Knuth's Generalization of Takeuchi's Tarai Function: Preliminary Report

Tom Bailey John Cowles
{tbailey, cowles}@uwyo.edu

Department of Computer Science
University of Wyoming
Laramie, WY 82071.

Abstract

Donald E. Knuth of Stanford University raises, in [2] and [3, chapter 22], intriguing open questions about his generalization of the tarai function and proposes an interesting candidate for machine verification. We answer some of the open questions and explore the use of ACL2 to meet Knuth's challenge.

1 Takeuchi's Tarai Function

Ikuo Takeuchi devised the following recursive function for benchmarking LISP systems. The recursion can take a long time to terminate without generating large intermediate numerical values. Knuth comments [3, chapter 22]: "Takeuchi called his function *tarai*, from the word 'taraimawashi,' which connotes passing an unpleasant object from one person to another."

For integer inputs, x, y, z,

$$t(x,y,z) \stackrel{\text{def}}{\Leftarrow} \text{ if } x \leq y \text{ then } y \text{ else } t(t(x-1,y,z), t(y-1,z,x), t(z-1,x,y)). \tag{1}$$

John McCarthy proved that this recursion terminates and that t can be computed without any recursion,

$$t(x,y,z) = \text{ if } x \leq y \text{ then } y \text{ else if } y \leq z \text{ then } z \text{ else } x. \tag{2}$$

J Moore [5] discovered a simpler measure than the one used by McCarthy and used the early Boyer-Moore theorem prover, THM, to verify termination and that t satisfies the simpler nonrecursive equation.

2 Knuth's Generalization

Knuth generalizes the tarai function to higher dimensions: For integer inputs, $x_1, x_2, \ldots, x_m$,

$$\begin{aligned} t(x_1, x_2, \ldots, x_m) \stackrel{\text{def}}{\Leftarrow} \text{ if } x_1 \le x_2 \text{ then } & x_2 \\ \text{else } & t(\, t(x_1 - 1, x_2, \ldots, x_m), \\ & \quad t(x_2 - 1, x_3, \ldots, x_m, x_1), \\ & \quad \vdots \\ & \quad t(x_m - 1, x_1, \ldots, x_{m-1})). \end{aligned}$$

Knuth raises two questions about this recursive definition.

1. Are there total functions on the integers that satisfy the recursive equations based on the definition? That is, are there total functions $f(x_1, x_2, \ldots, x_m)$ on the integers that satisfy the equation

$$\begin{aligned} f(x_1, x_2, \ldots, x_m) = \text{ if } x_1 \le x_2 \text{ then } & x_2 \\ \text{else } & f(\, f(x_1 - 1, x_2, \ldots, x_m), \\ & \quad f(x_2 - 1, x_3, \ldots, x_m, x_1), \\ & \quad \vdots \\ & \quad f(x_m - 1, x_1, \ldots, x_{m-1}))? \end{aligned} \tag{3}$$

2. Does the recursion terminate for all integer inputs?

If the answer to the second question is yes, then the answer to the first question must also be yes. But, Knuth points out, [2] and [3, chapter 22], "...we have not demonstrated that termination will occur, and there is no obvious ordering on the integer m-tuples $(x_1, \ldots, x_m)$ that will yield such a proof."

Question 1: Can the recursive equation be satisfied?

Knuth notes that when $m = 4$, the function

$$t(x_1, x_2, x_3, x_4) \stackrel{\text{def}}{\Leftarrow} \text{if } x_1 \le x_2 \text{ then } x_2 \\ \text{else if } x_2 \le x_3 \text{ then } x_3 \\ \text{else if } x_3 \le x_4 \text{ then } x_4 \\ \text{else } x_1.$$

satisfies the equation

$$t(x_1, x_2, x_3, x_4) = \text{if } x_1 \le x_2 \text{ then } x_2 \\ \text{else } t(\, t(x_1 - 1, x_2, x_3, x_4), \\ t(x_2 - 1, x_3, x_4, x_1), \\ t(x_3 - 1, x_4, x_1, x_2), \\ t(x_4 - 1, x_1, x_2, x_3)).$$

