...) ...)

For a call $u$ of $\text{FN}$ on distinct variables:

- literal $u$ is never irrelevant (dropped) if $p = \text{nil}$; and
- literal (not $u$) is never irrelevant (dropped) if $p = t$. 
Additional Parities

- A function symbol $FN$ can be irrelevant with parity $\top$ in one rule and with parity $nil$ in another rule. We then store $FN$ with parity $both$.
- We also store $FN$ as irrelevant for suitable occurrences of $t_1$ in conclusions. That might be overkill.
- There is a second criterion for irrelevant components (besides single-literal components based on calls of irrelevant literals): all function symbols the component are among a fixed set of primitives.
  - Unchanged, except that $\text{NOT}$ has been added to that set (since the other criterion is stricter).
Timing (1)

Does the use of irrelevance with parity slow down ACL2?

- Does using of that information slow down the eliminate-irrelevance procedure?
  - Not concerning — procedure is invoked only just before a sub-induction; rather rare in practice.

- Is maintaining such information expensive?
  - Info is stored in an alist.
  - Each suitable rule causes linear lookup in the alist and possibly its extension — potentially quadratic behavior. (Should we consider an applicative hash-table (fast alist)?)

Regression suite didn’t show significant time difference, but let’s look at other evidence against slowdown.
TIMING (2)

Stress test:
(time$ (include-book "doc/top" :dir :system)).
Showed essentially no change!

;;; old
; 782.20 seconds realtime, 777.17 seconds runtime
; (23,612,574,784 bytes allocated).

;;; new
; 775.99 seconds realtime, 772.39 seconds runtime
; (23,952,558,640 bytes allocated).

ACL2 !>(length (global-val 'never-irrelevant-fns-alist (w state)))
11869
ACL2 !>
TIMING (3)

Seems like the new global is a non-issue, since a symbol-alist of length 11,869 is trivial to traverse. On my Mac:

ACL2 !>:q

Exiting the ACL2 read-eval-print loop. To re-enter, execute (LP).

? (defun foo (sym n)
   (let ((x (make-list n :initial-element '(a . b))))
     (time$ (assoc-eq sym x))))

FOO

? (foo 'c 1000000)
; (ASSOC-EQ SYM ...) took
; 0.00 seconds realtime, 0.00 seconds runtime
; (0 bytes allocated).

NIL

? (foo 'c 10000000)
; (ASSOC-EQ SYM ...) took
; 0.03 seconds realtime, 0.03 seconds runtime
; (0 bytes allocated).

NIL
MISCELLANEOUS CONSIDERATIONS

Question 1: Make the heuristic attachable?
Answer: Seems like overkill. After all, *eliminate-irrelevance* only occurs before a sub-induction, and nobody should rely on sub-inductions.

Question 2: Extend irrelevance with a sort of transitive closure?
Suppose for example we have these three rewrite rules.

\[
\begin{align*}
&\text{(implies (f1 x) (f2 x))} \\
&\text{(implies (f2 x) (f3 x))} \\
&\text{(implies (f3 x) (h y z))}
\end{align*}
\]

Then just as we don’t want to drop a hypothesis (negated literal for) \((f3 \ x)\), we don’t want to drop \((f1 \ x)\) or \((f2 \ x)\).
Answer: Nah, seems like overkill for such a last-ditch heuristic.
CONCLUDING REMARKS

▶ **Bottom line:** Eliminate-irrelevance is fairly minor. But this tweak, which arose from J’s work on apply$, was helpful for that work and could help others.

▶ **Thanks for your attention.**

▶ (If there’s extra time, I could give a sense of the source code (e.g., eliminate-irrelevance-clause (through irrelevant-lits and irrelevant-clausep) and add-rewrite-rule (through add-rewrite-rule2 and extend-never-irrelevant-fns-alist).)