# Duce, an oracle based approach to constructive induction*

Stephen Muggleton
Turing Institute,
36 North Hinover Street,
Glasgow,
G12AD,
UNITED KINGDOM.

Abstract.

Duce is a Machine Learning system which suggests high-level domain features to the user (or *oracle)* on the basis of a set of example object descriptions. Six transformation operators are used to successively compress the given examples by generalisation and feature construction. In this paper Duce is illustrated by way of its construction of a simple animal taxonomy and a hierarchical parity checker. However, Duce's main achievement has been the restructuring of a substantial expert system for deciding whether positions within the chess endgame of King-and-Pawn-on-a7 v. King-and-Rook (KPa7KR) are won-for-while or not. The new concepts suggested by Duce for the chess expert system hierarchy were found to be meaningful by the chess expert Ivan Dratko. An existing manually created KPa7KR solution, which was the basis of a recent PhD. thesis, is compared to the structure interactively created by Duce.

## 1. Introduction.

It is well recognised (Feigenbaum 1979) that acquisition of expert knowledge is the major "bottleneck" in expert system development. However, Michalski and Chilausky (1980) and later Quinlan (1982) have shown that this bottleneck can be considerably eased by generalising low-level data to form high-level rules. Shapiro (1983) extended this methodology to deal effectively with extensive bodies of knowledge by employing structured programming techniques. Thus the expert structures the knowledge in a top-down fashion manually, and then provides examples which can be used to inductively generate each module in the hierarchy separately. Using this technique, Shapiro and Kopec created knowledge structures for correctly deciding a forced win for while in any position within the chess endgames of King-and-Pawn v. King (KPK) and King-and-Pawn-on-a7 v. King-and-Rook (KPa7KR). Doth solutions were completely verified by exhaustive computation. However, using an information theoretic approach Shapiro showed that around 80% of the endgame knowledge was provided by the expert in the creation of the knowledge structure. Thus almost all the work was still being done by the expert rather than by the machine. In an attempt to overcome this structuring bottleneck Paterson (1983) tried to use the statistical clustering algorithm CLUSTER (Michalski and Stepp 1982) to automatically restructure the knowledge for the simpler of the two endgames, KPK. Paterson's results were not promising, with the machine's suggested hierarchy not having any significance to domain experts.

In the context of Machine Learning, Michalski (1986) has called the problem of originating terms *constructive induction.* CLUSTER (Michalski and Stepp 1982), perhaps the best known constructive induction algorithm, uses a statistical clustering technique to group objects into conceptual clusters. Each object is initially described in terms of a vector of primitive attribute values. Objects are grouped using a heuristic inter-object distance metric. Rendell (1985), and Fu and Buchanan (1985) describe alternative similarity-based approaches to creating taxonomic hierarchies which work on much the same basis as CLUSTER.

## 2. Transformation-based Induction.

In Duce the approach to constructive induction differs considerably from that of Michalski and Stepp (1982), Rendell (1985), and Fu and Buchanan (1985), and can be more easily compared to the deductive transformational programming techniques of Burstall and Darlington (1977). Durstall and Darlington, and later Dershowitz (1985), suggest that *deductive* program synthesis can be carried out by gradual truth-preserving transformations of a program specification. At first sight, these techniques seem not to be applicable to *inductive* inference, which by definition progresses by performing non-truth-preserving generalisations of the supplied training set. However, if each inductive transformation is tested against an oracle which ensures the validity of any transformation, any such inductive transformation is as legal and safe as its deductive counterparts. This use of an oracle is closely related to Sammut and Banerji's (1986) method of learning concepts by asking questions Indeed, one of the generalisation operators described in the next section is directly due to Sammut and Banerji.

Constructive induction carries out transformations which introduce new terms into the learner's vocabulary. Though such transformations can be truth-preserving, they are not what might be called semantics-preserving. Thus the primary concern in constructive induction should not be "how can new terms be introduced into the vocabulary?' but rather "how can meaningful new terms be introduced?'. Again by using an oracle to either name or reject machine-suggested concepts, the difficult philosophical problems involved in defining the word meaningful can be sidestepped. Given a meaningful and valid training set, every transformation of which is both meaningful and valid (by agreement of the oracle), the resultant rule set will be meaningful and valid.

