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

Matt Kaufmann
Department of Computer Science
The University of Texas at Austin, Austin, TX, USA
kaufmann@cs.utexas.edu

We present a tool that automates termination proofs for recursive definitions by mining existing termination theorems.

The macro, `defunT` (**defun** with auto-**T**ermination), is a tool that can automate ACL2 proofs of termination (i.e., of measure conjectures). This note has three goals: to introduce this tool to potential users, to explain some of its implementation, and to advertise for research collaborators to improve the tool. The tool suite resides in community books directory `kestrel/auto-termination/`.[1]

`DefunT` relies on a database of already-proved termination theorems, each stored as a list of clauses (disjunctions). That database is generated by the script file `write-td-cands.sh`, which writes it to the generated book, `td-cands.lisp`, and creates an associated file, `td-cands.acl2`. This script computes the database after it includes the book `doc/top.lisp`, which in turn includes many of the community books (to build the manual), using algorithms implemented in the book, `termination-database.lisp`. The book `td-cands.lisp` will likely only need to be regenerated infrequently; but it is routinely certified by the build system on top of a world obtained by executing the 45 `include-book` events in `td-cands.acl2`, which define all necessary packages so that ACL2 can read all forms in the book.

We explain `defunT` — both its use and a little about its implementation — by focusing on the following example, which creates three distinct proof goals for termination: one for each recursive call. The book `defunt-top.lisp` includes both the database, `td-cands.lisp`, and the implementation of the `defunT` macro, `defunt.lisp`.

```
(include-book "kestrel/auto-termination/defunt-top" :dir :system)
(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)))
```

The output shown below notes that `defunT` finds three helpful termination theorems in the database, `td-cands.lisp`. Each of these suffices to prove one of the three goals with a `:termination-theorem` lemma-instance, where one of those three requires a book to be included.

```
*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.
```

[1] An archival version, from the time this paper was written, is under `books/workshops/2018/kaufmann/`.

Shilpi Goel and Matt Kaufmann (Eds.):
ACL2 Theorem Prover and its Applications (ACL2-2018)
EPTCS 280, 2018, pp. 161–163, doi:10.4204/EPTCS.280.12

The defunT macro uses make-event to do the search and to generate a suitable event as displayed below. The search can make two passes through the database, where the first pass only considers functions defined in the current session. In this example, a local include-book is generated because the first pass was not sufficient. In spite of making both passes, ACL2 reports only 0.04 seconds taken altogether, using a 2014 MacBook Pro.

```
(ENCAPSULATE ()
  ;; The following book is necessary, as noted in the output shown above.
  (LOCAL (INCLUDE-BOOK "misc/symbol-btree" :DIR :SYSTEM))
  ;; Six local defthm events are omitted here.  The seventh has the following form:
  (LOCAL (DEFTHM new-termination-theorem
           <termination theorem for f3 generated from found measure etc.>
           :HINTS (("Goal" :USE <..elided here..> :IN-THEORY (THEORY 'AUTO-TERMINATION-FNS)))))
  (DEFUN F3 (X Y)
    (DECLARE (XARGS :MEASURE (ACL2-COUNT X)
                    :HINTS (("Goal" :BY (:FUNCTIONAL-INSTANCE new-termination-theorem
                                                              (binary-stub-function F3))))))
    (IF (CONSP X)
        (IF (ATOM Y)
            (LIST (F3 (CDDR X) Y) (F3 (CADR X) Y))
            (F3 (CDR X) Y))
        (LIST X Y))))
```

A key aspect of defunT is that termination theorem clause-lists are stored in *simplified* form: thus, an old clause-list can subsume a new clause even when function bodies have minor differences, such as (if (endp x) ...) vs. (if (not (consp x)) ...). Also, the generated local theorems are carefully instrumented to make proofs fast and automatic. The flow is as follows (here, restricting to the case of a single old termination theorem), where *old* and *new* are old and new termination theorems, and old_s and new_s are their simplifications: *new* follows with a :use hint from new_s, which follows with a :by hint from old_s, which follows with a :use hint from *old*. The :by hint has two advantages over a corresponding :use hint: it avoids the need to supply a substitution (when the old and new functions have different formals), and it avoids if-splitting into clauses (goals). The :by hint succeeds because it employs essentially the same subsumption test as is used during the search for an old termination theorem to prove the new termination goal. The :use hints are accompanied by :in-theory hints that can be expected to make those proofs fast, by restricting to the small theories used for clause-list simplification. Stub functions replace functions called in their own termination schemes, to enhance subsumption.

Conclusion. Program termination is a rich field [1]. The goal of defunT is, however, simply to make it convenient to prove termination automatically when using ACL2. An extension of ACL2 with CCG analysis [2] can prove termination automatically; unlike that approach, defunT generates a measure for ACL2's usual termination analysis. J Moore's tool Terminatricks [3] is a different step towards that goal: while that tool does not use the defunT approach of taking advantage of the community books, it can however incrementally extend its database of termination theorems. This potential enhancement to defunT is discussed in file to-do.txt, as are more than 20 others. Further implementation-level details may be found in the README, which for example explains database organization by *justification* (which includes a measure), as well as several optimizations, such as the use of subsumption to restrict the database to 643 distinct termination schemes essentially shared by 821 functions. Others are invited to contribute to the enhancement of defunT!

Acknowledgments. This material is based upon work supported in part by DARPA under Contract No. FA8750-15-C-0007. We thank Eric Smith for encouragement to work on this problem.

References

[1] Byron Cook, Andreas Podelski & Andrey Rybalchenko (2011): *Proving Program Termination*. *Commun. ACM* 54(5), pp. 88–98. Available at http://doi.acm.org/10.1145/1941487.1941509.

[2] Matt Kaufmann, Panagiotis Manolios, J Strother Moore & Daron Vroon (2006): *Integrating CCG analysis into ACL2*. In: *Workshop Proceedings: WST 2006, Eighth International Workshop on Termination*, pp. 64–68. Available at http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.97.8994&rep=rep1&type=pdf.

[3] J Moore (Accessed: 2018): *Terminatricks*. https://github.com/acl2/acl2/tree/master/books/projects/codewalker/terminatricks.lisp.