

Foundations of Automated Induction for a Structured Mechanized Logic

Matt Kaufmann
J Strother Moore
Sandip Ray

Department of Computer Sciences
University of Texas at Austin

`{kaufmann,moore,sandip}@cs.utexas.edu`

Soundness Bug Fixed in ACL2 Version 3.1 (12/06)

```
(encapsulate ()
  (local
    (defun foo (x y)
      (declare (xargs :measure (acl2-count y)))
      (if (and (consp x) (consp y))
          (foo (cons x x) (cdr y))
          y)))
    (defun foo (x y) <... same body as above ...>))

(defthm bad
  (atom x)
  :rule-classes nil
  :hints (("Goal" :induct (foo x '(3)))))

(defthm contradiction
  nil
  :rule-classes nil
  :hints (("Goal" :use (:instance bad (x '(7)))))
```

Soundness Bug Fixed in ACL2 Version 3.1 (12/06)

```
(encapsulate ()
  (local
    (defun foo (x y)
      (declare (xargs :measure (acl2-count y)))
      (if (and (consp x) (consp y))
          (foo (cons x x) (cdr y))
          y)))
    (defun foo (x y) <... same body as above ...>))

(defthm bad
  (atom x)
  :rule-classes nil
  :hints (("Goal" :induct (foo x '(3)))))

(defthm contradiction
  nil
  :rule-classes nil
  :hints (("Goal" :use (:instance bad (x '(7)))))
```

Soundness Bug Fixed in ACL2 Version 3.1 (12/06)

```
(encapsulate ()
  (local
    (defun foo (x y)
      (declare (xargs :measure (acl2-count y)))
      (if (and (consp x) (consp y))
          (foo (cons x x) (cdr y))
          y)))
    (defun foo (x y) <... same body as above ...>))

(defthm bad
  (atom x)
  :rule-classes nil
  :hints (("Goal" :induct (foo x '(3)))))

(defthm contradiction
  nil
  :rule-classes nil
  :hints (("Goal" :use (:instance bad (x '(7)))))
```

Digging into the Soundness Bug

```
(encapsulate ()
  (local
    (defun foo (x y)
      (declare (xargs :measure (acl2-count y)))
      (if (and (consp x) (consp y))
          (foo (cons x x) (cdr y))
          y)))
  (defun foo (x y) <... same body as above ...>))
```

Measured subset determines the legal induction schemes.

- ▶ The wrong measured subset $\{x\}$ permits the spurious induction scheme generated from `(foo x '(3))`.

```
(AND (IMPLIES (NOT (CONSP X)) (:P X))
      (IMPLIES (AND (CONSP X) (:P (CONS X X)))
                (:P X)))
```

In the proof of NIL, $(:P x) \triangleq (\text{atom } x)$.

Apparently the problem seems to be merely an issue with ACL2's redundancy checking mechanism.

Digging into the Soundness Bug

```
(encapsulate ()
  (local
    (defun foo (x y)
      (declare (xargs :measure (acl2-count y)))
      (if (and (consp x) (consp y))
          (foo (cons x x) (cdr y))
          y)))
  (defun foo (x y) <... same body as above ...>))
```

Measured subset determines the legal induction schemes.

- ▶ The wrong measured subset $\{x\}$ permits the spurious induction scheme generated from $(foo\ x\ '(3))$.

```
(AND (IMPLIES (NOT (CONSP X)) (:P X))
      (IMPLIES (AND (CONSP X) (:P (CONS X X)))
                (:P X)))
```

In the proof of NIL, $(:P\ x) \triangleq (\text{atom } x)$.

Apparently the problem seems to be merely an issue with ACL2's redundancy checking mechanism.

Digging into the Soundness Bug

```
(encapsulate ()
  (local
    (defun foo (x y)
      (declare (xargs :measure (acl2-count y)))
      (if (and (consp x) (consp y))
          (foo (cons x x) (cdr y))
          y)))
  (defun foo (x y) <... same body as above ...>))
```

Measured subset determines the legal induction schemes.

- ▶ The wrong measured subset $\{x\}$ permits the spurious induction scheme generated from $(foo\ x\ '(3))$.

```
(AND (IMPLIES (NOT (CONSP X)) (:P X))
      (IMPLIES (AND (CONSP X) (:P (CONS X X)))
                (:P X)))
```

In the proof of NIL, $(:P\ x) \stackrel{\Delta}{=} (\text{atom } x)$.

Apparently the problem seems to be merely an issue with ACL2's redundancy checking mechanism.

Digging into the Soundness Bug

```
(encapsulate ()
  (local
    (defun foo (x y)
      (declare (xargs :measure (acl2-count y)))
      (if (and (consp x) (consp y))
          (foo (cons x x) (cdr y))
          y)))
  (defun foo (x y) <... same body as above ...>))
```

Measured subset determines the legal induction schemes.

- ▶ The wrong measured subset $\{x\}$ permits the spurious induction scheme generated from $(foo\ x\ '(3))$.

```
(AND (IMPLIES (NOT (CONSP X)) (:P X))
      (IMPLIES (AND (CONSP X) (:P (CONS X X)))
                (:P X)))
```

In the proof of NIL, $(:P\ x) \triangleq (\text{atom } x)$.

