Double Rewriting for Equivalential Reasoning in ACL2

Matt Kaufmann and J Strother Moore

ACL2 Workshop, FLoC, Seattle, August 16, 2006

Introduction

- ACL2 provides a powerful *congruence-based* rewriting capability.
- However, some have encountered an issue in using this feature.
- In this talk we describe that issue and a partial solution (starting with ACL2 Version 2.9.4, February, 2006).
- Above, we say "partial" because the solution requires users to annotate rewrite rules. We are soliciting ideas from the user community for how to automate this solution.

Introduction

- ACL2 provides a powerful *congruence-based* rewriting capability.
- However, some have encountered an issue in using this feature.
- In this talk we describe that issue and a partial solution (starting with ACL2 Version 2.9.4, February, 2006).
- Above, we say "partial" because the solution requires users to annotate rewrite rules. We are soliciting ideas from the user community for how to automate this solution.

Introduction

- ACL2 provides a powerful *congruence-based* rewriting capability.
- However, some have encountered an issue in using this feature.
- In this talk we describe that issue and a partial solution (starting with ACL2 Version 2.9.4, February, 2006).
- Above, we say "partial" because the solution requires users to annotate rewrite rules. We are soliciting ideas from the user community for how to automate this solution.

Introduction

- ACL2 provides a powerful *congruence-based* rewriting capability.
- However, some have encountered an issue in using this feature.
- In this talk we describe that issue and a partial solution (starting with ACL2 Version 2.9.4, February, 2006).
- Above, we say “partial” because the solution requires users to annotate rewrite rules. We are soliciting ideas from the user community for how to automate this solution.

Outline

- Introduction
- Review of congruence-based rewriting in ACL2
- The problem with caching
- A partial solution
- Warning messages
- Conclusion and a plea for help

Review of congruence-based rewriting in ACL2

The rewriter is given:

- a term, α;
- a substitution, σ;
- an equivalence relation, *equiv*; and
- some assumptions, γ

It returns a term β such that the following is an ACL2 theorem:

- (implies γ (*equiv* α/σ β))

The rewriter *maintains* a set of equivalence relations for which it can do such a rewrite.

Review of congruence-based rewriting in ACL2

The rewriter is given:

- a term, α;
- a substitution, σ;
- an equivalence relation, *equiv*; and
- some assumptions, γ

It returns a term β such that the following is an ACL2 theorem:

- (implies γ (*equiv* α/σ β))

The rewriter *maintains* a set of equivalence relations for which it can do such a rewrite.

Review of congruence-based rewriting in ACL2

The rewriter is given:

- a term, α;
- a substitution, σ;
- an equivalence relation, *equiv*; and
- some assumptions, γ

It returns a term β such that the following is an ACL2 theorem:

- (implies γ (*equiv* α/σ β))

The rewriter *maintains* a set of equivalence relations for which it can do such a rewrite.

The problem with caching – Wishful thinking

Distillation of an example from Dave Greve:

```
(defequiv equiv)
(defcong equiv iff (pred x) 1)
(defthm pred-h (pred (h x)))
(defthm g-to-h (equiv (g x) (h x)))
(defthm pred-implies-f
  (implies (pred x) (iff (f x) t)))
```

Consider the rewrite of `(f (g y))`. **NAIVELY**:

```
> (f (g y))           [matches pred-implies-f]
  >> (pred (g y))     [try to relieve hypothesis]
    >>> (g y)         [rewrite inside-out, in
                       equiv context (by defcong)]
    <<< (h y)         [by g-to-h]
  << (pred (h y))     ... rewrites to t by pred-h
```

The problem with caching – Wishful thinking

Distillation of an example from Dave Greve:

```
(defequiv equiv)
(defcong equiv iff (pred x) 1)
(defthm pred-h (pred (h x)))
(defthm g-to-h (equiv (g x) (h x)))
(defthm pred-implies-f
  (implies (pred x) (iff (f x) t)))
```

Consider the rewrite of `(f (g y))`. **NAIVELY**:

```
> (f (g y))          [matches pred-implies-f]
  >> (pred (g y))    [try to relieve hypothesis]
    >>> (g y)        [rewrite inside-out, in
                      equiv context (by defcong)]
    <<< (h y)        [by g-to-h]
  << (pred (h y))    ... rewrites to t by pred-h
```

The problem with caching – Wishful thinking

Distillation of an example from Dave Greve:

```
(defequiv equiv)
(defcong equiv iff (pred x) 1)
(defthm pred-h (pred (h x)))
(defthm g-to-h (equiv (g x) (h x)))
(defthm pred-implies-f
  (implies (pred x) (iff (f x) t)))
```

