DefunT: A Tool for Automating Termination Proofs by Using the Community Books (Extended Abstract)

Matt Kaufmann

UT Austin

November 6, 2018

SUMMARY

`defunT`:

- (**defun** with auto-**T**ermination)
- A tool that can automate ACL2 proofs of measure (termination) conjectures

SUMMARY

`defunT`:

- (**defun** with auto-**T**ermination)
- A tool that can automate ACL2 proofs of measure (termination) conjectures

`GOALS for this talk`:

- introduce this tool to potential users,
- explain some of its implementation, and
- advertise for research collaborators to improve the tool.

SUMMARY

`defunT`:

- (**defun** with auto-**T**ermination)
- A tool that can automate ACL2 proofs of measure (termination) conjectures

`GOALS for this talk`:

- introduce this tool to potential users,
- explain some of its implementation, and
- advertise for research collaborators to improve the tool. (Well, that's what it says in the paper; actually I'd like someone to **take over** the tool.)

SUMMARY

`defunT`:

- (**defun** with auto-**T**ermination)
- A tool that can automate ACL2 proofs of measure (termination) conjectures

`GOALS for this talk`:

- introduce this tool to potential users,
- explain some of its implementation, and
- advertise for research collaborators to improve the tool. (Well, that's what it says in the paper; actually I'd like someone to **take over** the tool.)

Relevant files are in `books/kestrel/auto-termination/`

SUMMARY

`defunT`:

- (**defun** with auto-Termination)
- A tool that can automate ACL2 proofs of measure (termination) conjectures

`GOALS for this talk`:

- introduce this tool to potential users,
- explain some of its implementation, and
- advertise for research collaborators to improve the tool. (Well, that's what it says in the paper; actually I'd like someone to **take over** the tool.)

Relevant files are in `books/kestrel/auto-termination/` (archival version in `books/workshops/2018/kaufmann/`).

Running Example

I'll use a running example:

- Start with an edited log.
- Drill down to get a high-level sense of the implementation.

```
ACL2 !>(include-book "kestrel/auto-termination/defunt-top"
                     :dir :system)
[[.. output elided ..]]
ACL2 !>
```

```
ACL2 !>(include-book "kestrel/auto-termination/defunt-top"
                     :dir :system)
[[.. output elided ..]]
ACL2 !>(defunt f3 (x y)
         (if (consp x)
             (if (atom y)
                 (list (f3 (cddr x) y) (f3 (cadr x) y))
               (f3 (cdr x) y))
           (list x y)))
```

```
ACL2 !>(include-book "kestrel/auto-termination/defunt-top"
                     :dir :system)
[[.. output elided ..]]
ACL2 !>(defunt f3 (x y)
         (if (consp x)
             (if (atom y)
                 (list (f3 (cddr x) y) (f3 (cadr x) y))
               (f3 (cdr x) y))
           (list x y)))

*Defunt note*: Using termination theorems for
SYMBOL-BTREE-TO-ALIST-AUX, EVENS and TRUE-LISTP.
```

```
ACL2 !>(include-book "kestrel/auto-termination/defunt-top"