This, together with similar results for $m = 3$ discussed earlier [see (1) and (2)], makes it natural to conjecture that the m-dimensional equation (3) is satisfied by

$$f(x_1, x_2, \ldots, x_m) \stackrel{\text{def}}{\Leftarrow} \text{if } (\exists k < m)(x_1 > x_2 > \cdots > x_k \le x_{k+1}) \text{ then } x_{k+1} \\ \text{else } x_1.$$

Things are not quite so simple, as shown by the following counterexample, due to Knuth, for $m = 5$. For the f just defined, with inputs $5, 3, 2, 0, 1$, the left side of equation (3) clearly yields $f(5, 3, 2, 0, 1) = 1$, while the right side, after some computation, produces 2.

Knuth modifies the definition of f in the following way.

$$f(x_1, x_2, \ldots, x_m) \stackrel{\text{def}}{\Leftarrow} \text{if } (\exists k < m)(x_1 > x_2 > \cdots > x_k \le x_{k+1}) \\ \text{then } g(x_1, x_2, \ldots, x_{k+1}) \\ \text{else } x_1. \tag{4}$$

Here g is a "function" that takes a variable number (at least two) of integer inputs.

$$g(x_1, x_2, \ldots, x_j) \stackrel{\text{def}}{\Leftarrow} \text{if } j = 2 \text{ then } x_2 \\ \text{else if } x_1 = x_2 + 1 \text{ then } g(x_2, \ldots, x_j) \\ \text{else if } x_2 = x_3 + 1 \text{ then } \max(x_3, x_j) \\ \text{else } x_j.$$

An Open Problem raised in [2] is to: *Prove the following theorem by computer.* Knuth writes in [3, chapter 22], "Indeed, I have checked the proof by hand twice, and I believe it is correct, but I do not want to have to check it again!"

Theorem 1 *The function f defined by (4) satisfies the m-dimensional tarai recurrence (3).*

In February 2000, Tom Bailey found a *counterexample* with $m = 6$: $f(8,6,4,3,1,2) = 2$ while expanding the right side of (3) finds it equal to $f(3,2,3,2,2,8) = 3$. Knuth's reaction [4] to the counterexample: "This is certainly a great way to make me believe in mechanical verification." Knuth further writes in [3, chapter 22] that the theorem "...is apparently correct when $m = 5$, although mechanical verification is still pending (and now imperative!). The behavior of the tarai recurrence in six or more dimensions remains unknown."

We have a proof that has been checked by hand many times for the following (but mechanical verification is not yet complete).

Conjecture 1 *The function f, defined by modifying (4) by replacing the call to $g(x_1, x_2, \ldots, x_{k+1})$ with a call to $g_b(x_1, x_2, \ldots, x_{k+1})$, satisfies the m-dimensional tarai recurrence (3).*

Here g_b, like g, takes a variable number of integer inputs.

$$g_b(x_1, x_2, \ldots, x_j) \stackrel{\text{def}}{\Leftarrow} \text{if } j \le 3 \text{ then } x_j \text{ else if } x_1 = x_2 + 1 \text{ or } x_2 > x_3 + 1 \text{ then } g_b(x_2, \ldots, x_j) \text{ else } \max(x_3, x_j).$$

Question 2: Does the recursion terminate?

Knuth points out, [2] and [3, chapter 22], that the answer to this question *may* very well depend on which recursive calls we insist on fully evaluating: "...a call-by-need technique *will* always terminate when applied to the recursive equation for $t(x_1, \ldots, x_m)$. If $x_1 > x_2 > \cdots > x_k \le x_{k+1}$, the values $y_i = t(x_i - 1, x_{i+1}, \ldots, x_{i-1})$ need be expanded only for $1 \le i \le k+1$, and this will be sufficient to determine the value of $t(y_1, \ldots, y_m) = t(x_1, \ldots, x_m)$ in a finite number of steps." Knuth's argument for this depends on the faulty proof given for Theorem 1.

Mechanical verification for the following is not yet complete.