## 3. Operators.

**Duce takes as input a set of conjunctive productions, or rules, in a form close to that of disjunctive-normal-form propositional calculus. Six operators are employed to progressively transform subsets of the rule base. These operators are described below.**

1) **Inter-construction. This transformation takes a group of rules, such as**

$$X \leftarrow B \wedge C \wedge D \wedge E \qquad (1.1)$$
$$Y \leftarrow A \wedge B \wedge D \wedge F \qquad (1.2)$$

**and replaces them with the rules**

$$X \leftarrow C \wedge E \wedge Z? \qquad (1.1')$$
$$Y \leftarrow A \wedge F \wedge Z? \qquad (1.2')$$
$$Z? \leftarrow B \wedge D \qquad (1.3)$$

**Here the rule for the new concept Z? (1.3) is the most specific generalisation of the rules for X (1.1) and Y (1.2).**

* Thii paper descobes wort which wasunded in part by the British Government's Alvey Logic Database Demonstrator. Research facilities were provided by he Taring Institute, Glasgow. UK. Computational facalties for the prepsrstion of this paper were provided by Interact RAD Corporation, Victoria, BC, Canada

```
(blackbird ← [beak t] ∧ [colour black] ∧ [legs 2] ∧ [tail t] ∧ [wings t])
        eg (blockhead_the_blackbird)
(chimp ← [colour brown] ∧ [hairy t] ∧ [legs 2] ∧ [tail t] ∧ [wings f])
        eg (maggie_the_chimp)
(eagle ← [beak t] ∧ [colour golden] ∧ [legs 2] ∧ [tail t] ∧ [wings t])
        eg (egg_the_eagle)
(elephant ← [colour grey] ∧ [legs 4] ∧ [size big] ∧ [tail t] ∧ [trunk t] ∧
        [wings f]) eg (adult_elephant)
(elephant ← [colour grey] ∧ [legs 4] ∧ [size small] ∧ [tail t] ∧ [trunk t] ∧
        [wings f]) eg (baby_elephant)
(falcon ← [beak t] ∧ [colour brown] ∧ [legs 2] ∧ [size big] ∧ [tail t] ∧
        [wings t]) eg (flap_the_falcon)
(gorilla ← [colour black] ∧ [hairy t] ∧ [legs 2] ∧ [tail f] ∧ [wings f])
        eg (ronnie_the_gorilla)
(lemur ← [colour grey] ∧ [legs 2] ∧ [tail t] ∧ [wings f])
        eg (lemur_alone)
(man ← [colour brown] ∧ [hairy f] ∧ [legs 2] ∧ [size big] ∧ [tail f] ∧
        [wings f]) eg (harry_the_hamite)
(man ← [colour pink] ∧ [hairy f] ∧ [legs 2] ∧ [size small] ∧ [tail f] ∧
        [wings f]) eg (clap_the_caucasian)
(sparrow ← [beak t] ∧ [colour brown] ∧ [legs 2] ∧ [size small] ∧ [tail t] ∧
        [wings t]) eg (sparky_the_sparrow)
```

**Figure 1. Initial set of animal examples.**

```
|- induce
   TRUNCATION -- (-12)
      Is (elephant ← [legs 4]) a valid rule? (y/n/i) n
   TRUNCATION -- (-11)
      Is (elephant ← [legs 4] ∧ [wings f]) a valid rule? (y/n/i) n
   TRUNCATION -- (-11)
      Is (elephant ← [legs 4] ∧ [trunk t]) a valid rule? (y/n/i) y
|- induce
   TRUNCATION -- (-9)
      Is (man ← [hairy f] ∧ [legs 2] ∧ [tail f] ∧ [wings f])
                                          a valid rule? (y/n/i) y
|- induce
   INTER-CONSTRUCTION -- (-1)
    Rule:
      (? ← [legs 2] ∧ [wings f])
    What shall I call <?>? (name/n/i) primate
|- induce
   INTER-CONSTRUCTION -- (-7)
    Rule:
      (? ← [beak t] ∧ [legs 2] ∧ [tail t] ∧ [wings t])
    What shall I call <?>? (name/n/i) bird
|- induce
   No applicable transformation
```

**Figure 2. Animal taxonomy session.**

```
(bird ← [beak t] ∧ [legs 2] ∧ [tail t] ∧ [wings t]) eg (blockhead_the_blackbird
        egg_the_eagle flap_the_falcon sparky_the_sparrow)
(blackbird ← bird ∧ [colour black]) eg (blockhead_the_blackbird)
(chimp ← primate ∧ [colour brown] ∧ [hairy t] ∧ [tail t]) eg (maggie_the_chimp)
(eagle ← bird ∧ [colour golden]) eg (egg_the_eagle)
(elephant ← [legs 4] ∧ [trunk t]) eg (adult_elephant baby_elephant)
(falcon ← bird ∧ [colour brown] ∧ [size big]) eg (flap_the_falcon)
(gorilla ← primate ∧ [colour black] ∧ [hairy t] ∧ [tail f]) eg (ronnie_the_gorilla)
(lemur ← primate ∧ [colour grey] ∧ [tail t]) eg (lemur_alone)
(man ← primate ∧ [hairy f] ∧ [tail f]) eg (clap_the_caucasian harry_the_hamite)
(primate ← [legs 2] ∧ [wings f]) eg (maggie_the_chimp clap_the_caucasian
        ronnie_the_gorilla harry_the_hamite lemur_alone)
(sparrow ← bird ∧ [colour brown] ∧ [size small]) eg (sparky_the_sparrow)
```