                     :dir :system)
[[.. output elided ..]]
ACL2 !>(defunt f3 (x y)
         (if (consp x)
             (if (atom y)
                 (list (f3 (cddr x) y) (f3 (cadr x) y))
               (f3 (cdr x) y))
           (list x y)))

*Defunt note*: Using termination theorems for
SYMBOL-BTREE-TO-ALIST-AUX, EVENS and TRUE-LISTP.

*Defunt note*: Evaluating
(LOCAL (INCLUDE-BOOK "misc/symbol-btree" :DIR :SYSTEM))
to define function SYMBOL-BTREE-TO-ALIST-AUX.
```

```
ACL2 !>(include-book "kestrel/auto-termination/defunt-top"
                     :dir :system)
[[.. output elided ..]]
ACL2 !>(defunt f3 (x y)
         (if (consp x)
             (if (atom y)
                 (list (f3 (cddr x) y) (f3 (cadr x) y))
               (f3 (cdr x) y))
           (list x y)))

*Defunt note*: Using termination theorems for
SYMBOL-BTREE-TO-ALIST-AUX, EVENS and TRUE-LISTP.

*Defunt note*: Evaluating
(LOCAL (INCLUDE-BOOK "misc/symbol-btree" :DIR :SYSTEM))
to define function SYMBOL-BTREE-TO-ALIST-AUX.

*Defunt note*: Concluded local include-books.

 F3
ACL2 !>
```

```
ACL2 !>:trans1 (defunt f3 (x y)
                 (if (consp x)
                     (if (atom y)
                         (list (f3 (cddr x) y)
                               (f3 (cadr x) y))
                       (f3 (cdr x) y))
                   (list x y)))
```

```
ACL2 !>:trans1 (defunt f3 (x y)
                 (if (consp x)
                     (if (atom y)
                         (list (f3 (cddr x) y)
                               (f3 (cadr x) y))
                       (f3 (cdr x) y))
                   (list x y)))
 (WITH-OUTPUT :OFF :ALL :ON ERROR :GAG-MODE NIL :STACK :PUSH
  (MAKE-EVENT
   (CREATE-DEFUNT
    '(F3 (X Y)
         (IF (CONSP X)
             (IF (ATOM Y)
                 (LIST (F3 (CDDR X) Y) (F3 (CADR X) Y))
                 (F3 (CDR X) Y))
             (LIST X Y)))
    T '(DEFUNT . F3) STATE)
   :ON-BEHALF-OF :QUIET!))
ACL2 !>
```

```
ACL2 !>(CREATE-DEFUNT
        '(F3 (X Y)
             (IF (CONSP X)
                 (IF (ATOM Y)
                     (LIST (F3 (CDDR X) Y) (F3 (CADR X) Y))
                     (F3 (CDR X) Y))
                 (LIST X Y)))
        T '(DEFUNT . F3) STATE)
```

```
ACL2 !>(CREATE-DEFUNT
        '(F3 (X Y)
             (IF (CONSP X)
                 (IF (ATOM Y)
                     (LIST (F3 (CDDR X) Y) (F3 (CADR X) Y))
                     (F3 (CDR X) Y))
                 (LIST X Y)))
        T '(DEFUNT . F3) STATE)
 (PROGN
  (ENCAPSULATE
   NIL
   [[.. Events for printing and locally including a book ..]]
   [[.. Local defthm events ..]]
   (DEFUN F3 (X Y)
     (DECLARE (XARGS :MEASURE (ACL2-COUNT X)
                     :HINTS (("Goal"
                              :BY (:FUNCTIONAL-INSTANCE
                                   F3-TERMINATION-LEMMA-3
                                   (TD-STUB-2 F3))))))
     (IF (CONSP X) ...)))
  (DEFUNT-NOTE "" T)
  (VALUE-TRIPLE 'F3))
ACL2 !>
```