Conjecture 2 *The recursion for computing $t(x_1, \ldots, x_m)$ always terminates using the following version of Knuth's call-by-need. If $x_1 > x_2 > \cdots > x_k \le x_{k+1}$, it is sufficient to expand the values $y_i = t(x_i - 1, x_{i+1}, \ldots, x_{i-1})$ only for $1 \le i \le k$ (note the change from $k+1$ to k in this range for i), to determine the value of $t(y_1, \ldots, y_m) = t(x_1, \ldots, x_m)$.*

Knuth continues, [2] and [3, chapter 22]: "Therefore we come to a final question, which will perhaps prove to be the most interesting aspect of the present investigation, particularly if it has a negative answer. ... If so, the tarai recurrence would be an extremely interesting example to include in *all* textbooks about recursion."

Open Problem. "Does the m-dimensional tarai recursive equation define a total function, for all $m \ge 3$, if it is expanded fully (without call-by-need)?"

The answer to this open problem was shown, by Tom Bailey, Jim Caldwell, and John Cowles, in January 2000, to be, "no." For $m = 4$, $t(3,2,1,5) = \cdots = t(2,1,5,4) = \cdots = t(1,5,4,t(3,2,1,5))$.

3 Progress Using ACL2

Applying ACL2, in an inelegant way with brute force, verifies the following:

1. For $2 \le m \le 7$, the function f of Conjecture 1 (with Knuth's g replaced with g_b) satisfies the m-dimensional tarai recurrence (3). Thus ACL2 verifies Conjecture 1 for $2 \le m \le 7$.

2. For $2 \le m \le 5$, Knuth's version of f, defined by (4), computes the same values as the version of f given in Conjecture 1 (with g replaced with g_b). Together with item 1, this finishes the mechanical verification that Knuth's f satisfies the recursive equation (3), for $m = 5$ (as well as for $2 \le m \le 4$).

3. For $2 \le m \le 7$, the function f of Conjecture 1 (with Knuth's g replaced with g_b) is the *unique* total function on the integers that satisfies the m-dimensional tarai recurrence (3).

4. For $2 \le m \le 7$, the function f of Conjecture 1 (with Knuth's g replaced with g_b) satisfies the m-dimensional *restricted* tarai recurrence:

$$f(x_1, x_2, \ldots, x_m) \;=\; \text{if } \; x_1 \le x_2 \; \text{ then } \; x_2 \qquad (5)$$
$$\text{else } \; f(\, f(x_1 - 1, x_2, \ldots, x_m), \; f(x_2 - 1, x_3, \ldots, x_m, x_1), \; \vdots \; f(x_k - 1, x_{k'}, \ldots, x_{k-1})).$$

Note the k and k' in the last line of this equation. Here k' stands for $(k+1) \bmod' m$ (where $i \bmod' m$ is the unique $j \in \{1, \ldots, m\}$ such that $i \equiv j \bmod m$) and k is the integer such that $1 \le k \le m$ and $x_1 > x_2 > \cdots > x_k \le x_{k'}$. This definition requires that f, like g and g_b, be a "function" that takes a variable number (at least two) of integer inputs.

5. For $2 \le m \le 7$, the function f of Conjecture 1 (with Knuth's g replaced with g_b) is the *unique* total function on the integers that satisfies the m-dimensional *restricted* tarai recurrence (5).

6. For $2 \le m \le 7$, the recursive calls on the right side of equation (5), defining the m-dimensional *restricted* tarai recurrence, always terminate. Thus ACL2 verifies Conjecture 2 for $2 \le m \le 7$.

Coping with a variable number of inputs

Lisp provides an obvious way of dealing with functions, like g, g_b, and the restricted tarai recurrence, that take a variable number of inputs: Form the inputs into a list and use that list as the single input to the function.

