Now at *U. of Miami*:
Yoram Yakimovsky (from JPL and Stanford U.), medical student.

Now at Dept. of Computer Science, *U. of Utah* (Salt Lake City, UT 84112):
Gary Lindstrom (from U. of Pittsburgh).

At Dept. of Computer Science, *CMU* (Pittsburgh, PA 15213):
Donald Michie (from U. of Edinbourgh) Visiting Professor, Feb. 27 -- Apr. 16, 1977.

Now at *Xerox PARC* (Palo Alto, CA 94304):
Richard Fikes (from SRI) to work with Warren Teitelman and Danny Bobrow's *Understander* group.

Now at *UC Irvine* (Dept. of Inf. and Computer Sci.):
James Meehan (from Yale).

AI FORUM

The Samefringe Problem

Bruce Anderson
Computing Centre
University of Essex Essex, England

Unfortunately (well, fortunately for Prenner, Hewitt . . .) Patrick Greussay (1976) has a correct solution but to the wrong problem! The original puzzle, as told to me by Carl Hewitt at IJCAI2, is to test the equality of the fringes WITHOUT CONSING. The motivation is that after all you can walk the tree this way, for example to print it or find if it contains FOO.

Here's why you can't do it. The obvious solution is to treat each tree as a stream and then compare them incrementally, which is entirely analogous to putting the trees leaf by leaf into a pair of one-leaf buffers, comparing in between puts. But consider what the streams do. Each has the usual depth-first tree traversal algorithm in it, and each time it is asked for "next leaf" it must continue the algorithm from where it last output a leaf; i.e., it must preserve its own CONTROL state between calls, which is a thing LISP can't do and coroutines and the continuing debate/research on AI languages is mostly about.

This is in a strong sense the only way to do the problem, and indeed Greussay's FRINGE constructs stream-like objects with CAR the next leaf and CDR a coded form of what to do next, which SAMEFRINGE interprets. (Very interesting <Weizenbaum 1976>)

Oddly enough Steve Hardy of Sussex sent essentially the same program to me in response to my own incomplete statement of the SAMEFRINGE problem in a draft of a paper I wrote <Anderson 1976> which touches on some of these issues.

Obvious remarks:

1. I suppose we should say "in a fixed amount of storage" rather than "without CONSing", and indeed we have to include the "data" qualification since the coroutine hack uses control storage. Actually it isn't a hack since the allocation and deallocation is handled perfectly and without garbage-collection.
2. Yes you can do it in a Lisp with spaghetti-stacks, since then you can implement the streams, though not very cleanly in Interlisp, for example.
3. Yes (Roger and all you other hackers!) in the case of no sharing you could trade space for time and keep the stream as a pointer to the list cell containing the last leaf compared, next time running right through the tree until you got there, etc. But with sharing you can't because that pointer might not be unique and you could need O(tree storage) storage to keep counts. And so on.

References

Anderson, D. B. "A brief critique of Lisp", *Proc. AISB 1976 Summer Conference*, Edinburgh (1976).

Greussay, P. "An iterative Lisp solution to the Samefringe problem", *SIGART Newsletter*, 59, p. 14 (1976).

Weizenbaum, J. "Response to Colby", *SIGART Newsletter* 59, p. 10-11 (1976).

Different Fringe for Different Folk

Tim Finin and Paul Rutter
Coordinated Science Lab
University of Ill. Urbana, IL 61801

As Greussay has pointed out,[1] the SAMEFRINGE problem has been notoriously overused as a justification for coroutines. However, the "ordinary LISP" solution he offers is unnecessarily inefficient in both time and space. Briefly, the SAMEFRINGE problem is to write a function that will return true if the atoms in two LISP trees are the same and appear in the same order when the trees are traversed in left-to-right preorder (i.e., if you PRINT them and delete all parentheses and spaces, the two resulting character strings would be identical). The problem is in finding an efficient solution that will not do any unnecessary work when the two trees differ in their first positions. The obvious hack of flattening the two trees and then using EQUAL is not a good idea for this reason.

We note first of all that Greussay's algorithm does not treat NIL as an atom but as a "leafless" tree. That is, the two trees (A NIL B) and (A B) are considered to have the same fringe. We consider this to be a bug, although a minor one which could be fixed.

A more serious criticism is that his algorithm is grossly inefficient in both execution time and memory requirements. Both time and memory are proportional to the square of the number of atoms in the trees to be compared, regardless of their structures. Given two trees, T1 and T2, such that each contains N atoms and the two have the same fringe, his algorithm requires:

$$3N^2 - 5N + 3(\text{pathlength}(T1) + \text{pathlength}(T2))$$

CONS cells to determine that the trees have the same fringe. Although some of these cells can be "garbage collected" along the way, the number which cannot is still proportional to N^2. For example,

(SAMEFRINGE '(1 2 ... 100) '(1 2 ... 100))

would explicitly chew up 30,100 cells, about a third of which could not be garbage collected until the end.

The first SAMEFRINGE algorithm we present requires memory proportional to the number of atoms in the trees to be compared. The total number of CONS cells used is less than the sum of the number of atoms in the two trees. This algorithm has the nice property that the number of CONS cells needed is inversely related to the similarity between the two trees. For example, SAMEFRINGE of EQUAL trees requires no additional CONS cells. In fact, we have found it difficult to devise examples which require more than a few extra CONS cells.

The execution time is apparently linear with respect to the number of atoms in the trees. We hedge because there is some chance that N^2 computations may be hiding in the calls to NCONC. This algorithm avoids order N^2 memory requirements by judiciously "flattening" parts of the trees as it goes along. Don't let the NCONC scare you, the algorithm is not destructive.