Events for printing and locally including a book

```
(DEFUNT-NOTE
 (MSG
     "Using termination theorem~#0~[~/s~] for ~&0."
     '(SYMBOL-BTREE-TO-ALIST-AUX EVENS TRUE-LISTP)))
(DEFUNT-NOTE
     (MSG "Evaluating ~x0~|to define function ~x1."
          '(LOCAL (INCLUDE-BOOK "misc/symbol-btree"
                                :DIR :SYSTEM))
          'SYMBOL-BTREE-TO-ALIST-AUX))
(LOCAL (INCLUDE-BOOK "misc/symbol-btree"
                     :DIR :SYSTEM))
(DEFUNT-NOTE (MSG "Concluded local include-books."))
```

Local defthm events

```
(LOCAL
 (DEFTHM F3-TERMINATION-LEMMA-1-SYMBOL-BTREE-TO-ALIST-AUX ...)
(LOCAL
 (DEFTHM F3-TERMINATION-LEMMA-2-SYMBOL-BTREE-TO-ALIST-AUX ...)
(LOCAL (DEFTHM F3-TERMINATION-LEMMA-1-EVENS ...))
(LOCAL (DEFTHM F3-TERMINATION-LEMMA-2-EVENS ...))
(LOCAL (DEFTHM F3-TERMINATION-LEMMA-1-TRUE-LISTP ...))
(LOCAL (DEFTHM F3-TERMINATION-LEMMA-2-TRUE-LISTP ...))
(LOCAL
 (DEFTHM F3-TERMINATION-LEMMA-3
   [[.. termination theorem for F3 ..]]
   :HINTS
   (("Goal"
     :USE (F3-TERMINATION-LEMMA-2-SYMBOL-BTREE-TO-ALIST-AUX
           F3-TERMINATION-LEMMA-2-EVENS
           F3-TERMINATION-LEMMA-2-TRUE-LISTP)
     :IN-THEORY (THEORY 'AUTO-TERMINATION-FNS)))))
```

```
(LOCAL
 (DEFTHM F3-TERMINATION-LEMMA-1-EVENS
   (IF (O-P (ACL2-COUNT L))
       (IF (NOT (CONSP L))
           'T
           (O< (ACL2-COUNT (CDR (CDR L)))
               (ACL2-COUNT L)))
       'NIL)
   :HINTS (("Goal"
             :USE ((:TERMINATION-THEOREM EVENS
                    ((EVENS TD-STUB-1))))
             :IN-THEORY (THEORY 'AUTO-TERMINATION-FNS)))))
```

```
(LOCAL
 (DEFTHM F3-TERMINATION-LEMMA-1-EVENS
   (IF (O-P (ACL2-COUNT L))
       (IF (NOT (CONSP L))
           'T
           (O< (ACL2-COUNT (CDR (CDR L)))
               (ACL2-COUNT L)))
       'NIL)
   :HINTS (("Goal"
            :USE ((:TERMINATION-THEOREM EVENS
                                        ((EVENS TD-STUB-1))))
            :IN-THEORY (THEORY 'AUTO-TERMINATION-FNS)))))

(LOCAL
 (DEFTHM F3-TERMINATION-LEMMA-2-EVENS
   (IF (NOT (CONSP X))
       'T
       (IF (CONSP Y)
           'T
           (O< (ACL2-COUNT (CDR (CDR X)))
               (ACL2-COUNT X))))
   :HINTS (("Goal" :BY F3-TERMINATION-LEMMA-1-EVENS))))
```

Putting it all together:

```
(LOCAL
 (DEFTHM F3-TERMINATION-LEMMA-3
   [[.. termination theorem for F3 ..]]
   :HINTS
   (("Goal"
     :USE (F3-TERMINATION-LEMMA-2-SYMBOL-BTREE-TO-ALIST-AUX
           F3-TERMINATION-LEMMA-2-EVENS
           F3-TERMINATION-LEMMA-2-TRUE-LISTP)
     :IN-THEORY (THEORY 'AUTO-TERMINATION-FNS)))))
```

```
(LOCAL
 (DEFTHM F3-TERMINATION-LEMMA-3
  (IF (O-P (ACL2-COUNT X))
      (IF (IF (NOT (CONSP X))
              'T
              (IF (NOT (ATOM Y))
                  'T
                  (O< (ACL2-COUNT (CDR (CDR X)))
                      (ACL2-COUNT X))))
          (IF (IF (NOT (CONSP X))
                  'T
                  (IF (NOT (ATOM Y))
                      'T
                      (O< (ACL2-COUNT (CAR (CDR X)))
                          (ACL2-COUNT X))))
              (IF (NOT (CONSP X))
                  'T
                  (IF (ATOM Y)
                      'T
                      (O< (ACL2-COUNT (CDR X))
                          (ACL2-COUNT X))))
              'NIL)
          'NIL)
      'NIL)

  :HINTS
  (("Goal"
    :USE (F3-TERMINATION-LEMMA-2-SYMBOL-BTREE-TO-ALIST-AUX
          F3-TERMINATION-LEMMA-2-EVENS
          F3-TERMINATION-LEMMA-2-TRUE-LISTP)
    :IN-THEORY (THEORY 'AUTO-TERMINATION-FNS)))))
```

The Database

QUESTION

But where did the tool find the termination theorems to use?

The Database

QUESTION

But where did the tool find the termination theorems to use?

ANSWER:

The *termination database candidates* file,
`td-cands.lisp`, which come from `defun` forms.

The Database

QUESTION

But where did the tool find the termination theorems to use?

ANSWER:

The *termination database candidates* file, `td-cands.lisp`, which come from `defun` forms.

- It is generated by invoking the script `write-td-cands.sh`, which:

The Database

QUESTION

But where did the tool find the termination theorems to use?

ANSWER:

The *termination database candidates* file, `td-cands.lisp`, which come from `defun` forms.

- It is generated by invoking the script `write-td-cands.sh`, which:
  - includes the book `books/doc/top.lisp` (to include `defun` forms from all books that support bulding the manual);

The Database

QUESTION

But where did the tool find the termination theorems to use?

ANSWER:

The *termination database candidates* file, `td-cands.lisp`, which come from `defun` forms.

- It is generated by invoking the script `write-td-cands.sh`, which:
  - includes the book `books/doc/top.lisp` (to include `defun` forms from all books that support bulding the manual);
  - includes the database-building book, `termination-database.lisp`; then

The Database

QUESTION

But where did the tool find the termination theorems to use?