**Figure 3. Resultant animal rule base.**

2) Intra •construction. This is simply the distributive law of Boolean equations. Intra-consiruction takes a group of rules all having the same rule head, such as

$$X \leftarrow B \wedge C \wedge D \wedge E \qquad (2.1)$$
$$X \leftarrow A \wedge B \wedge D \wedge F \qquad (2.2)$$

and replaces them with

$$X \leftarrow B \wedge D \wedge Z? \qquad (2.1/2.2)$$
$$Z? \leftarrow C \wedge E \qquad (2.3)$$
$$Z? \leftarrow A \wedge F \qquad (2.4)$$

Note that while operator 1, inter-construction, could legitimately be applied to rules 2.1 and 2.2, the result would be less compact.

3) Absorption. This operator is due to Samtmit and Danorji (1986), who use it to generate recursive Prolog clauses. Even though recursion is not meaningful within propositions! calculus, this operator can be employed profitably in generalising rule sets. Given a set of rules, the body of one of which is completely contained within the bodies of the others, such as

$$X \leftarrow A \wedge B \wedge C \wedge D \wedge E \qquad (3.1)$$
$$Y \leftarrow A \wedge B \wedge C \qquad (3.2)$$

one can hypothesise

$$X \leftarrow Y \wedge D \wedge E \qquad (3.1')$$
$$Y \leftarrow A \wedge B \wedge C \qquad (3.2)$$

Note that the preconditions for applying this operator are stronger than those for applying inter-construction. Also, if rule 3.2 were the only rule with rule head Y, men the new rule is necessarily valid. Otherwise it is a generalisation and must be verified by the oracle. In general, asking the oracle unnecessary questions can be avoided by first attempting to answer the question deductively from the rule base.

4) Identification. The identification operator is again a potential generalisation, whose preconditions are stronger than those of intra-construction. A set of rules which all have the same head, the body of at least one of which contains exactly one symbol not found within the other rules, such as

$$X \leftarrow A \wedge B \wedge C \wedge D \wedge E \qquad (4.1)$$
$$X \leftarrow A \wedge B \wedge Y \qquad (4.2)$$

can be replaced by the rules

$$X \leftarrow A \wedge B \wedge Y \qquad (4.1')$$
$$Y \leftarrow C \wedge D \wedge E \qquad (4.3)$$

5) Dichotomisation. This is operator works on sets of mixed positive and negative examples. Thus a set of rules which contain positive and negative heads, and which all have some common symbols within the bodies, such as

$$X \leftarrow A \wedge B \wedge C \wedge D \qquad (5.1)$$
$$\backslash X \leftarrow A \wedge C \wedge J \wedge K \qquad (5.2)$$
$$\backslash X \leftarrow A \wedge B \wedge C \wedge L \qquad (5.3)$$

are replaced with the rules

$$X \leftarrow A \wedge C \wedge Z? \qquad (5.1')$$
$$\backslash X \leftarrow A \wedge C \wedge \backslash Z? \qquad (5.3')$$
$$Z? \leftarrow B \wedge D \qquad (5.4)$$
$$\backslash Z? \leftarrow J \wedge K \qquad (5.5)$$
$$\backslash Z? \leftarrow B \wedge L \qquad (5.6)$$

where the replacement is dependent on the oracle naming Z7.

