Improving Eliminate-Irrelevance for ACL2

Matt Kaufmann
(Joint Work with J Moore)

The University of Texas at Austin

October 14, 2016
## OUTLINE

Organization of this talk.
OUTLINE

Organization of this talk.

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

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
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

- Destructor Elimination
- Equality
- Generalization
- Elimination of Irrelevance
- Induction
- Simplification

User formula \rightarrow pool

\text{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).
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*:

\[
\text{Example:}\quad \text{A }\mathbf{goal} \text{ and corresponding } \mathbf{clause}: \\
(\text{implies } (\text{and } (p1 \ x) \ (p2 \ x \ y)) \\
(p3 \ y)) \\
((\text{not } (p1 \ x)), \ (\text{not } (p2 \ x \ y)), \ (p3 \ y))
\]
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:

\[(\text{implies } (\text{and } (p_1 x) (p_2 x y)) (p_3 y))\]
\[(((\text{not } (p_1 x)), (\text{not } (p_2 x y)), (p_3 y))\]

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

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

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 "FUPOLE")
(rebuild "fuproducto.lisp" '*)

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

; Fails:
(thm ; polinomio-*
   (polinomio (* 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).
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:

(IMPLIES (AND (MONOMIOP P1)

(POLINOMIOP P2)

(POLINOMIOP V*0))

(POLINOMIOP (APPEND (*-MONOMIO P1 Q) V*0)))
A HEURISTIC

Consider again this goal:

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

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

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

Consider again this goal:

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

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

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

The relation of *sharing a variable* has two components.

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

Consider again this goal:

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

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

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

The relation of \textit{sharing a variable} has two components.

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

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.

```
(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)))
```
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)
      (implies (and (true-listp x)
                  (equal (my-app x y)
                        (append x y)))))))
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))
CHANGING THE HEURISTIC: AN EXAMPLE

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

(encapsulate (p) (my-app x y))
  (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' (IMPLIES (AND (P) (TRUE-LISTP X2)) (EQUAL (REV (APPEND RV (LIST X1))) (CONS X1 (REV RV))).

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' (EQUAL (REV (APPEND RV (LIST X1))) (CONS X1 (REV RV))). 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' (IMPLIES (P) (EQUAL (REV (APPEND RV (LIST X1))) (CONS X1 (REV RV)))).
ACL2 Version 7.2 discards \((P)\): proof then fails!

Subgoal *1/2’5’

\[
\text{(IMPLIES (AND (P) (TRUE-LISTP X2))}
\]
\[
\text{\hspace{1em} (EQUAL (REV (APPEND RV (LIST X1))))}
\]
\[
\text{\hspace{1em} (CONS X1 (REV RV))).}
\]

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’

\[
\text{(EQUAL (REV (APPEND RV (LIST X1)))}
\]
\[
\text{\hspace{1em} (CONS X1 (REV RV))).}
\]

Name the formula above *1.1.
ACL2 Version 7.2 discards \((P)\): proof then fails!

Subgoal *1/2'5’
(IMPLIES (AND (P) (TRUE-LISTP X2))
  (EQUAL (REV (APPEND RV (LIST X1)))
  (CONS X1 (REV RV))))).

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’
(EQUAL (REV (APPEND RV (LIST X1)))
  (CONS X1 (REV RV))).

Name the formula above *1.1.

But now, ACL2 keeps \((P)\), and the proof succeeds.
ACL2 Version 7.2 discards \((P)\): proof then fails!

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

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

Subgoal \(*1/2'6'\)
\[(\text{EQUAL} \ (\text{REV} \ (\text{APPEND} \ RV \ (\text{LIST} \ X1))) 
\quad \text{(CONS} \ X1 \ (\text{REV} \ RV)))).\]

Name the formula above \(*1.1.\)

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

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

Subgoal \(*1/2'6'\)
\[(\text{IMPLIES} \ (P) \n\quad \text{(EQUAL} \ (\text{REV} \ (\text{APPEND} \ RV \ (\text{LIST} \ X1))) 
\quad \quad \text{(CONS} \ X1 \ (\text{REV} \ RV))))).\]
The Change in a Nutshell

Why does ACL2 now keep the hypothesis \((P)\)?
The Change in a Nutshell

Why does ACL2 now keep the hypothesis \((P)\)?
Technically: Why does ACL2 keep the literal \((\text{NOT} \ (P))\)?
**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 encapsulate event.

```lisp
(defthm my-app-def
  (implies (p)
    (equal (my-app x y)
      (append x y))))
```
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 encapsulate event.

