STANFORD ARTIFICIAL INTELLIGENCE PROJECT
MEMO 159.

July [illegible]

"THE MIKADO" AS AN ADVICE TAKER PROBLEM

by Fred Safier

Abstract: The situation of the Second Act of "The Mikado" is analyzed from the point of view of Advice Taker formalism. This indicates defects still present in the language

The research reported here was supported in part by the Advanced Research Projects Agency of the Office of the Secretary of Defense (SD-183).

At the LISP 2 meeting in July, considerable discussion was devoted to the problem of allowing a wide variety of new types of entities, but no definite conclusions were reached. In the last few weeks a new approach to types has been developed that simplifies the problem greatly. The simplicity comes from separating the problem of computation types (e.g. integer vs real) from the problem of storage types and solving the storage type problem by itself.

Here is the solution. The basic entity in the system is list structure as in LISP 1.5. However, a word with negative sign is a storage layout word and can say any of the following things:-

1. Following me in memory is a block of n words none of which have relocatable addresses or decrements.

2. Following me in memory is a block of n words each of which has relocatable address and decrement.

3. Following me is a block of n words whose relocatability is determined by bits in as many of the words that immediately follow me as is necessary (Both address and decrement relocate if either does).

4. Following me is a block of n words of program. The bits that immediately follow say which words have relocatable address parts.

In each case the address part of the layout word may contain a type name (e.g. real or integer) usable for determining computation type and serving to distinguish the type in sets defined by the direct union operation of "A Basis".

The first advantage of this scheme is that the marker of the packing garbage collector can know how to trace the storage from the layout words alone; it does not have to refer to separate type definition statements.

The second advantage of this scheme is that the public pushdown list can be made to include layout words just like free storage itself. This simplifies the marker which can now simply start marking at the head of the pushdown list.

We would like to include the LISP 2 program itself and its subroutines in the free storage area and have it referred to from the top of the pushdown list. This would mean that parts of the system could be excised, and the garbage collector would automatically pack everything else.

We shall now review the garbage collector for LISP 2 which is substantially as discussed at the July conference. It operates in four phases.

1. Marking. In a separate table two bits are reserved for each word of free storage and this table is initially set to all zeroes. The marker starts at the head of the pushdown list and traces the memory structure determined by list structure and the layout words. Each word that it finds has one of three entries made in the mark table according to whether it doesn't relocate (numbers), has address relocation only (certain program words), or has address and decrement relocatable (list structure). Layout words are of the third type.

Besides these functions which are analogous to LISP 1.5 we also want to be able to reserve an array in storage for later assignment statements. This is accomplished by the functions:-

declare 1 [n; type]

declare 2 [n; type]

and declare 3 [n; type; r_1; ... r_i]

which have as value the location of the layout word of a new block of a given type but do not actually put anything into the block (It is set to all zeroes).

The storage scheme described above provides the ability to use most of the kinds of entity proposed at the conference without making any commitments to specific computation types.

In their present form the functions are unsafe. For example if x is the location of a layout word specifying a block of three unrelocatable words, then cwr(x +4) does not depend on the quantity represented by x but belongs in another block, and even x + 1 will not be a legitimate LISP 2 entity since it is not the location of either list structure or specification words. This suggests that these functions not be used directly, but that the compiler generate the appropriate functions when reading type definition statements. Thus the compiler might generate $\lambda[[x]; cwr[x + 1]]$ and $\lambda[[x]; cwr[x + 2]]$ to pick up the real and imaginary parts of a complex number. For the time being, however, a very powerful and easy to produce system can be made from the basic functions. Misuse of the basic functions will lead to obscure bugs because the marker will become confused.

JMcC/afg.

The situations occurring in an operetta afford good opportunity for analysis as potential Advice Taker problems since they are approximations to real life problems, yet are simple enough to be reduced to formalizations. With the end in view of sharpening the proposed Advice Taker language and pointing up its deficiencies, we can study the situation involved in the second act of Gilbert and Sullivan's "The Mikado". I propose to show how the knot of the situation can be untied in the present formulation of the Advice Taker language.

Here is an exposition of the problem, shorn of irrelevancies:-

Katisha, an elderly lady of influence, has interpreted Nanki-Poo's natural amiability as flirting, the only crime punishable by execution in Japan. On her application to the Mikado, Nanki-Poo was ordered to marry her within a week or die. He promptly disappeared and has turned up in Titipu, disguised as a second Trombone. He is in love with Yum-Yum whom he cannot marry since she is engaged to Ko-Ko, the Lord High Executioner. Ko-Ko's difficulty is that if he does not execute somebody soon, he will lose his job. Since Nanki-Poo is about to kill himself in despair, Ko-Ko suggests that he wait, become Ko-Ko's victim, and die thus. Nanki-Poo agrees on condition that he can marry Yum-Yum for the short time he is yet alive. Ko-Ko consents, and the two are married. When the Mikado appears, Ko-Ko describes to him with a wealth of corroborative detail the execution of the second trombone (the actual execution was impossible since by Japanese law Yum-Yum would die upon her husband's execution). It turns out that the Mikado was not very interested in the execution, but rather has arrived in Titipu because he heard that his son was there. The Mikado and Katisha discover that Ko-Ko has apparently executed Nanki-Poo, not knowing that the latter was the heir-apparent. The penalty for compassing the death of the Heir Apparent is death by boiling oil. The only way Ko-Ko can save himself is by producing Nanki-Poo and explaining away the execution. However, should Nanki-Poo appear, the latter will die because of Katisha's accusation, and since he is now happily married to Yum-Yum he prefers to be, to Katisha's knowledge, a disembodied spirit. If Katisha gets married however, she will have no further claim on Nanki-Poo and Ko-Ko's problem will be solved.