Apparently the problem seems to be merely an issue with ACL2's redundancy checking mechanism.

Local Measures

Should we require that the measure be supplied explicitly?

- ▶ But suppose the measure in a LOCAL definition is also LOCAL.

```
(local
  (defun foo (x y)
    (declare (xargs :measure (my-local-meas y)))
    ...))
```

- ▶ How can we tell ACL2 to admit `foo` non-LOCALLY without exporting the measure?

```
(defun foo (x y)
  (declare (xargs :measure (:? y)))
  ...)
```

ACL2 considers the second measure *redundant*.

- ▶ **One small change** for users;
one giant headache for foundations!

Local Measures

Should we require that the measure be supplied explicitly?

- ▶ But suppose the measure in a LOCAL definition is also LOCAL.

```
(local
  (defun foo (x y)
    (declare (xargs :measure (my-local-meas y)))
    ...))
```

- ▶ How can we tell ACL2 to admit `foo` non-LOCALLY without exporting the measure?

```
(defun foo (x y)
  (declare (xargs :measure (:? y)))
  ...)
```

ACL2 considers the second measure *redundant*.

- ▶ One small **change** for users;
one giant **headache** for foundations!

Local Measures

Should we require that the measure be supplied explicitly?

- ▶ But suppose the measure in a LOCAL definition is also LOCAL.

```
(local
  (defun foo (x y)
    (declare (xargs :measure (my-local-meas y)))
    ...))
```

- ▶ How can we tell ACL2 to admit `foo` non-LOCALLY without exporting the measure?

```
(defun foo (x y)
  (declare (xargs :measure (:? y)))
  ...)
```

ACL2 considers the second measure *redundant*.

- ▶ One small **change** for users;
one giant **headache** for foundations!

Local Measures

Should we require that the measure be supplied explicitly?

- ▶ But suppose the measure in a LOCAL definition is also LOCAL.

```
(local
  (defun foo (x y)
    (declare (xargs :measure (my-local-meas y)))
    ...))
```

- ▶ How can we tell ACL2 to admit `foo` non-LOCALLY without exporting the measure?

```
(defun foo (x y)
  (declare (xargs :measure (:? y)))
  ...)
```

ACL2 considers the second measure *redundant*.

- ▶ One small **change** for users;
one giant **headache** for foundations!

Local Measures

Should we require that the measure be supplied explicitly?

- ▶ But suppose the measure in a LOCAL definition is also LOCAL.

```
(local
  (defun foo (x y)
    (declare (xargs :measure (my-local-meas y)))
    ...))
```

- ▶ How can we tell ACL2 to admit `foo` non-LOCALLY without exporting the measure?

```
(defun foo (x y)
  (declare (xargs :measure (:? y)))
  ...)
```

ACL2 considers the second measure *redundant*.

- ▶ **One small change** for users;
one giant headache for foundations!

Local Events in ACL2

- ▶ Soundness bugs in past versions of ACL2 have often been due to subtle issues with ACL2's structuring mechanisms, in particular `LOCAL` events.
- ▶ We need a clear specification of ACL2 at the logical level in order to have any hope of getting its design right, especially in the presence of `LOCAL`!

Local Events in ACL2

- ▶ Soundness bugs in past versions of ACL2 have often been due to subtle issues with ACL2's structuring mechanisms, in particular `LOCAL` events.

- ▶ We need a clear specification of ACL2 at the logical level in order to have any hope of getting its design right, especially in the presence of `LOCAL`!

Structured Theory

Existing logical formalization to account for LOCAL events:

- ▶ M. Kaufmann and J Moore, “Structured Theory Development for a Mechanized Logic.” *Journal of Automated Reasoning* 26(2) (2001) 161-203.

See **Books and Papers** Link in **ACL2 Home Page**

Main Result: If a formula ϕ is proven as a theorem in an ACL2 session, then ϕ is first-order derivable from the ground-zero theory together with only the axiomatic events in the session.

- ▶ **Key Observation** Each extension principle (other than `defaxiom`) produces a **conservative** extension of the current theory.

Reworking the Structured Theory

The “Structured Theory” paper used a notion of **interpreter admissibility** to formalize the notion of a valid `defun` event.

- ▶ If a definition is admitted (with a measure) in ACL2 then it is interpreter admissible
- ▶ If a definition is interpreter admissible then there is a **canonical measure** admitting it.

Interpreter admissibility is a key ingredient in the proof of conservativity of ACL2’s definitional principle.

But this notion does not account for induction schemes that take advantage of measured subsets.

We are taking a somewhat different approach in a new paper that addresses this issue. It also fixes a mistake or two, and it provides a cleaner logical specification.

Reworking the Structured Theory

The “Structured Theory” paper used a notion of **interpreter admissibility** to formalize the notion of a valid `defun` event.

- ▶ If a definition is admitted (with a measure) in ACL2 then it is interpreter admissible
- ▶ If a definition is interpreter admissible then there is a **canonical measure** admitting it.

Interpreter admissibility is a key ingredient in the proof of conservativity of ACL2’s definitional principle.