The ACL2 versions of the functions f and g_b mentioned in Conjecture 1 are now straight forward:

```
(defun
    Fb (lst)
    "The input lst is intended to be a nonempty
     list of integers."
    (declare (xargs :guard (and (integer-listp lst)
                                (consp lst))))
    (if (decreasing-p lst)
```

```
        (first lst)
        (Gb (first-non-decrease lst))))

(defun
    Gb (lst)
    "The input lst is intended to be a nonempty
     list of integers."
    (declare (xargs :guard (and (integer-listp lst)
                                (consp lst))))
    (cond ((consp (nthcdr 3 lst))    ;; (len lst) > 3
           (if (or (equal (first lst)
                          (+ (second lst) 1))
                   (> (second lst)
                      (+ (third lst) 1)))
               (Gb (rest lst))
               (max (third lst)
                    (last-el lst))))
          (t (last-el lst))))      ;; (len lst) <= 3
```

Here

(`decreasing-p` $(x_1, x_2, \ldots, x_m)$) returns `true` if and only if $x_1 > x_2 > \cdots > x_m$,

(`first-non-decrease` $(x_1, x_2, \ldots, x_m)$) returns $(x_1, x_2, \ldots, x_k, x_{k+1})$ where k is the index such that $x_1 > x_2 > \cdots > x_k \leq x_{k+1}$,

`(last-el lst)` returns the last element in the list `lst`.

The tarai recurrences are both satisfiable

The following functions provide one way of formally stating in ACL2 that `Fb` satisfies both the tarai (3) and restricted tarai (5) recurrences.

(`Fb-lst lst`) returns the list obtained by applying `Fb` to each element of `lst` (which should be a list of lists),

(`dec-front-len` $(x_1, x_2, \ldots, x_m)$) returns the k such that $x_1 > x_2 > \cdots > x_k \leq x_{k'}$. Here k' equals $(k+1) \bmod' m$.

(`lst-rotates-with-minus-1` n $(x_1, x_2, \ldots, x_m)$) returns the list of the first $n+1$ elements in this list of lists:

$$\begin{array}{c}(x_1 - 1, x_2, \ldots, x_m),\\(x_2 - 1, x_3, \ldots, x_m, x_1),\\\vdots\\(x_m - 1, x_1, \ldots, x_{m-1}),\\(x_1 - 1, x_2, \ldots, x_m),\\\vdots\end{array}$$

The following theorems are proved by exhaustive consideration of cases. The case for $m = 7$, using a machine with a 600 MHz pentium processor, requires time, as reported by ACL2, of over 3.85 hours to complete. The first theorem formally states that for $2 \le m \le 7$, `Fb` satisfies the tarai (3) recurrence and the second says that `Fb` also satisfies the restricted tarai (5) recurrence.

```
(defthm
    Fb-sat-tarai-def
    (implies (and (integer-listp lst)
                  (consp (rest lst))        ;; (len lst) > 1
                  (not (consp (nthcdr 7 lst)))
                  )                         ;; (len lst) <= 7
             (equal (Fb lst)
                    (if (<= (first lst)
                            (second lst))
                        (second lst)
                        (Fb (Fb-lst (lst-rotates-with-minus-1
                                     (- (LEN lst) 1)
                                     lst))))))
    ...)

(defthm
    Fb-sat-tarai-def-a
    (implies (and (integer-listp lst)
                  (consp (rest lst))        ;; (len lst) > 1
                  (not (consp (nthcdr 7 lst)))
                  )                         ;; (len lst) <= 7
             (equal (Fb lst)
```

```
                          (if (<= (first lst)
                                  (second lst))
                              (second lst)
                              (Fb (Fb-lst (lst-rotates-with-minus-1
                                          (- (DEC-FRONT-LEN lst) 1)
                                          lst))))))
        ...)
```

The tarai recurrences are both *uniquely* satisfiable

`Encapsulate`, using `Fb` as the witness, is used to consistently axiomatize four functions `tarai`, `tarai-lst`, `rTarai`, and `rTarai-lst` so that

- `tarai` is a total function that satisfies the axiom obtained by replacing `Fb`, in the theorem, `Fb-sat-tarai-def`, of the previous section, with `tarai`.
- `tarai` returns an integer whenever the input is a list of integers of length 2 or more.
- `rTarai` is a total function that satisfies the axiom obtained by replacing `Fb`, in the theorem, `Fb-sat-tarai-def-a`, of the previous section, with `rTarai`.