(defthm my-app-def
  (implies (p)
    (equal (my-app x y)
      (append x y))))

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

```lisp
(defthm my-app-def
  (implies (p)
    (equal (my-app x y)
      (append x y))))
```

- 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 *relevant*, since \((p)\) can be useful for rewriting calls that don’t involve its (empty set of) variables.
Suppose $p$ is a Boolean and we have two terms, as follows.

- Let $t_1$ be $(FN \ V_1 \ldots V_K)$, an application of a function symbol to distinct variables.
- Let $t_2$ be a term whose free variables are disjoint from those of $t_1$. 
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:

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

The “hypothesis” (P) is, internally, the literal \( \neg (P) \).

Parity \( t \) corresponds to “negated literal should be kept”, so:

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

ACL2 !>
EXAMPLE OF “RELEVANT WITH PARITY”

Recall our earlier example rewrite rule and the problem goal:

```
(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))))
```
Example of "Relevant with Parity"

Recall our earlier example rewrite rule and the problem goal:

(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 \( \text{NOT} \ (P) \). Parity \( t \) corresponds to "negated literal should be kept", so:
**Example of “Relevant with Parity”**

Recall our earlier example rewrite rule and the problem goal:

```
(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 t 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:
**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**: \texttt{REWRITE} and \texttt{DEFINITION}: $t_2$ is the rule’s left-hand side.
- **Rule-class**: \texttt{LINEAR}: $t_2$ is a max-term.
- **Rule-class**: \texttt{TYPE-PRESCRIPTION}: $t_2$ is a typed-term.
- **Rule-class**: \texttt{FORWARD-CHAINING}: $t_2$ is the conclusion.
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:

- \( p=t \) : \((\text{implies} \ (\text{and} \ldots \ t_1 \ldots) \ldots)\)
- \( p=\text{nil} \) : \((\text{implies} \ (\text{and} \ldots \ (\text{not} \ t_1) \ldots) \ldots)\)
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**: \text{REWRITE and DEFINITION}: \( t_2 \) is the rule’s left-hand side.
- **Rule-class**: \text{LINEAR}: \( t_2 \) is a max-term.
- **Rule-class**: \text{TYPE-PRESCRIPTION}: \( t_2 \) is a typed-term.
- **Rule-class**: \text{FORWARD-CHAINING}: \( t_2 \) is the conclusion.

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

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

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

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

- A function symbol $\text{FN}$ can be irrelevant with parity $\text{t}$ in one rule and with parity $\text{nil}$ in another rule. We then store $\text{FN}$ with parity $\text{both}$.
A function symbol $FN$ can be irrelevant with parity $\top$ in one rule and with parity $\text{nil}$ in another rule. We then store $FN$ with parity $:\text{both}$.

We also store $FN$ as irrelevant for suitable occurrences of $t_1$ in conclusions. That might be overkill.
Additional Parities

- A function symbol $\text{FN}$ can be irrelevant with parity $\top$ in one rule and with parity $\text{nil}$ in another rule. We then store $\text{FN}$ with parity $\text{both}$.
- We also store $\text{FN}$ 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.
A function symbol $\text{FN}$ can be irrelevant with parity $\top$ in one rule and with parity $\text{nil}$ in another rule. We then store $\text{FN}$ with parity $\text{both}$.

We also store $\text{FN}$ 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?
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.
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*)?)
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:

```lisp
(time$ (include-book "doc/top" :dir :system)).
```

Showed essentially no change!

```lisp
;;; 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
?
Question 1: Make the heuristic attachable?
**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.
**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?
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 \((f2 \ x)\).
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).)