

1

4

```

0 ( EXPERT - Variables and constants                2-26-85) ( EXPERT - EXPERT                2-26-85)
1
2 V= COND-CNT                                     : EXPERT ( #conditions #rules -- , <name> \ compile an expert)
3                                     ( Defined word leaves a truth value to indicate if any )
4 1 C= TRUE                                     ( rules were satisfied when executed.)
5 0 C= FALSE                                     COND-CNT OFF
6 255 C= UNKNOWN                                CREATE HERE >R C, C, 0 C, R> >EXPERT
7                                     RULELIM C@ 2* CONDLIM C@ 3 * + ALLOT
8 0 C= RULELIM ( addr of rule limit byte field) DOES>
9 0 C= CONDLIM ( addr of condition limit byte field) DUP >EXPERT >V
10 0 C= RULECNT ( addr of rule count byte field) RULEARR DUP RULECNT C@ 2* + SWAP
11 0 C= RULEARR ( addr of rule array) DO I @ EXECUTE-RULE
12 0 C= CONDARR ( addr of condition pfa array) IF V> DUP >EXPERT NOT >V LEAVE
13 0 C= CFLGARR ( addr of condition flag array) I RULEARR - 2/ 2/ 2* RULEARR +
14                                     DUP @ I @ SWAP I ! SWAP ! THEN
15                                     2 /LOOP V> NOT ;

```

2

5

```

0 ( EXPERT - XC@ C+! >EXPERT DISTRACT                2-26-85) ( EXPERT - --> ?INDEX {transient words}                2-26-85)
1
2 : XC@ ( addr -- 7/8b ) C@ 127 AND ;                : --> ( <word-name> -- pfa \ search dictionary for pfa of word)
3                                     [COMPILE] ' ;
4 : C+! ( c addr -- \ byte +!) DUP C@ ROT + SWAP C! ;
5                                     : ?INDEX ( cond-pfa -- index or -1 \ search condition array )
6 : >EXPERT ( pfa-expert -- \ set reference constants for expert) ( for index number or flag failure.)
7 >R I ' RULELIM ! I 1+ ' CONDLIM ! I 2+ ' RULECNT ! >R -1 ( null-index-marker -- cond-pfa)
8 R> 3 + ' RULEARR ! RULELIM C@ 2* RULEARR + ' CONDARR ! COND-CNT @ 2* 0
9 CONDLIM C@ 2* CONDARR + ' CFLGARR ! ; DO CONDARR I + @ J = ( search for condition)
10                                     IF DROP I 2/ LEAVE THEN
11 : DISTRACT ( expert-pfa -- \ clear short-term memory of expert) 2 /LOOP R> DROP ;
12 >EXPERT CFLGARR CONDLIM C@ UNKNOWN FILL ;
13
14
15

```

3

6

```

0 ( EXPERT - EXECUTE-RULE                2-26-85) ( EXPERT - CONDITION {transient words}                2-26-85)
1
2 : EXECUTE-RULE ( addr -- f \ attempt to satisfy each predicate) : CONDITION ( <word-name> -- index OR -1 \ find condition)
3 ( in the rule at addr and if satisfied execute the action and) ( index in condition array and possibly install a new )
4 ( report rule satisfaction.) ( OR flag end of cond.)
5 >R TRUE ( t -- \ predicate accum) --> >R ( -- pfa \ get pfa of next word)
6 I 3 + DUP I C@ + SWAP ( for each predicate) I ' --> = NOT
7 DO CFLGARR I XC@ + C@ DUP UNKNOWN = ( t t/f ?unk --) IF CONDLIM C@ COND-CNT @ > NOT ABORT" Condition overflow"
8 IF DROP CONDARR I XC@ 2* + @ EXECUTE NOT NOT I ?INDEX ( cond-index or -1 -- )
9 DUP CFLGARR I XC@ + C! THEN ( t t/f --) DUP 0<
10 I C@ 128 AND NOT NOT = AND ( t --) IF DROP COND-CNT @ I OVER 2* CONDARR + ! ( install cond)
11 DUP 0= IF LEAVE THEN I COND-CNT +! THEN
12 I /LOOP DUP ELSE -1 ( indicate end of conditions)
13 IF >R I' 1+ @ EXECUTE R> THEN THEN
14 R> DROP ;
15