Thus the axioms specifically restrict their validity to input lists of lengths 2–7.

The following theorems are proved by cases, one case for each list length from 2–7. Induction is used to prove each case.

```
(defthm
    tarai=Fb
    (implies (and (integer-listp lst)
                  (consp (rest lst))       ;; (len lst) > 1
                  (not (consp (nthcdr 7 lst)))
                  )                        ;; (len lst) <= 7
             (equal (tarai lst)(Fb lst)))
    ...)

(defthm
    rTarai=Fb-7
```

```
(implies (and (integer-listp lst)
              (consp (nthcdr 1 lst)) ;; (len lst) > 1
              (not
               (consp (nthcdr 7 lst))));; (len lst) <= 7
         (equal (rTarai lst)(Fb lst)))
...)
```

The measure of lists of integers $(x_1, x_2, \ldots, x_m)$, used for the induction, is based on the lexicographical ordering on pairs $(k, \mathtt{nfix}(x_1 - x_2))$, where k is the integer such that $x_1 > x_2 > \cdots > x_k \leq x_{k'}$ (k' equals $(k+1)$ mod$'$ m).

For example, here is the induction scheme used to prove `tarai=Fb` when `lst` is the 4 element list `(LIST FIRST SECOND THIRD FOURTH)`.

```
(AND (IMPLIES (NOT (INTEGER-LISTP (LIST FIRST SECOND
                                        THIRD FOURTH)))
              (:P FIRST SECOND THIRD FOURTH))
     (IMPLIES (AND (INTEGER-LISTP (LIST FIRST SECOND
                                        THIRD FOURTH))
                   (<= FIRST SECOND))
              (:P FIRST SECOND THIRD FOURTH))
     (IMPLIES (AND (INTEGER-LISTP (LIST FIRST SECOND
                                        THIRD FOURTH))
                   (< SECOND FIRST)
                   (< THIRD SECOND)
                   (< FOURTH THIRD)
                   (:P (+ -1 FIRST) SECOND THIRD FOURTH)
                   (:P (+ -1 SECOND) THIRD FOURTH FIRST)
                   (:P (+ -1 THIRD) FOURTH FIRST SECOND)
                   (:P (+ -1 FOURTH) FIRST SECOND THIRD)
                   (:P (FB (LIST (+ -1 FIRST) SECOND
                                 THIRD FOURTH))
                       (FB (LIST (+ -1 SECOND) THIRD
                                 FOURTH FIRST))
                       (FB (LIST (+ -1 THIRD) FOURTH
                                 FIRST SECOND))
                       (FB (LIST (+ -1 FOURTH) FIRST
                                 SECOND THIRD))))
              (:P FIRST SECOND THIRD FOURTH))
     (IMPLIES (AND (INTEGER-LISTP (LIST FIRST SECOND
```

```
                                         THIRD FOURTH))
                       (< SECOND FIRST)
                       (< THIRD SECOND)
                       (<= THIRD FOURTH)
                       (:P (+ -1 FIRST) SECOND THIRD FOURTH)
                       (:P (+ -1 SECOND) THIRD FOURTH FIRST)
                       (:P (+ -1 THIRD) FOURTH FIRST SECOND)
                       (:P (FB (LIST (+ -1 FIRST) SECOND
                                     THIRD FOURTH))
                           (FB (LIST (+ -1 SECOND) THIRD
                                     FOURTH FIRST))
                           (FB (LIST (+ -1 THIRD) FOURTH
                                     FIRST SECOND))
                           (TARAI (LIST (+ -1 FOURTH) FIRST
                                        SECOND THIRD)))))
                  (:P FIRST SECOND THIRD FOURTH))
         (IMPLIES (AND (INTEGER-LISTP (LIST FIRST SECOND
                                         THIRD FOURTH))
                       (< SECOND FIRST)
                       (<= SECOND THIRD)
                       (:P (+ -1 FIRST) SECOND THIRD FOURTH)
                       (:P (+ -1 SECOND) THIRD FOURTH FIRST)
                       (:P (FB (LIST (+ -1 FIRST) SECOND
                                     THIRD FOURTH))
                           (FB (LIST (+ -1 SECOND) THIRD
                                     FOURTH FIRST))
                           (TARAI (LIST (+ -1 THIRD) FOURTH
                                        FIRST SECOND))
                           (TARAI (LIST (+ -1 FOURTH) FIRST
                                        SECOND THIRD)))))
                  (:P FIRST SECOND THIRD FOURTH))).
```