Consider the rewrite of `(f (g y))`. **NAIVELY**:

```
> (f (g y))           [matches pred-implies-f]
  >> (pred (g y))     [try to relieve hypothesis]
    >>> (g y)         [rewrite inside-out, in
                       equiv context (by defcong)]
    <<< (h y)         [by g-to-h]
  << (pred (h y))     ... rewrites to t by pred-h
```

The problem with caching – The reality

```
(defcong equiv iff (pred x) 1)
(defthm pred-h (pred (h x)))
(defthm g-to-h (equiv (g x) (h x)))
(defthm pred-implies-f
  (implies (pred x) (iff (f x) t)))
-------------------------------------------------
> (f (g y))
  >> (g y)            [rewrite inside-out]
  << (g y)            [unable to apply g-to-h]
[Now match pred-implies-f]
> (f x)               {x := (g y)}
  >> (pred x)         {x := (g y)} [relieve hyp]
    >>> x             {x := (g y)} [rw inside-out]
    <<< (g y)         [by lookup]
  << (pred (g y))     [cannot be further rewritten,
                       so ‘relieve hyp’ fails]
```

A partial solution

```
(defcong equiv iff (pred x) 1)
(defthm pred-h (pred (h x)))
(defthm g-to-h (equiv (g x) (h x)))
(defthm pred-implies-f
  (implies (pred (double-rewrite x))
           (iff (f x) t)))
-------------------------------------------------
> (f x)                  {x := (g y)}
  >> (pred (d-rw x)) {x := (g y)}
    >>> (d-rw x)         {x := (g y)} [rw inside-out]
      >>>> (g y)         {} [d-rw, so rewrite again!]
      <<<< (h y)         [by g-to-h]
    <<< (h y)
  << (pred (h y)) [Now rewrite with pred-h.]
  >> (pred (h x))        {x := y}
  << t                   [by pred-h]
```

Warning messages

ACL2 warns as follows when it sees possible benefit for the insertion of a `double-rewrite` call. See the paper for details. (Most of the implementation work was in producing warnings.)

```
ACL2 Warning [Double-rewrite] in ( DEFTHM
PRED-IMPLIES-F ...): In a :REWRITE rule generated
from PRED-IMPLIES-F, equivalence relation EQUIV is
maintained at one problematic occurrence of
variable X in the first hypothesis, but not at any
binding occurrence of X.  Consider replacing that
occurrence of X in the first hypothesis with
(DOUBLE-REWRITE X).  See :doc double-rewrite for
more information on this issue.
```

Conclusion and a plea for help

- Manual insertion of `double-rewrite` can avoid failures to relieve hypotheses due to rewrite caching.
- **NOTE**: We automate double rewriting (since Version 2.9, October 2004) at the top level of a hypothesis.
- **CURRENT ADVICE**: Insert `double-rewrite` when there is a warning. If ACL2 seems slow, use `accumulated-persistence` for debug.
- **CHALLENGE**: Find heuristics for when to insert `double-rewrite` without significantly slowing down the rewriter. Insertion to eliminate every warning appears to be too expensive (see 100x example in the paper).

Conclusion and a plea for help

- Manual insertion of `double-rewrite` can avoid failures to relieve hypotheses due to rewrite caching.
- **NOTE**: We automate double rewriting (since Version 2.9, October 2004) at the top level of a hypothesis.
- **CURRENT ADVICE**: Insert `double-rewrite` when there is a warning. If ACL2 seems slow, use `accumulated-persistence` for debug.
- **CHALLENGE**: Find heuristics for when to insert `double-rewrite` without significantly slowing down the rewriter. Insertion to eliminate every warning appears to be too expensive (see 100x example in the paper).

Conclusion and a plea for help

- Manual insertion of `double-rewrite` can avoid failures to relieve hypotheses due to rewrite caching.
- **NOTE**: We automate double rewriting (since Version 2.9, October 2004) at the top level of a hypothesis.
- **CURRENT ADVICE**: Insert `double-rewrite` when there is a warning. If ACL2 seems slow, use `accumulated-persistence` for debug.
- **CHALLENGE**: Find heuristics for when to insert `double-rewrite` without significantly slowing down the rewriter. Insertion to eliminate every warning appears to be too expensive (see 100x example in the paper).

Conclusion and a plea for help

- Manual insertion of `double-rewrite` can avoid failures to relieve hypotheses due to rewrite caching.
- **NOTE**: We automate double rewriting (since Version 2.9, October 2004) at the top level of a hypothesis.
- **CURRENT ADVICE**: Insert `double-rewrite` when there is a warning. If ACL2 seems slow, use `accumulated-persistence` for debug.
- **CHALLENGE**: Find heuristics for when to insert `double-rewrite` without significantly slowing down the rewriter. Insertion to eliminate every warning appears to be too expensive (see 100x example in the paper).