Advances in ACL2 Proof Debugging Tools

Matt Kaufmann
J Moore

UT Austin (retired)

This research was developed with funding from the Defense Advanced Research Projects Agency (DARPA). The views, opinions and/or findings expressed are those of the authors and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government.
Released under Distribution Statement "A" (Approved for Public Release, Distribution Unlimited).

November, 2023

TALK OVERVIEW

Proofs often fail; need debugging tools! In this talk:

TALK OVERVIEW

Proofs often fail; need debugging tools! In this talk:

- **Break-rewrite**
  - Around since Version 1.3 (early 1990s), but recent improvements include the addition of *near-miss criteria*

TALK OVERVIEW

Proofs often fail; need debugging tools! In this talk:

- **Break-rewrite**
  - Around since Version 1.3 (early 1990s), but recent improvements include the addition of *near-miss criteria*
- **`With-brr-data`** together with associated queries
  - Shows how rewriting produced a surprising term in a **checkpoint** from a failed proof

TODAY

This talk will consist largely of *demos* based on the supporting materials (`books/workshops/2023/kaufmann-moore`), which in turn follow the paper.

TODAY

This talk will consist largely of *demos* based on the supporting materials (`books/workshops/2023/kaufmann-moore`), which in turn follow the paper.

See the documentation for more details, and see the paper for implementation aspects.

WITH-BRR-DATA

Start by collecting data, for example:

```
(with-brr-data (defthm ...))
(with-brr-data (progn ...))
(with-brr-data (define ...))
```

WITH-BRR-DATA

Start by collecting data, for example:

```
(with-brr-data (defthm ...))
(with-brr-data (progn ...))
(with-brr-data (define ...))
```

Then query for source of a subterm or term, e.g.:

```
(cw-gstack-for-subterm (foo (bar x)))
(cw-gstack-for-subterm* (foo (bar x)))
(cw-gstack-for-term (foo (bar x)))
(cw-gstack-for-term* (foo (bar x)))
```

WITH-BRR-DATA

Start by collecting data, for example:

```
(with-brr-data (defthm ...))
(with-brr-data (progn ...))
(with-brr-data (define ...))
```

Then query for source of a subterm or term, e.g.:

```
(cw-gstack-for-subterm (foo (bar x)))
(cw-gstack-for-subterm* (foo (bar x)))
(cw-gstack-for-term (foo (bar x)))
(cw-gstack-for-term* (foo (bar x)))
```

Or even, for example:

```
(cw-gstack-for-subterm (:free (v) (foo v)))
```

** DEMO **

of `with-brr-data` and its query utilities

BREAK-REWRITE

- The break-rewrite utilities help to answer the question:
  Why did the attempt to apply a certain lemma fail?

BREAK-REWRITE

- The break-rewrite utilities help to answer the question: Why did the attempt to apply a certain lemma fail?
- New after Version 8.5: *Near-miss criteria*, which allow breaks even when a rewrite rule's left-hand side doesn't quite match; see the paper or :DOC **monitor** for documentation.
  - `:lambda` — illustrated by demo
  - `:depth`
  - `:abstraction`

BREAK-REWRITE

- The break-rewrite utilities help to answer the question: Why did the attempt to apply a certain lemma fail?
- New after Version 8.5: *Near-miss criteria*, which allow breaks even when a rewrite rule's left-hand side doesn't quite match; see the paper or :DOC **monitor** for documentation.
  - `:lambda` — illustrated by demo
  - `:depth`
  - `:abstraction`
- Also new after Version 8.5: several improvements not discussed here (e.g., `:GO!` works now)

** DEMO **

of break-rewrite utilities

Concluding remarks

- `With-brr-data` was built on the same infrastructure that already supported break-rewrite:
  - **`wormholes`**, but with **`wormhole-eval`** rather than **`wormhole`**, for efficiency;
  - uses functions `brkpt1` and `brkpt2`, which were already in the rewriter for entering and exiting break-rewrite; but,
  - data is collected only for "top-level" calls of the rewriter, not during backchaining (technically, `ancestors` is `nil`) — by default, as **attachments** are supported (see the paper).

Concluding remarks

- `With-brr-data` was built on the same infrastructure that already supported break-rewrite:
  - **wormholes**, but with **wormhole-eval** rather than **wormhole**, for efficiency;
  - uses functions `brkpt1` and `brkpt2`, which were already in the rewriter for entering and exiting break-rewrite; but,
  - data is collected only for "top-level" calls of the rewriter, not during backchaining (technically, `ancestors` is `nil`) — by default, as **attachments** are supported (see the paper).
- Break-rewrite has been improved:
  - now supports near misses, particularly useful for debugging rewriting failures that involve **LOOP$** – future work is to allow **attachments** to define near misses; and
  - lots of clean-up, including improvements to the **wormhole** implementation.

Concluding remarks

- `With-brr-data` was built on the same infrastructure that already supported break-rewrite:
  - **wormholes**, but with **wormhole-eval** rather than **wormhole**, for efficiency;
  - uses functions `brkpt1` and `brkpt2`, which were already in the rewriter for entering and exiting break-rewrite; but,
  - data is collected only for "top-level" calls of the rewriter, not during backchaining (technically, `ancestors` is `nil`) — by default, as **attachments** are supported (see the paper).
- Break-rewrite has been improved:
  - now supports near misses, particularly useful for debugging rewriting failures that involve **LOOP$** – future work is to allow **attachments** to define near misses; and
  - lots of clean-up, including improvements to the **wormhole** implementation.
- The two tools can be used together (see the paper).

Again, see the paper and **online documentation** for more information.

Thank you. And we thank DARPA and ForrestHunt, Inc. for the support.