Dichotomisation is a generalisation of the the way that 1D3 (Quinlan 1982) creates the internal nodes of decision trees.

```
(even ← [v1 t] ∧ [v2 t] ∧ [v3 t] ∧ [v4 t] ∧ [v5 t] ∧ [v6 t] ∧
        [v7 t] ∧ [v8 t]) eg (tttttttt)
(even ← [v1 t] ∧ [v2 t] ∧ [v3 t] ∧ [v4 t] ∧ [v5 f] ∧ [v6 f] ∧
        [v7 f] ∧ [v8 f]) eg (ttttffff)
(even ← [v1 f] ∧ [v2 f] ∧ [v3 t] ∧ [v4 t] ∧ [v5 f] ∧ [v6 f] ∧
        [v7 f] ∧ [v8 f]) eg (ffttffff)
(even ← [v1 f] ∧ [v2 f] ∧ [v3 f] ∧ [v4 f] ∧ [v5 f] ∧ [v6 f] ∧
        [v7 f] ∧ [v8 f]) eg (ffffffff)
(¬even ← [v1 t] ∧ [v2 t] ∧ [v3 t] ∧ [v4 t] ∧ [v5 t] ∧ [v6 t] ∧
        [v7 t] ∧ [v8 f]) eg (tttttttf)
(¬even ← [v1 t] ∧ [v2 t] ∧ [v3 t] ∧ [v4 t] ∧ [v5 f] ∧ [v6 f] ∧
        [v7 f] ∧ [v8 t]) eg (ttttffft)
(¬even ← [v1 t] ∧ [v2 f] ∧ [v3 t] ∧ [v4 t] ∧ [v5 f] ∧ [v6 f] ∧
        [v7 f] ∧ [v8 f]) eg (tfttffff)
(¬even ← [v1 f] ∧ [v2 t] ∧ [v3 t] ∧ [v4 t] ∧ [v5 t] ∧ [v6 t] ∧
        [v7 t] ∧ [v8 t]) eg (fttttttt)
```

**Figure 4. Even-parity examples.**

```
!- induce
  DICHOTOMISATION -- (-6)
    (even ← [v3 t] ∧ [v4 t] ∧ ?)
    (¬even ← [v3 t] ∧ [v4 t] ∧ ?)
  What shall I call <?>? (name/n/i) n
  DICHOTOMISATION -- (-5)
    (even ← [v1 t] ∧ [v3 t] ∧ [v4 t] ∧ ?)
    (¬even ← [v1 t] ∧ [v3 t] ∧ [v4 t] ∧ ?)
  What shall I call <?>? (name/n/i) n
  DICHOTOMISATION -- (-5)
    (even ← [v2 t] ∧ [v3 t] ∧ [v4 t] ∧ ?)
    (¬even ← [v2 t] ∧ [v3 t] ∧ [v4 t] ∧ ?)
  What shall I call <?>? (name/n/i) n
  DICHOTOMISATION -- (-4)
    (even ← [v1 t] ∧ [v2 t] ∧ [v3 t] ∧ [v4 t] ∧ ?)
    (¬even ← [v1 t] ∧ [v2 t] ∧ [v3 t] ∧ [v4 t] ∧ ?)
  What shall I call <?>? (name/n/i) sev
!- induce
  ABSORPTION -- (-3)
    Is (¬even ← sev ∧ [v1 f] ∧ [v2 t] ∧ [v3 t] ∧ [v4 t])
                                a valid rule? (y/n/i) y
!- induce
  ABSORPTION -- (-9)
    Is (even ← sev ∧ [v1 f] ∧ [v2 f] ∧ [v3 t] ∧ [v4 t])
                                a valid rule? (y/n/i) y
    Is (even ← sev ∧ [v1 f] ∧ [v2 f] ∧ [v3 f] ∧ [v4 f])
                                a valid rule? (y/n/i) y
    Is (¬even ← sev ∧ [v1 t] ∧ [v2 f] ∧ [v3 t] ∧ [v4 t])
                                a valid rule? (y/n/i) y
!- induce
  DICHOTOMISATION -- (-2)
    (even ← sev ∧ [v3 t] ∧ [v4 t] ∧ ?)
    (¬even ← sev ∧ [v3 t] ∧ [v4 t] ∧ ?)
  What shall I call <?>? (name/n/i) ffev
!- induce
  ABSORPTION -- (-1)
    Is (¬even ← ffev ∧ [v3 t] ∧ [v4 t] ∧ v) a valid rule? (y/n/i) y
!- induce
  ABSORPTION -- (-1)
    Is (even ← ffev ∧ sev ∧ [v3 f] ∧ [v4 f]) a valid rule? (y/n/i) y
!- induce
  TRUNCATION -- (-7)
    Is (even ← ffev ∧ sev) a valid rule? (y/n/i) n
No applicable transformation
```

**Figure 5. Parity session.**

6) Truncation, The truncation operator, like dichotomisalion is intended for use with rule sets containing positive and negative examples of the same concept. However, truncation generalises by dropping conditions. A set of rules which all contain the same head, such as

$$X \leftarrow A \wedge B \wedge C \wedge D \qquad (6.1)$$
$$X \leftarrow A \wedge C \wedge J \wedge K \qquad (6.2)$$

is replaced by

$$X \leftarrow A \wedge C \qquad (6.1/6.2)$$

This operator generalises in a similar manner to that of the AQ learning algorithms (Michalski and Chilausky, 1980). Its use is restricted by the precondition that the resultant rule (6.1/6.2) must not clash (i.e. be inconsistent) with any other rule within the rule base. Of all the operators truncation is the only one which reduces the number of rules. All other operators compact the rules by shortening the average rule length.

