Desperation Heuristics for ACL2

Matt Kaufmann

The University of Texas at Austin

July 12-13, 2014

Outline (and Motivation)

- A problem with linear arithmetic

Outline (and Motivation)

- A problem with linear arithmetic
- Solution: *desperation heuristics*

Outline (and Motivation)

- A problem with linear arithmetic
- Solution: *desperation heuristics*
- Conclusion: Additional desperation heuristics?

Outline (and Motivation)

- A problem with linear arithmetic
- Solution: *desperation heuristics*
- Conclusion: Additional desperation heuristics?

Acknowledgments: Thanks to Robert Krug and J Moore for helpful discussions.

A problem with linear arithmetic (1)

The following example came from J Moore (email, April 11, 2014).

A problem with linear arithmetic (1)

The following example came from J Moore (email, April 11, 2014).

```
(defthm lemma1
  (<= (len (cdr stk)) (len stk))
  :rule-classes :linear)

(defthm lemma2
  (<= (len (nthcdr n stk)) (len stk))
  :rule-classes :linear)
```

A problem with linear arithmetic (1)

The following example came from J Moore (email, April 11, 2014).

```
(defthm lemma1
  (<= (len (cdr stk)) (len stk))
  :rule-classes :linear)

(defthm lemma2
  (<= (len (nthcdr n stk)) (len stk))
  :rule-classes :linear)

(thm (<= (len (cdr (cdr (nthcdr n s))))
         (len s))
     :hints (("Goal" :do-not-induct t)))
```

A problem with linear arithmetic (2)

Informal proof:

A problem with linear arithmetic (2)

Informal proof:

```
(len (cdr (cdr (nthcdr n s))))
```

A PROBLEM WITH LINEAR ARITHMETIC (2)

Informal proof:

```
(len (cdr (cdr (nthcdr n s))))

<= {by lemma1: (<= (len (cdr stk)) (len stk))}
```

A PROBLEM WITH LINEAR ARITHMETIC (2)

Informal proof:

```
(len (cdr (cdr (nthcdr n s))))

<= {by lemma1: (<= (len (cdr stk)) (len stk))}

(len (cdr (nthcdr n s)))
```

A PROBLEM WITH LINEAR ARITHMETIC (2)

Informal proof:

```
(len (cdr (cdr (nthcdr n s))))

<= {by lemma1: (<= (len (cdr stk)) (len stk))}

(len (cdr (nthcdr n s)))

<= {by lemma1: (<= (len (cdr stk)) (len stk))}
```

A PROBLEM WITH LINEAR ARITHMETIC (2)

Informal proof:

```
(len (cdr (cdr (nthcdr n s))))

<= {by lemma1: (<= (len (cdr stk)) (len stk))}

(len (cdr (nthcdr n s)))

<= {by lemma1: (<= (len (cdr stk)) (len stk))}

(len (nthcdr n s))
```

A PROBLEM WITH LINEAR ARITHMETIC (2)

Informal proof:

```
(len (cdr (cdr (nthcdr n s))))

<= {by lemma1: (<= (len (cdr stk)) (len stk))}

(len (cdr (nthcdr n s)))

<= {by lemma1: (<= (len (cdr stk)) (len stk))}

(len (nthcdr n s))

<= {by lemma2: (<= (len (nthcdr n stk)) (len stk))}
```

A PROBLEM WITH LINEAR ARITHMETIC (2)

Informal proof:

```
(len (cdr (cdr (nthcdr n s))))

<= {by lemma1: (<= (len (cdr stk)) (len stk))}

(len (cdr (nthcdr n s)))

<= {by lemma1: (<= (len (cdr stk)) (len stk))}

(len (nthcdr n s))

<= {by lemma2: (<= (len (nthcdr n stk)) (len stk))}

(len stk)
```

A problem with linear arithmetic (3)

- Sadly, the proof fails in ACL2 Version 6.4.

A problem with linear arithmetic (3)

- Sadly, the proof fails in ACL2 Version 6.4.
- Problem: linear arithmetic needs to be applied before `len` and `nthcdr` have opened up.

A problem with linear arithmetic (3)

- Sadly, the proof fails in ACL2 Version 6.4.
- Problem: linear arithmetic needs to be applied before `len` and `nthcdr` have opened up.
- Robert Krug suggested extending an existing heuristic for 'linearizing' the *unrewritten* conclusion of a linear lemma.

A problem with linear arithmetic (3)

- Sadly, the proof fails in ACL2 Version 6.4.
- Problem: linear arithmetic needs to be applied before `len` and `nthcdr` have opened up.
- Robert Krug suggested extending an existing heuristic for 'linearizing' the *unrewritten* conclusion of a linear lemma.
- Spent many hours creating perhaps 30 patch files:

A problem with linear arithmetic (3)

- Sadly, the proof fails in ACL2 Version 6.4.
- Problem: linear arithmetic needs to be applied before `len` and `nthcdr` have opened up.
- Robert Krug suggested extending an existing heuristic for 'linearizing' the *unrewritten* conclusion of a linear lemma.
- Spent many hours creating perhaps 30 patch files:
  - Focused on $\sim$ 20 relevant lemmas from regression suite

A problem with linear arithmetic (3)

- Sadly, the proof fails in ACL2 Version 6.4.
- Problem: linear arithmetic needs to be applied before `len` and `nthcdr` have opened up.
- Robert Krug suggested extending an existing heuristic for 'linearizing' the *unrewritten* conclusion of a linear lemma.
- Spent many hours creating perhaps 30 patch files:
  - Focused on $\sim$ 20 relevant lemmas from regression suite
  - Found two lemmas that pushed me in opposite directions