Knuth's f matches `Fb` for $2 \leq m \leq 5$

Knuth's version of f is straight forward to formalize.

```
(defun
    Fk (lst)
```

```
    "Knuth's f.
     The input lst is intended to be a nonempty
     list of integers."
    (declare (xargs :guard (and (integer-listp lst)
                                (consp lst))))
    (if (decreasing-p lst)
        (first lst)
        (Gk (first-non-decrease lst))))

(defun
    Gk (lst)
    "Knuth's g function.
     The input lst is intended to be a nonempty
     list of integers."
    (declare (xargs :guard (and (integer-listp lst)
                                (consp lst))))
    (cond ((consp (nthcdr 2 lst))    ;; (len lst) > 2
           (cond ((equal (first lst)
                         (+ (second lst) 1))
                  (Gk (rest lst)))
                 ((equal (second lst)
                         (+ (third lst) 1))
                  (max (third lst)
                       (last-el lst)))
                 (t (last-el lst))))
          (t (last-el lst))))        ;; (len lst) <= 2
```

The following theorem is proved by considering all the cases.

```
(defthm
    Fk=Fb-0-5
    (implies (and (integer-listp lst)
                  (not (consp (nthcdr 5 lst))))
                                            ;; (len lst) <= 5
             (equal (Fk lst)(Fb lst)))
    ...)
```

Direct computation verifies the example showing that `Fk` and `Fb` can return different results for $m = 6$.

```
(defthm
    Fk<>Fb-6-example
    (let ((lst '(8 6 4 3 1 2)))
      (and (equal (Fk lst) 2)
           (equal (Fb lst) 3)
           (not (equal (Fk lst)(Fb lst))))))
```

The *restricted* tarai recursion halts

The measure mentioned earlier is used to demonstrate that the recursion terminates. The recursive calls in the definition of the restricted tarai function all occur within this call to `rTarai`

```
(rTarai (rTarai-lst (lst-rotates-with-minus-1
                         (- (DEC-FRONT-LEN lst) 1)
                         lst))).
```

In addition to the explicit call to `rTarai` shown above, several calls to `rTarai` are required to compute

```
(rTarai-lst (lst-rotates-with-minus-1
                 (- (DEC-FRONT-LEN lst) 1)
                 lst)).
```

The calls to `rTarai` required to compute the call to `rTarai-lst` all have input lists with smaller measure than the original input list, at least for lists of lengths 2–7.

```
(defthm
    e0-ord-<-measure-rotates
    (implies
     (and (integer-listp lst)
          (consp (nthcdr 1 lst))       ;; (len lst) > 1
          (not (consp (nthcdr 7 lst))) ;; (len lst) <= 7
          (> (first lst)(second lst))
          (member-equal rlst (lst-rotates-with-minus-1
                              (- (DEC-FRONT-LEN lst) 1)
                              lst)))
     (e0-ord-< (measure rlst)
               (measure lst)))
     ...)
```

The input list to the explicit call to `rTarai` shown above also has smaller measure than the original input list.

```
(defthm
    e0-ord-<-measure-rTarai-lst
    (implies
       (and (integer-listp lst)
            (consp (nthcdr 1 lst))       ;; (len lst) > 1
            (not (consp (nthcdr 7 lst))) ;; (len lst) <= 7
            (> (first lst)(second lst)))
       (e0-ord-< (measure (rTarai-lst
                            (lst-rotates-with-minus-1
                             (- (DEC-FRONT-LEN lst) 1)
                             lst)))
                 (measure lst)))
    ...)
```

4 Current and Future Work

Use ACL2 to prove, in an elegant way, all items 1, 3–6 in section 3, for all integers $m \geq 2$.