## 4. The search algorithm.

For any stale of a rule base, there arc many possible operator applications. Any subset of rules within the rule base $R$ is a candidate for the application of one of the 6 operators. Thus the search-space for the "best" operator application is of size $2^{|R|}$, the size of the power set of $R$. What is meant by a "good" operator application? Since each of the operators can reduce the number of symbols in the rule base, Duce searches for the application which produces the largest symbol reduction, i.e. Occam's razor is applied. If each rule is taken as having a symbol size equal to the number of conjunctive terms in the rule body plus one (for the rule head), then for each operator there exists an equation which can be used to predict the exact symbol reduction for any operator. Let R' be a subset of the rule base $R$ and $I'$ be a common subset of all the bodies of rules within $R'$. In the following equations $V_{operator}$ is the symbol reduction produced when the operator is applied to $R'$. The total number of symbols within the rule set $R'$ is written as total $(R')$. The symbol reduction equations arc

$V_{Inter} = (|I'|-1).(|R'|-1)-2$

Note that $V_{op}$ can take a zero or negative value, in which case there is no symbol reduction. Searching for the best operator application is clearly intractable. Moreover, there is no guarantee that such an operator application, once found, would be acceptable to the oracle. In Ducc the next operator application is chosen using a best first search through the power set of the symbols in the rule base $R$. Let a subset of symbols $I'$ be found among the bodies of the rule set $R'$ where $R'$ is the largest subset of $R$ containing $I'$. The operator application *apply(Op, I', R')*, using operator *Op*, is only suggested to the oracle when some $I'$ has been found for which $V_{Op}$ is locally maximal. Any rejection of an operator application by the oracle leads to continued search. Transformations arc carried out itcralivcly until no further operations can reduce the rule base size further. At termination, by nature of the operators, almost all symbols occur within a restricted number of rules. Thus, although the termination condition requires searching the entire remaining space the search space has shrunk to manageable proportions. Since only operations which reduce the number of symbols arc applied termination is guaranteed.

## 5. Animal taxonomy.

This section illustrates the behaviour of Ducc when creating a simple animal taxonomy. Figure 1 shows the set of example animal descriptions given to Ducc. In English the first example says

```
(even ← ffev ∧ sev ∧ [v3 t] ∧ [v4 t]) eg (fftfffff tttffff tttttttt)
(even ← ffev ∧ sev ∧ [v3 f] ∧ [v4 f]) eg (ffffffff)
(ffev ← [v1 t] ∧ [v2 t]) eg (ttttffff tttttttt)
(ffev ← [v1 f] ∧ [v2 f]) eg (fftfffff)
(sev ← [v5 t] ∧ [v6 t] ∧ [v7 t] ∧ [v8 t]) eg (tttttttt)
(sev ← [v5 f] ∧ [v6 f] ∧ [v7 f] ∧ [v8 f]) eg (ttttffff)
(\even ← sev ∧ [v3 t] ∧ [v4 t] ∧ \ffev) eg (fttttttt ttffffff)
(\even ← ffev ∧ [v3 t] ∧ [v4 t] ∧ \sev) eg (ttttffft tttttttf)
(\ffev ← [v1 t] ∧ [v2 f]) eg (tttfffff)
(\ffev ← [v1 f] ∧ [v2 t]) eg (fttttttt)
(\sev ← [v5 t] ∧ [v6 t] ∧ [v7 t] ∧ [v8 f]) eg (tttttttf)
(\sev ← [v5 f] ∧ [v6 f] ∧ [v7 f] ∧ [v8 t]) eg (ttttffft)
```

**Figure 6. Resultant parity rule base.**

```
Pa7
    bxqsq
    rimmx
    stlmt
    Delayed-queening
        hdchk
        Mate-threat
            bkxbq
            bkxwp
            qxmsq
            rxmsq
            r2ar8
        WK-on-a8
            blxwp
            r2ar8
            simpl
            wkna8
        WK-in-check
            bkxcr
            mulch
            rimmx
            rkxwp
            thrsk
            wknck
        Black-attacks-queening-square-soon
                bknwy
                bkona
                bkon8
                skrxp
                wkovl
        Double-attack-threat
            bkblk
            bkxbq
            cntxt
            katri
            wkpos
            Black-advantage-from-potential-skewer
                bkspr
                reskd
                reskr
                r2ar8
                skach
                wkcti
    Delayed-skewer
        bkxbq
        dsopp
        dwipd
        skewr
        spcop
        wtoeg
```

**Figure 7. Human expert's KPa7KR problem decomposition (Shapiro 1983).**

A blackbird has a beak, is black, has two legs, a tail and wings.
"blockhead the blackbird is an instance of the "blackbird" concept

Note the inclusion of the instance set (blockhead the blackbird) within the role. This can be used as a powerful tool for illustrating the meaning of new rules and concepts.