In the following analysis, we use these definitions and axioms from "Situations, Actions and Causal Laws", by John McCarthy:

1. cause $(\pi)(s)$ means that the situation s will lead in the future to a situation that satisfies the fluent π.

2. can$(p,\pi)(s)$ means that the person p can make the situation s satisfy π.

(From now on we use the convention of suppressing explicit mention of the situation s)

C1. $\forall.\text{cause}(\pi) \wedge (\forall.\pi \Rightarrow \rho) \Rightarrow \text{cause}(\rho)$

C2. $\forall.\ \text{cause}(\text{cause}(\pi)) \Rightarrow \text{cause}(\pi)$

K1. $\forall\pi\forall\sigma\forall p\ (can(p,\pi)\wedge(\pi==>\rho)\ ==>can(p,\rho))$

KC1. $\forall p\forall\pi\ .\ .\ .\ \forall\ can(p,cause(canult(p,\pi)))\ ==>canult(p,\pi)$, which, as McCarthy remarks, is partially equivalent to the LISP recursive definition:

$$canult(p,\pi)=\ .\ .\ \forall\ can(p,cause(canult(p,\pi)))$$

Theorem 1. $can(p,cause(\pi))\wedge\forall.\ (\pi == can(p,cause(\rho)));=>can(p\ cause(can(p,cause(\rho))))$

We are now ready to put down the formalization, first of the problem and then of the solution. We have nine observations:

H1. $\forall x\forall y(\neg married(x)\wedge\neg married(y)\wedge male(x)\wedge female(y))==>can(x,can(y,marry(x,y)))$

H2. $\forall x\forall y marry(x,y)==>cause(married(x)\wedge married(y))$

H3. $\neg married(Katisha)\wedge\neg married(Ko\text{-}Ko)\wedge male(Ko\text{-}Ko)\wedge female(Katisha)$

H4. $\forall(married(Katisha)<==>$ -Katisha has claim on Nanki-Poo)

H5. $\forall$(-Katisha has claim on Nanki-Poo ==>cause(-Katisha accuses Nanki-Poo))

H6. $\forall$(-Katisha accuses Nanki-Poo ==>can(Nanki-Poo, appear$\wedge$continue living))

H7. $\forall$(can(Nanki-Poo, appear$\wedge$continue living) ==>can(Ko-Ko,produce(Ko-Ko,Nanki-Poo)))

H8. $\forall p.$ (produce(p,Nanki-Poo)==>cause(-Mikado thinks Nanki-Poo dead))

H9. $\forall$(-Mikado thinks Nanki-Poo dead ==>cause(Ko-Ko continues living))

The Problem is to prove that canult(Ko-Ko, Ko-Ko continues living).

Proof:

From H2 we deduce

1. marry(Katisha, Ko-Ko)==>cause(married(Katisha))

From 1 and H4, by use of axiom C1, we deduce

2. marry(Katisha,Ko-Ko)==>cause(-Katisha has claim on Nanki-Poo)

From 2 and H5, we again find from axiom C1

3. marry(Katisha,Ko-Ko)==>cause(cause(-Katisha accuses Nanki-Poo))

From 3, by application of axiom C2, we have

4. marry(Katisha,Ko-Ko)==>cause(-Katisha accuses Nanki-Poo)

From H1 and H3, we deduce

5. can(Ko-Ko, marry(Ko-Ko Katisha))

4 and 5 give, on application of axiom K1,

6. can(Ko-Ko, cause(-Katisha accuses Nanki-Poo))

From H7 and H6, we deduce

7. -Katisha accuses Nanki-Poo ⇒ can(Ko-Ko, produce(Ko-Ko,Nanki-Poo))

From H8 and H9, we deduce by use of axiom C1

8. ∀p(produce(p,Nanki-Poo)⇒cause(cause(Ko-Ko continues living)))

From 8, by application of axiom C2, we have

9. ∀p.(produce(p,Nanki-Poo)⇒cause(Ko-Ko continues living))

From 7 and 9 we get by axiom K1

10. -Katisha accuses Nanki-Poo⇒can(Ko-Ko,cause(Ko-Ko continues living))

The crucial deduction is from 6 and 10, by Theorem 1

11. can(Ko-Ko,cause(can(Ko-Ko,cause(Ko-Ko continues living))))

Then we have the desired result from 11 by axiom KC1

12. canult(Ko-Ko, Ko-Ko continues living).

This completes the deduction.

The principal moral that I draw from this analysis is that the Advice-Taker language is not yet properly tuned to actions by more than one person. For example, the deduction step 5, which says that Ko-Ko can marry Katisha by himself, does not follow from the permissible statement:

can(Katisha,can(Ko-Ko, marry(Ko-Ko,Katisha))).

Further, a deduction using canult is not possible involving actions by two persons. More analysis of this problem is likely to indicate a solution to these difficulties.