But this notion does not account for induction schemes that take advantage of measured subsets.

We are taking a somewhat different approach in a new paper that addresses this issue. It also fixes a mistake or two, and it provides a cleaner logical specification.

Reworking the Structured Theory

The “Structured Theory” paper used a notion of **interpreter admissibility** to formalize the notion of a valid `defun` event.

- ▶ If a definition is admitted (with a measure) in ACL2 then it is interpreter admissible
- ▶ If a definition is interpreter admissible then there is a **canonical measure** admitting it.

Interpreter admissibility is a key ingredient in the proof of conservativity of ACL2’s definitional principle.

But this notion does not account for induction schemes that take advantage of measured subsets.

We are taking a somewhat different approach in a new paper that addresses this issue. It also fixes a mistake or two, and it provides a cleaner logical specification.

Reworking the Structured Theory

The “Structured Theory” paper used a notion of **interpreter admissibility** to formalize the notion of a valid `defun` event.

- ▶ If a definition is admitted (with a measure) in ACL2 then it is interpreter admissible
- ▶ If a definition is interpreter admissible then there is a **canonical measure** admitting it.

Interpreter admissibility is a key ingredient in the proof of conservativity of ACL2’s definitional principle.

But this notion does not account for induction schemes that take advantage of measured subsets.

We are taking a somewhat different approach in a new paper that addresses this issue. It also fixes a mistake or two, and it provides a cleaner logical specification.

Random Closing Remarks

- ▶ **Thesis:** Getting the logical foundations of structuring mechanisms right is worth the considerable effort required, because it can help to avoid soundness bugs.
- ▶ **Question:** Why bother to support LOCAL at all?
Answers:
 1. Supports independent proof development by different users.
 2. LOCAL events are skipped when including a book, which provides potentially large speed-ups in book inclusion.
 3. We want to require that theorems of an included book follow from its axiomatic events. LOCAL events allow us freely to add auxiliary definitions while preserving this requirement.
- ▶ Details are in our new paper. Please feel free to request a preprint (available soon... 😊).

Random Closing Remarks

- ▶ **Thesis:** Getting the logical foundations of structuring mechanisms right is worth the considerable effort required, because it can help to avoid soundness bugs.

- ▶ **Question:** Why bother to support LOCAL at all?

Answers:

1. Supports independent proof development by different users.
 2. LOCAL events are skipped when including a book, which provides potentially large speed-ups in book inclusion.
 3. We want to require that theorems of an included book follow from its axiomatic events. LOCAL events allow us freely to add auxiliary definitions while preserving this requirement.
- ▶ Details are in our new paper. Please feel free to request a preprint (available soon... 😊).

Random Closing Remarks

- ▶ **Thesis:** Getting the logical foundations of structuring mechanisms right is worth the considerable effort required, because it can help to avoid soundness bugs.
- ▶ **Question:** Why bother to support LOCAL at all?
Answers:
 1. Supports independent proof development by different users.
 2. LOCAL events are skipped when including a book, which provides potentially large speed-ups in book inclusion.
 3. We want to require that theorems of an included book follow from its axiomatic events. LOCAL events allow us freely to add auxiliary definitions while preserving this requirement.
- ▶ Details are in our new paper. Please feel free to request a preprint (available soon... 😊).


Random Closing Remarks

- ▶ **Thesis:** Getting the logical foundations of structuring mechanisms right is worth the considerable effort required, because it can help to avoid soundness bugs.
- ▶ **Question:** Why bother to support LOCAL at all?
Answers:
 1. Supports independent proof development by different users.
 2. LOCAL events are skipped when including a book, which provides potentially large speed-ups in book inclusion.
 3. We want to require that theorems of an included book follow from its axiomatic events. LOCAL events allow us freely to add auxiliary definitions while preserving this requirement.
- ▶ Details are in our new paper. Please feel free to request a preprint (available soon... 😊).

Random Closing Remarks

- ▶ **Thesis:** Getting the logical foundations of structuring mechanisms right is worth the considerable effort required, because it can help to avoid soundness bugs.
- ▶ **Question:** Why bother to support LOCAL at all?
Answers:
 1. Supports independent proof development by different users.
 2. LOCAL events are skipped when including a book, which provides potentially large speed-ups in book inclusion.
 3. We want to require that theorems of an included book follow from its axiomatic events. LOCAL events allow us freely to add auxiliary definitions while preserving this requirement.
- ▶ Details are in our new paper. Please feel free to request a preprint (available soon... 😊).

Random Closing Remarks

- ▶ **Thesis:** Getting the logical foundations of structuring mechanisms right is worth the considerable effort required, because it can help to avoid soundness bugs.
- ▶ **Question:** Why bother to support LOCAL at all?
Answers:
 1. Supports independent proof development by different users.
 2. LOCAL events are skipped when including a book, which provides potentially large speed-ups in book inclusion.
 3. We want to require that theorems of an included book follow from its axiomatic events. LOCAL events allow us freely to add auxiliary definitions while preserving this requirement.
- ▶ Details are in our new paper. Please feel free to request a preprint (available soon....).