Figure 2 shows user interaction for this example set. User input is shown in bold type. When asked to induce, Duce searches for an operation and suggests an application of the truncation operator which will save 12 symbols. The operation is valid if and only if everything having four legs is an elephant. The user can either answer affirmatively ("y"), negatively ("n") or ask for illustrative examples ("i"). If Duce is asked for illustrative examples it lists the instances adultelcpliant and babyelcphant. The suggestion although consistent with the limited universe of the examples, is too general, and is rejected. Puce continues its search and finds a slightly less advantageous truncation, which would save 11 symbols. The new suggestion, that anything with four legs and no wings is an elephant is similarly rejected. There is no particular mechanism for specialising over-generalised hypotheses. This is merely a by-product of the search mechanism. On the third attempt, Duce questions whether everything that has four legs and a trunk is an elephant. Since this is affirmed, Duce replaces all elephant rules by the new more general rule and returns to the "!-" prompt.

The second generalisation, concerning man is accepted first lime around, producing a saving of 9 symbols. In the third interaction Ducc finds that using the interconstruction operator, one symbol can. be saved by defining a new concept for all things which have two legs and no wings. The user can either reject this new concept ("n"), ask for illustrative examples ("i"), or give a name for the concept, The name "primate" is given to the concept. Duce goes on to suggest another new concept for all things which have a beak, two legs, a tail and wings. This concept is named "bird". When asked to search for another operator application Ducc comes back with the message, "No applicable transformation", meaning that none of the operators reduce the rule base. The time between each prompt in this example is in the order of one second.

Figure 3 shows the result of the transformations. Not only is the rule base more compact but also the new concepts have made the rules more conceptually transparent. For example, a blackbird is simply defined as a bird which is black. Note that the illustrative examples arc propagated to all new rules.

## 6. Even-parity.

According to Minsky and Papert (1969) the "parity" function is unlearnable by single-layer perceptions, The even-parity problem is that of recognising whether Strings of binary digits contain an even number of l's. Recent techniques using multi-layered perceptron networks (Rumelhart and McClelland 1986) have been shown lo be capable of learning parity effectively. However, in the paradigm of explicit rule formation, algorithms such as 1D3 (Quinlan 1982) and AQ11 (Michalski and Chilausky 1980) turn out to be rather inadequate when used to learn such functions. It has been shown (Muggleton 1986) that whereas single-level concept representations of parity have a description complexity which is necessarily non-polynomially dependent on the number of attributes, multi-level descriptions can be built whose size is only linearly dependent on the number of primitive attributes. Efficient multi-concept solutions inevitably rely on a divide-and-conquer approach. Thus the decision of the top-level concept is based on the combination of values of lower-level predicates. Each lower-level predicate has a domain which depends on a restricted subset of the total set of problem attributes.

Figure 4 depicts examples of 8-variable even-parity. The variables (or primitive attributes) are numbered vl to v8, and each it bound to a value from the set {f t} (rather than {0 1}). In the first example, the variables have even-parky, since all eight have the value t, i.e. an even number of variables arc bound to t. The "eg" part of the example shows a string of this form. Figure 5 shows the session in which Duce transforms the training set of figure 4 into the partial, hierarchical solution of figure 6. The responses arc based on

```
Pa7
    bxqsq hdchk
    rimmx spcop
    stlmt
    Delayed-queening-1
        bknwy bkon8
        dsopp mulch
        rxmsq skach
        skrxp wkna8
        Delayed-queening-2
            bkblk
            White-king-in-check-delay
                bkxcr bkxwp
                cntxt simpl
                skewr wknck
                wtoeg
            Skewer-threat
                bkspr rkxwp
                r2ar8 rkxwp
                wkcti wkovl
                wkpos
            Double-attack-threat-2
                bkona bkxwp
                bkxbq blxwp
                dwipd katri
                reskr
    Mate-threat-1
        cntxt wkna8
        Mate-and-double-attack
            bkblk bkxbq
            bkxcr katri
            thrsk
            Mate-and-double-attack-safe-from-promoted-queen
                dsopp qxmsq
                r2ar8 rkxwp
                wkcti wknck
        Mate-threat-2
            bkon8 bkspr
            bkxcr r2ar8
            skrxp thrsk
            wknck wkovi
            Mate-threat-safe-from-promoted-q
                bkxbq blxwp
                dsopp dwipd
                rkxwp rxmsq
                simpl skewr
                wtoeg
    Double-attack-1
        bkblk bkona
        bkspr bkxcr
        bkxwp blxwp
        dwipd skrxp
        Potential-double-attack-useful-to-black
            bknwy bkxbq
            cntxt katri
            r2ar8 wkcti
            wknck wkovl
            wkpos
```