A formal proof, using ACL2, of **Conjecture 1**, is currently under construction, but not yet complete. An informal proof, checked by hand, required consideration of many cases. To ensure that all possible input lists of size two or more had been considered, regular expressions from formal language theory were employed in the following way.

The proof depends not on the values of the individual components of the input $x_1, \ldots, x_m$, but on the differences between adjacent pairs of input values. With each input list $x_1, \ldots, x_m$ associate another list $s_1, \ldots, s_{m-1}$, where $s_i = x_{i+1} - x_i$. The s_i are called *steps*. For example, associated with the input list $4, 2, 3, 2, 2, 8$ is the step list $-2, 1, -1, 0, 6$.

Step lists are encoded as strings of the following symbols with the indicated meaning.

symbol	meaning	
R	$s_i \geq 0$	a *R*ising step
U	$s_i = -1$	a *U*nit step down
D	$s_i = -2$	a *D*ouble step down
T	$s_i \leq -3$	a *T*remendous step down
$B(= D + T)$	$s_i \leq -2$	a *B*ig step down
$A(= U + B)$	$s_i \leq -1$	*A*ny step down
$C(= R + A)$		the *C*omplete set of all possible steps

So the step list $-2, 1, -1, 0, 6$ can be encoded by several strings, two of which are $DRURR$ and $BRARR$.

Recall that a regular expression is a string of symbols representing a set of strings. The operations union, concatenation, and Kleene closure, on sets of strings, are represented, in expressions, respectively, by $+$, juxtaposition, and the superscript $*$.

With these definitions, the set of all possible input lists is represented by the regular expression CC^*. Each case is a set of input lists represented by a regular expression such as $U^*DUU^*BB^*RC^*$. This case includes all input lists of length four or more having zero or more *U*nit steps, followed by a *D*ouble step, followed by one or more *U*nit steps, followed by one or more *B*ig steps, followed by a *R*ising step, followed by zero or more additional steps.

Our informal proof considers sixteen such cases. Using the algebra of regular expressions [1], it is possible to show that the sum of the sixteen regular expressions representing our cases equals CC^*. This process revealed several flaws in earlier proofs, including three new cases that had been overlooked. The cases we considered are represented by

$$\begin{aligned} CC^* &= AA^* + A^*RC^* \\ &= UU^* + \\ &\quad UU^*BA^* + \\ &\quad B + \\ &\quad BBA^* + \\ &\quad BUU^* + \\ &\quad BUU^*BA^* + \\ & \\ &\quad RC^* + \end{aligned}$$

$$
\begin{aligned}
&UU^*RC^* + \\
&U^*DRC^* + \\
&U^*DUU^*RC^* + \\
&U^*DUU^*BB^*RC^* + \\
&U^*DUU^*BB^*UA^*RC^* + \\
&U^*DBB^*RC^* + \\
&U^*DBB^*UA^*RC^* + \\
&U^*TB^*RC^* + \\
&U^*TB^*UA^*RC^*
\end{aligned}
$$

5 Conclusion

Our attempt to use ACL2 to meet Knuth's machine verification challenge about his generalization of Takeuchi's tarai function led to a correction of the original formulation of his theorem. ACL2 has verified the corrected version of Knuth's theorem for input lists of length m, when $2 \leq m \leq 7$. ACL2 is currently being used to formally check the proof of the theorem for all $m \geq 2$.

References

[1] J. Hein. *Discrete Structures, Logic, and Computability.* Jones and Bartlett, 1995, page 581.

[2] D.E. Knuth. Textbook Examples of Recursion. In V. Lifschitz, Editor, *Artificial Intelligence and Mathematical Theory of Computation: Papers in Honor of John McCarthy*, pages 207–230. Academic Press, 1991.

[3] D.E. Knuth. *Selected Papers on the Analysis of Algorithms.* CSLI Publications, Distributed by Cambridge University Press, 2000. Chapter 22 is an update of [2].

[4] D.E. Knuth. Personal communication. 16 February 00.

[5] J S. Moore. "A Mechanical Proof of the Termination of Takeuchi's Function". *Information Precessing Letters 9*, 4 (1979), 176–181.