A problem with linear arithmetic (3)

- Sadly, the proof fails in ACL2 Version 6.4.
- Problem: linear arithmetic needs to be applied before `len` and `nthcdr` have opened up.
- Robert Krug suggested extending an existing heuristic for 'linearizing' the *unrewritten* conclusion of a linear lemma.
- Spent many hours creating perhaps 30 patch files:
  - Focused on $\sim$ 20 relevant lemmas from regression suite
  - Found two lemmas that pushed me in opposite directions
  - Decided on a radically different solution

SOLUTION: DESPERATION HEURISTICS (1)

The waterfall *clause processors* (see :DOC waterfall):

```
apply-top-hints-clause
preprocess-clause
simplify-clause
settled-down-clause
eliminate-destructors-clause
fertilize-clause
generalize-clause
eliminate-irrelevance-clause
push-clause
```

SOLUTION: DESPERATION HEURISTICS (1)

The waterfall *clause processors* (see :DOC waterfall):

```
apply-top-hints-clause
preprocess-clause
simplify-clause
settled-down-clause
eliminate-destructors-clause
fertilize-clause
generalize-clause
eliminate-irrelevance-clause
push-clause
```

Each application of one of these clause processors either *hits* or *misses* on a goal (clause).

Solution: Desperation Heuristics (1)

The waterfall *clause processors* (see :DOC waterfall):

```
apply-top-hints-clause
preprocess-clause
simplify-clause
settled-down-clause
eliminate-destructors-clause
fertilize-clause
generalize-clause
eliminate-irrelevance-clause
push-clause
```

Each application of one of these clause processors either *hits* or *misses* on a goal (clause).

- *Hits*: Then each resulting subgoal is processed starting at the top of the waterfall.
- *Misses*: Then go on to the next clause processor.

SOLUTION: DESPERATION HEURISTICS (2)

```
apply-top-hints-clause
preprocess-clause
simplify-clause
settled-down-clause
eliminate-destructors-clause ...
```

SOLUTION: DESPERATION HEURISTICS (2)

```
apply-top-hints-clause
preprocess-clause
simplify-clause
settled-down-clause
eliminate-destructors-clause ...
```

`Settled-down-clause` hits if it hasn't already hit on an ancestor clause.

SOLUTION: DESPERATION HEURISTICS (2)

```
apply-top-hints-clause
preprocess-clause
simplify-clause
settled-down-clause
eliminate-destructors-clause ...
```

`Settled-down-clause` hits if it hasn't already hit on an ancestor clause. If it hits, then from the top of the waterfall:

SOLUTION: DESPERATION HEURISTICS (2)

```
apply-top-hints-clause
preprocess-clause
simplify-clause
settled-down-clause
eliminate-destructors-clause ...
```

`Settled-down-clause` hits if it hasn't already hit on an ancestor clause. If it hits, then from the top of the waterfall:

- `Apply-top-hints-clause` and `preprocess-clause` both miss.

SOLUTION: DESPERATION HEURISTICS (2)

```
apply-top-hints-clause
preprocess-clause
simplify-clause
settled-down-clause
eliminate-destructors-clause ...
```

`Settled-down-clause` hits if it hasn't already hit on an ancestor clause. If it hits, then from the top of the waterfall:

- `Apply-top-hints-clause` and `preprocess-clause` both miss.
- What does `simplify-clause` do?

SOLUTION: DESPERATION HEURISTICS (2)

```
apply-top-hints-clause
preprocess-clause
simplify-clause
settled-down-clause
eliminate-destructors-clause ...
```

`Settled-down-clause` hits if it hasn't already hit on an ancestor clause. If it hits, then from the top of the waterfall:

- `Apply-top-hints-clause` and `preprocess-clause` both miss.
- What does `simplify-clause` do?
  - **Formerly**: typically nothing, except during induction.

SOLUTION: DESPERATION HEURISTICS (2)

```
apply-top-hints-clause
preprocess-clause
simplify-clause
settled-down-clause
eliminate-destructors-clause ...
```

`Settled-down-clause` hits if it hasn't already hit on an ancestor clause. If it hits, then from the top of the waterfall:

- `Apply-top-hints-clause` and `preprocess-clause` both miss.
- What does `simplify-clause` do?
  - **Formerly**: typically nothing, except during induction.
  - **New**: Simplify, using **both** the rewritten and unrewritten conclusion of any linear lemma.

SOLUTION: DESPERATION HEURISTICS (2)

```
apply-top-hints-clause
preprocess-clause
simplify-clause
settled-down-clause
eliminate-destructors-clause ...
```

`Settled-down-clause` hits if it hasn't already hit on an ancestor clause. If it hits, then from the top of the waterfall:

- `Apply-top-hints-clause` and `preprocess-clause` both miss.
- What does `simplify-clause` do?
  - **Formerly**: typically nothing, except during induction.
  - **New**: Simplify, using **both** the rewritten and unrewritten conclusion of any linear lemma.

Measured cost to regression suite: about 2%.

Conclusion: Additional desperation heuristics?

Motivation to give this talk:

Conclusion: Additional desperation heuristics?

Motivation to give this talk:

I'd be interested in other potential desperation heuristics.

Conclusion: Additional desperation heuristics?

Motivation to give this talk:

I'd be interested in other potential desperation heuristics.

- Could be very cheap to add more desperation heuristics. (The 2% performance hit is already there.)

Conclusion: Additional desperation heuristics?

Motivation to give this talk:

I'd be interested in other potential desperation heuristics.

- Could be very cheap to add more desperation heuristics. (The 2% performance hit is already there.)
- Maybe you'll notice situations where such automation can complete some proofs.