**Figure 8. KPa7KR knowledge structure generated by Duce.**

a standard solution in which the variables arc recursively broken into two equal sized sets at each level. The total set of variables have even-parity if and only if both subsets have even-parity, or both have odd-parity, 'The first three concept suggestions do not follow this scheme, and arc rejected. The fourth is recognised as "the second-half of the variables have even parity" (sev). 'The user then affirmatively answers questions concerning the application of the absorption operator, The next suggested concept is named ffev or "first-half of the first-half of the variables are even". Given the original eight examples, Ducc's solution is generalised to cover 16 of the 256 possible instances. If presented initially with the complete instance set, Duce tends towards a solution consisting of an 8-ievel deep hierarchy in which levels are used to count the number of variables set to t.

## 7. Recreation of the Kl'a7KR structure.

Both the animal taxonomy and parity problem have highly restricted domains. The real test of Duce's capabilities has been the attempt to restructure Shapiro and Kopec's expert system (Shapiro 1983) for deciding whether positions within the chess endgame of King-and-Pawn-on-a7 v. King-and Rook (KPa7KR) are won-for-white or not. The domain contains around 200,000 positions Shapiro generated a database of all positions, labelling each with its minimax backup value of forced win-for-white or not. A set of 36 primitive board features were calculated for each position. Since many positions had the same feature vector and won-for-white value, the number of distinct examples was reduced to 3196. With this number of examples Duce's search-space for applying the first operation is $2^{3196}$ (see section 4), or approximately $10^{1009}$. Nilsson (1982) states that the complete game tree for chess has approximately $10^{120}$ nodes; even that well-known hard problem has a considerably smaller search space than that attempted here.

For the purposes of the experiment, Shapiro provided a randomly chosen board position for each example. Thus the initial rule base given to Ducc consisted of examples of the form

*(wonforwhtte featurel* A *feature2* A .. *feature36) eg (position)*

Two chess experts, Ivan Dratko and Tim Niblett, helped in giving oracle answers to questions asked by Ducc. The rule base started with 118,252 symbols. 'The first suggestion reduced 21,606 of these, a reduction of around 20%. After three questions, around 60% of the rule base had been reduced. After 41 transformations, the rule base had been reduced to 553 rules, and contained a total of 9050 symbols. At this point there were still applicable operations, but symbol reductions had been reduced to the low hundreds.

In questions 3 and 5, in which new concepts were introduced, the size of the common set of symbols, Int was too large for a comprehensible rule description. It is here that the illustrative board positions were indispensible. For this experiment, a domain-dependent graphics front-end was built into Ducc, which gave the user the ability to peruse a large number of board positions representing the concept and counter-concept Without this graphical device, new concepts could not have been recognised and named As it was, concepts were named with confidence within the presentation of 20 to 40 board positions. It was rarely necessary to reject new concepts and generalisations suggested by Ducc in the KPa7KR experiment.

Figure 7 shows the structure created manually by Shapiro and Kopec, which took an estimated 6 man months of effort. Figure 8 shows Ducc's solution. Ducc carried out the 41 oracle agreed transformations during a single working day. The computation time between each question was in the order of a minute. It should be noted that where Shapiro and Kopec have used nine hierarchically arranged concepts, Duce has used thirteen. Ducc's solution also contains 553 rules and 9050 symbols where Shapiro and Kopec's manually created solution contains the equivalent of around 225 productions and around 1000 symbols. Although Ducc's solution could have been made more compact by applying more transformations or by generating decision trees for each concept using 1D3, it seems unlikely to the author that this would have resulted in a solution which was as compact as that of Shapiro and Kopec.

By virtue of the operators used by Ducc, the KPa7KR solution is guaranteed correct by construction.

## 8. Discussion.

Duce *in* a program which, with the aid of a human oracle discovers useful new concepts. AM (Lenat 1979), an early concept discovery program, was criticised (Ritchie and Hanna 1984) for the obscurity of the techniques involved. Unlike AM, Duce uses a simple and explicit set of six operators to create and refine concepts. In addition the meaning-giving agent, implicitly present within any Machine Learning system, is explicitly represented as the oracle within Ducc.