1. *SIGART Newsletter* 59, August 1976.

```
(DE SAMEFRINGE (t1 t2)
   ((LAMBDA (from? depth) (EQ (SF t1 t2 0.) T)) NIL 0.))
(DE SF (t1 t2 depth)
 (COND ((NULL t1)
        (COND ((NULL t2) (SETQ from? 'BOTH) T)
              ((SETQ from? 't2) (NCONS t2))))
       ((NULL t2) (SETQ from? 't1) (NCONS t1))
       ((ATOM (CAR t1))
        (COND ((ATOM (CAR t2))
               (AND (EQUAL (CAR t1) (CAR t2))
                    (SF (CDR t1) (CDR t2) depth)))
              ((JOIN (SF t1 (CAR t2)(1+ depth)) NIL (CDR t2)))))
       ((ATOM (CAR t2))
        (JOIN (SF (CAR t1) t2 (1+ depth)) (CDR t1) NIL))
       (T (JOIN (SF (CAR t1)(CAR t2)(1+ depth))
                (CDR t1)
                (CDR t2)))))
(DE JOIN (excess rest1 rest2)
 (COND ((NULL excess) NIL)
       ((NOR rest1 rest2) excess)
       ((EQ from? 'BOTH) (SF rest1 rest2 (1+ depth)))
       ((EQ from? 't1)
        (COND (rest2 (SF (CONSER excess rest1) rest2
                         (1+ depth)))
              (T (CONSER excess rest1))))
       ((EQ from? 't2)
        (COND (rest1 (SF rest1 (CONSER excess rest2)
                         (1+ depth)))
              (T (CONSER excess rest2))))))
(DE CONSER (a b)
   (OR (AND a b (NCONC a (COND ((ZEROP depth) b)
                               (T (NCONS b)))))
       a
       b))
```

Finally, one must wonder why the obvious two-stack non-recursive solution is unacceptable to so many people. It runs in linear time, is easy to understand, and works for atomic arguments and trees which contain "dotted-pairs" (CONS cells whose CDR is atomic but not NIL) -- things which both the above solution and Greussay's blow up on.

```
(DE SAMEFRINGE (t1 t2)
  (PROG (stack1 stack2)
    (WHILE (AND (EQUAL (PROGN (UNTIL (ATOM t1)
                                 (IF (CDR t1)
                                    (PUSH stack1 (CDR t1)))
                                 (SETQ t1 (CAR t1)))
                               t1)
                       (PROGN (UNTIL (ATOM t2)
                                 (IF (CDR t2)
                                    (PUSH stack2 (CDR t2)))
                                 (SETQ t2 (CAR t2)))
                               t2))
                stack1
                stack2)
           (POP stack1 t1)
           (POP stack2 t2))
    (RETURN (AND (EQUAL t1 t2) (NULL stack1) (NULL stack2)))))
```

In the above, the two stacks are implemented as lists (PUSH and POP are macros in our system), so there is some consing (always less than the total size of the two trees). However, the stacks can be implemented as arrays (by appropriately changing PUSH and POP, and the tests for stack-empty) and then there is no consing at all.

Grosch on McCarthy on Weizenbaum

Herb Grosch
Sunnyvale, CA 94087

I read and re-read the McCarthy-Weizenbaum exchange in the June *SIGART Newsletter* with keen interest. My own sympathies and antipathies are well known, and I don't need to use up your valuable space to express them again. Can't resist the temptation to make two brief remarks, however. First, I call the attention of other readers to the remarkable wording with which McCarthy begins his review: "This moralistic and incoherent book . . ."; that is, he regards the adjective "moralistic" as pejorative!

More important, I want to most strongly second Weizenbaum's position, that he finds ". . . the idea of disembodying the brain and eyes of a cat . . . and using them as the receptors of a computing system obscene". It isn't something one can argue about; it goes down into the deepest wellsprings of the heart. I fervently hope most members of our curious fraternity share Joe's horror.

No living creature should be sacrificed on such a tawdry altar, not even a housefly. Biological contributions should be limited to volunteers, and I would encourage appropriate individuals to step forward!

LOGO and Faith

Fr. Aloysius Hacker, Interdisciplinary Chaplain
Cognitive Divinity Program Institute of Applied Epistemology[1]
- LOGO Keeps the Faith

Dear Sir,

In these fast-changing times, with the lowering of moral standards and the increase in new programming languages, I often find troubled young men asking me the question, "Father, how can I be sure that LOGO is the one true programming language?"

For such confused souls, an examination of their childhood LOGO catechism often proves of comfort. Remember how it begins . . . .

1. What is LOGO? LOGO is a programming language.
2. Who made LOGO? Papert made LOGO.
3. Why did he make LOGO? He made LOGO to help children and social science students know it, love it, and through it learn to juggle and ride a unicycle.
4. Is there only one LOGO? No, there are many LOGO's but they are all reflections of the one true LOGO.
5. Is LOGO just a programming language? No, LOGO is more than a programming language; it is also a theory of learning and the path to deep psychological insight.
6. Will LOGO help me? Yes, LOGO will help you by making your problem solving process explicit and transparent unto you.
7. What are the four last things? Procedures, debugging, problem decomposition, and recursion.
8. Will LOGO stop people from kicking sand in my face? No, but it should keep you off the beach.[2]

1. Reprinted from *AISB European Newsletter*, 23, July, 1976.
2. For further details apply to Computer Truth Society.