Systems that have ideas instead of rule based knowledge

Rene Heuer
(040) 250 44 38
Salingtwiete 4 g
D-2000 Hamburg 26 W-Germany

Definition of an idea

If you have a screwdriver, you have a lot of ideas what could be done with this tool. These ideas are coming up because you know that the screwdriver is a tool and there could be a situation where you have - for example - an unopened bottle and a screwdriver. A screwdriver is one of a lot tools that opens bottles.

An idea is *knowing* about alternatives which are found by assoziations. My *human* idea was to organize possible assoziations in computer memorys. After a short time of hard work I stopped because I found that it is absolutely impossible to organize assoziations. Assoziations *must* be found and organized by the system itself. What I had to do was to find out something like *basic motivation* to search assoziations. This *basic motivation* must be programmable.

Basic motivation and datastructure

There must be some reasons for the system to manipulate his own data structures. These reasons also must have the same currency in every situation that could be. All possible reasons can be represented only by one, called *basic motivation*. I formulate it to:

If there is anything bad, try to eliminate your internal assumptions of this negative situation.

Because the manipulation must be possible on each level of datarepresentation I have saerched for a datastructure that must be easy to handle with time efficient CPU instructions. *FORTH showed me the right way*. In FORTH it is very easy to define structures that have pointers to memory fields with structures that are redefinable under process. But the restrictions of 16 bit organisation in normal FORTH systems are keeping the system slow. I need 32, better 64 bits for addressing the assoziations in a very short time - because we are talking about real time artifical intelligence. Today I write a subset of FORTH that is very specified to these problems.

Every date in the system is represented by characters and pointers. The characters are oniy characters, a pointer is the difference to the next character or to an array of pointers.

Structure of a normal character:

```
+------+----------------------+
! byte !     32 bit address   !
+------+----------------------+

  byte           = single character of a sequence
  32 bit address = difference to next character or array
```

Structure of an pointer array:

```
                  +-----------------------+
                  !      32 bit addresss   !
                  +-----------------------+
                  !      32 bit addresss   !
                  +-----------------------+
                  !      32 bit addresss   !
                  +-----------------------+
                              .
                              :
                  +-----------------------+
                  !      32 bit addresss   !
                  +-----------------------+

              32 bit address = difference to next character or array
```

Assoziations on character level:

```
                                r->-+
                                    !
                            +---o-<-+
                            !
                            +->-s->-+->-+->-+ pointer to 3 pointer array
                                    !   !   !
 pointer to assoziation ...... e-<-+   !   ! rose
                                        !   !
                            +-<-m---<---+   !
                            !               !
                            +->-a-----<-----+
                                !
                                r
                                !
 pointer to assoziations ..... y ... rosary, rosemary
```

There are two reasons for this structure. First - *every often used character sequence can be represented only by one pointer.* Second - *all words of a natural language can be reached by a pointer.* At the end of every word or sequence of characters, there is an pointer too that addresses an array of pointers in every case. This is because the word could have an assoziation to an other word (for example: tool). Possible assoziations, are now represented only by one pointer. Because ther is to do a lot of garbage collection when a new character is integrated to the system I found that this *system needs sleep.* This seems to be stupid but I found no other synonym. Every new characters, pointers to existent character sequences or assoziations must be available immediately. So the manifestation of knowledge is done when the system is not in use. At the other time the pointer arrays are positioned at the end of datamemory and differences are integrated while the system works for the user. But now I want to go through an example of assoziations in the next higher level and coming back to lowest assoziation after I told you, why my system needs *sleep.*

If the system finds the word screwdriver it has an assoziation to tool and now, via toolś pointer array, to many other tools. In this example tool is one level higher than screwdriver. The datasructure of knowledge in my system is stricly organized in notion levels. To work through this knowlwdge with the *lambda calculus* it is very very easy to do, because you need no parathensis or other synomymous of things what FORTH programmers make to hate this structure. I went an other way to work through heuristic organized knowledge, but the realisation in FORTH takes time. I hope the end of this year will bring me *the truth in FORTH*. Whatever, knowledge in this systems is not absolutely true. This is because assoziations in this representation are very dynamically. When the system is in use, it *learns* new facts and gets new assoziations in his knowledge base. The manifestation of new pointers or pointer arrays is done by the *basic motivation*. New assoziations are coming up and new *IDEAS* are in system's *mind*. Because this takes time and memory space the system needs *sleep* to do not forget. After more and more knowledge is manifested the sleeping time is going down more and more too becuse the user cant tell the system news.

To build and test assoziations, my system needs *basic motivation*. There must be some definitions to sense positive or negative situations. My definitions are extremely simple, but very efficient.

-	... Info not found, info not received, external reaction negative.
0	... Info still available, no external reaction.
+	... Info found, Info received, external reaction positive.

Learning of new facts and manifestation

If anyone tells the sytem that *the rose is a plant* after a lot of assoziations are manifested, something like the following is happend:

1. the	=	next word could be plural or singular
2. rose	=	object(s) that gets an assotioazion
3. is	=	plural can't be
4. a	=	assoziation is singular also
5. plant	=	assoziation of rose

One of the best things in FORTH are the *primitives* and so my sytems has something like this also. I call them *simples* to have a differntiation to FORTH. Till now there are the following *simples* :

-> = word -> assoziation,	the character sequence gets a new pointer in its assotiation array.
!! = word !!,	there can't be found an assoziation.
<- = <- word,	word is an assoziation itself.
?! = word -> assoziation ?!,	found an assoziation that is contraher to one or more other.
?> = word -> assoziation ?>,	question if there are assoziations in a higher level.

Using the sentence "*the rose is a plant*" as an example for this *simples* it is possible to show the internal building of new assoziations. Or let me reformulate the title of this paper: *The system has an new idea*.

Here is the detailed showing what is going on in the system:

```
 "the"   -> singular
 "rose"  !!
 "is"    -> "build assoziation"
 "a"     -> singular
 "plant" <- "plant"

Scheme : "rose" -> "plant"    done in upper example
         "rose" <- "plant"    one more pointer to the lower level
         "rose" ?>            more assoziations?
         <- "plant"           "plant" has more assoziations
         "flower" <- "plant"  assoziation to "flower"

NOW BUILD

         "rose" -> "flower"
         "rose" <- "flower"   put "flower" one level up. Now "flower" is
                              an assoziation too and  on the  same level
                              as "plant".
```

This is the actual situation with my system and it is a good situation to get results like this without heuristics. But there is a lot of work to do, to tell the system that *not all plants are flowers*. I think the best way to put flowers one level higher is to define my *basic motivation* in heuristics also and define *simples* to manipulate the assoziation levels. At this time the complete system is existent on a very small 68008 system with only 512k bytes. To define all basic ideas this is enough, but to get a good working system I think we need one 68020 or NOVIX on the lowest level (character sequences) and eight 68008 on each higher assoziation level. If every level works autarc with his own pointers it needs very short times to find and test new *ideas*. I hope that the definition of the very comlex hardware does not takes too many time and is not too expensive, because my system also must learn to read. This is the simplest way to tell it a lot of facts in a short time. If everything goes all right and the costs of electronics goes down and I find a good join venture I hope that in 2 or 3 years a system like this had read and *understood* the *enceclopaedia britannica*.

I hope that my using force with your language was not too unbearable, nevertheless, thank you for reading this paper with tolerance.