ANSWER:

The *termination database candidates* file, `td-cands.lisp`, which come from `defun` forms.

- It is generated by invoking the script `write-td-cands.sh`, which:
  - includes the book `books/doc/top.lisp` (to include `defun` forms from all books that support bulding the manual);
  - includes the database-building book, `termination-database.lisp`; then
  - writes out `td-cands.lisp` and (for necessary packages) `td-cands.acl2`.

Some Engineering Considerations

Some Engineering Considerations

- Generated lemmas are carefully orchestrated.

Some Engineering Considerations

- Generated lemmas are carefully orchestrated.
- Store each termination scheme as a set of *clauses* (disjunctions)

Some Engineering Considerations

- Generated lemmas are carefully orchestrated.
- Store each termination scheme as a set of *clauses* (disjunctions)
  - in *simplified* form, e.g., replacing `(endp x)` by `(not (consp x))` and expanding lambda applications (beta reduction);

Some Engineering Considerations

- Generated lemmas are carefully orchestrated.
- Store each termination scheme as a set of *clauses* (disjunctions)
  - in *simplified* form, e.g., replacing `(endp x)` by `(not (consp x))` and expanding lambda applications (beta reduction);
  - using *subsumption* to minimize database size;

Some Engineering Considerations

- Generated lemmas are carefully orchestrated.
- Store each termination scheme as a set of *clauses* (disjunctions)
  - in *simplified* form, e.g., replacing `(endp x)` by `(not (consp x))` and expanding lambda applications (beta reduction);
  - using *subsumption* to minimize database size;
  - during the search, using subsumption tailored to termination theorem clause sets; and

Some Engineering Considerations

- Generated lemmas are carefully orchestrated.
- Store each termination scheme as a set of *clauses* (disjunctions)
  - in *simplified* form, e.g., replacing `(endp x)` by `(not (consp x))` and expanding lambda applications (beta reduction);
  - using *subsumption* to minimize database size;
  - during the search, using subsumption tailored to termination theorem clause sets; and
  - filtering clauses with limits on both the number of function symbols and the size.

Some Engineering Considerations

- Generated lemmas are carefully orchestrated.
- Store each termination scheme as a set of *clauses* (disjunctions)
  - in *simplified* form, e.g., replacing `(endp x)` by `(not (consp x))` and expanding lambda applications (beta reduction);
  - using *subsumption* to minimize database size;
  - during the search, using subsumption tailored to termination theorem clause sets; and
  - filtering clauses with limits on both the number of function symbols and the size.
- Make (up to) two passes, first restricting to functions defined in the current world.

Some Engineering Considerations

- Generated lemmas are carefully orchestrated.
- Store each termination scheme as a set of *clauses* (disjunctions)
  - in *simplified* form, e.g., replacing `(endp x)` by `(not (consp x))` and expanding lambda applications (beta reduction);
  - using *subsumption* to minimize database size;
  - during the search, using subsumption tailored to termination theorem clause sets; and
  - filtering clauses with limits on both the number of function symbols and the size.
- Make (up to) two passes, first restricting to functions defined in the current world.
- Limit the number of injections allowed from a candidate's measured subset to the new formals.

Concluding Remarks and Future Work

Concluding Remarks and Future Work

Much more about the algorithms is discussed in the `README` file in the directory, `books/kestrel/auto-termination/`.

Concluding Remarks and Future Work

Much more about the algorithms is discussed in the `README` file in the directory, `books/kestrel/auto-termination/`.

In spite of making two passes, ACL2 reports only 0.04 seconds taken altogether for the example in this talk (and paper), using a 2014 MacBook Pro.

Concluding Remarks and Future Work

Much more about the algorithms is discussed in the `README` file in the directory, `books/kestrel/auto-termination/`.

In spite of making two passes, ACL2 reports only 0.04 seconds taken altogether for the example in this talk (and paper), using a 2014 MacBook Pro.

But there is probably a lot more to do to make `defunt` widely useful. The file `to-do.txt` in the directory above has 26 tasks to consider.

Concluding Remarks and Future Work

Much more about the algorithms is discussed in the `README` file in the directory, `books/kestrel/auto-termination/`.

In spite of making two passes, ACL2 reports only 0.04 seconds taken altogether for the example in this talk (and paper), using a 2014 MacBook Pro.

But there is probably a lot more to do to make `defunt` widely useful. The file `to-do.txt` in the directory above has 26 tasks to consider.

I'd be thrilled for someone to take ownership of this tool!