```

7

10

```

0 ( EXPERT - RULE:      (transient words)      2-26-85)
1 : RULE: ( <cond> {T:F}, ..., <cond> {T:F} --> <action> \ )
2 ( compile a rule for the current expert)
3 RULELIM C@ RULECNT C@ > NOT ABORT" Rule overflow"
4 HERE 0 C. ' NOP ,      ( lay down rule head place holders)
5 BEGIN
6   CONDITION DUP 0< NOT
7   IF [COMPILE] ' DUP ' TRUE = OVER ' FALSE = OR NOT
8   ABORT" TRUE or FALSE needed"
9   EXECUTE 128 * + C.    ( lay down predicate)
10  1 OVER C+! 0          ( increment predicate count)
11  THEN
12  UNTIL -->             ( -- rule-pfa action-pfa)
13  OVER 1+ !             ( store action pfa in rule)
14  RULECNT C@ 2* RULEARR + ! ( store rule-pfa in expert head)
15  1 RULECNT C+! ;

```

8

11

```

0 ( ANIMAL EXPERT)
1 exit
2 : ?Y/N ( nfa -- t \ ask a yes/no question)
3   ID. ." ? (Y/N) "
4   KEY DUP EMIT 95 AND ASCII Y = CR ;
5
6 : AT: ( <attribute> -- \ create and attribute predicate)
7   CREATE
8   DOES> NFA ?Y/N ;
9
10
11 : AN: ( <animal> -- \ create an animal predicate)
12   CREATE DOES> ." IS IT A(N) " NFA ?Y/N CR
13   IF ." I knew it!" ELSE ." Your lying!" THEN ;
14
15

```

9

12

```

0 ( ANIMAL EXPERT)
1 exit
2 AT: FEATHERS AT: SCALES AT: NO.LEGS AT: WARM.BLOODED
3 AT: LAYS.EGGS AT: HORNS AT: FUR
4 AT: SHELL AT: TEETH AT: LEAVES AT: FLOWERS
5 AT: FRUIT AT: LIVES.IN.WATER AT: LIVES.ON.LAND
6 AT: JUMPS AT: PURRS AT: HAS.STRIPES AT: EATS.MEAT
7 AT: BARKS AT: OINKS AT: MIGRATORY.EYES
8 AT: TAIL AT: CAMOFLAGE
9 AN: FROG AN: TIGER AN: PLATYPUS
10 AN: TURTLE AN: ZEBRA AN: CHAMELEON
11 AN: CLAM AN: ALLIGATOR
12 AN: CAT AN: SNAKE
13 AN: DOG AN: BIRD
14 AN: PIG AN: STRAWBERRY
15 AN: COW AN: SPEMIN AN: FLOUNDER

```

13

```
0 ( ANIMAL EXPERT)
1 exit
2 RULE: FUR TRUE PURRS TRUE --> CAT
3 RULE: FUR TRUE BARKS TRUE --> DOG
4 RULE: FUR TRUE HORNS TRUE --> COW
5 RULE: FUR TRUE HORNS FALSE EATS.MEAT FALSE HAS.STRIPE TRUE -->
6 ZEBRA
7 RULE: FUR FALSE DINKS TRUE --> PIG
8 RULE: HAS.STRIPE TRUE EATS.MEAT TRUE --> TIGER
9 RULE: LEAVES TRUE FRUIT TRUE --> STRAWBERRY
10 RULE: LAYS.EGGS TRUE FUR TRUE --> PLATYPUS
11 RULE: LIVES.IN.WATER TRUE LIVES.ON.LAND TRUE SHELL FALSE
12 TEETH FALSE JUMPS TRUE --> FROG
13 RULE: LIVES.IN.WATER TRUE LIVES.ON.LAND TRUE SHELL TRUE -->
14 TURTLE
15
```

14

```
0 ( ANIMAL EXPERT)
1 exit
2
3 RULE: LIVES.IN.WATER TRUE LIVES.ON.LAND FALSE SHELL TRUE -->
4 CLAM
5 RULE: LIVES.IN.WATER TRUE LIVES.ON.LAND TRUE SHELL FALSE
6 TEETH TRUE --> ALLIGATOR
7 RULE: LIVES.IN.WATER TRUE LIVES.ON.LAND TRUE SHELL FALSE
8 TEETH FALSE JUMPS FALSE --> SNAKE
9 RULE: FEATHERS TRUE --> BIRD
10 RULE: LAYS.EGGS FALSE MIGRATORY.EYES TRUE --> SPERM
11 RULE: LAYS.EGGS TRUE MIGRATORY.EYES TRUE --> FLOUNDER
12 RULE: CAMOFLAGE TRUE SCALES TRUE --> CHAMELEON
13
14 \ END OF ANIMAL RULES
15
```