Extensive search is used to decrease the number of questions asked by Ducc of the oracle. However, in what circumstances is the use of an oracle either justified or feasible? In this respect it is worth noting that on the basis of a meagre number of empirical studies the ratio of oracle rejections to acceptances seems to be inversely related to the percentage of examples provided from the domain. In the parity problem, where Duce was supplied with a sparse set of examples, a large number of rejections were necessary (figure 5). In the more complex KPa7KR chess domain, Duce was given an exhaustive set of examples, and required almost no rejections from the oracle. Thus it might be expected that in domains in which a moderate amount of example material is available the oracle would need to reject a moderate number of proposals. Further research is necessary to show the truth of this hypothesis.

Duce works with statements in prepositional logic. One way of extending the present work would be to attempt using similar techniques within other representations. Danerji (1986) is presently looking at the problem of constructive induction within first-order calculus. The author believes that techniques akin to those used in Ducc could be profitably employed in learning hierarchically definable context free grammars.

Acknowledgements.

I would like to thank Donald Michie for suggesting the application of Duce to the KPa7KR problem. Thanks are also due to Alcn Shapiro for supplying the chess example database, Ivan Bratko and Tim Niblctt for acting as chess oracles, Pete Mowforth for suggesting the name "Duce" and Michael Bain, Dave Haussler, Claude Sammut and Wray Buntine for helpful discussion.

References.

1 Banerji R.B., Changing language while learning recursive descriptions from examples. *Machine Learning: A Guide to Current Research*. Ed. T. Mitchell, J. Carbondl and R. Michalski. Dordrecht: Kluwer, (1986), pp. 3-9.

2 Durst all R.M. and Darlington J., A transformation system for developing recursive programs. *Journal of the Association/or Computing Machinery,* Vol. 24, (1977), pp. 44-67.

3 Dershowilz N., Synthesis by completion. *Proceedings of the Ninth International Conference on Artificial Intelligence*. Los Altos, CA: Kaufmann, (1985), pp. 208-214.

4 Feigenbaum E.A, Themes and case studies of knowledge engineering. *Expert systems in the Micro-electronic Age,* Ed. D. Michie. Edinburgh: Edinburgh University Press, (1979), pp. 3-25.

5 Fu L.M and Buchanan B.G., Learning intermediate concepts in constructing a hierarchical knowledge base. *Proceedings of the Ninth International Joint Conference on Artificial Intelligence.* Los Altos, CA: Kaufmann, (1985), pp. 659-666.

6 Lenat D.B., On automated scientific theory formation: a case study using the AM program. *Machine Intelligence 9.* Ed. J.E.Hayes, D.Michie and L I. Mikulich. Chichester Horwood, (1979), pp. 251-283.

7 Michalski R.S. and Chilausky R.L., Learning by being told and learning from examples: an experimental comparison of the two methods of knowledge acquisition in the context of developing an expert system for soybean disease diagnosis. *International Journal of Policy Analysis and Information Systems.* 4(2): (1980), pp. 126-161.

8 Michalski R.S. and Stepp R., Revealing conceptual structure *in* data by inductive *inference, Machine Intelligence 10.* Ed. J.E. Hayes, D. Michie and Y-H Pao. Chichester Horwood. (1982), pp. 173-196.

9 Michalski R.S., Understanding the nature of learning: issues and research directions. *Machine Learning: An Artificial Intelligence Approach.* Vol. 2. Eds Michalski R.S., Carbondl J.O. and Mitchell R.M. Los Altos, CA: Kaufmanp (1986), pp. 3-25.

10 Minsky M. and Papert S., *Perceptrons.* Cambridge, MA: MIT Press, 1969.

11 Muggleton *S.IL, Inductive Acquisition of Expert Knowledge.* Edinburgh: Edinburgh University (Ph.D. thesis), 1986.

12 Nilaaon N.J., *Principles of Artificial Intelligence.* Berlin Springer-Verlag, (1982), p. 115.

13 Paterson *A, An attempt to use CLUSTER to synthesise humanly intelligible subproblems for the KPK chess endgame.* Urbana, 1L: Univ. Illinois, (UIUCDCS-R-83-1156), 1983.

14 Quinlan J.R., Semi-autonomous acquisition of pattern-based knowledge. *Introductory readings in expert systems.* Ed. D. Michie. New York: Gordon & Breach. (1982), pp 192-207.

15 Rendoll, L., Substantial constructive induction using layered information compression: tractable feature formation in search. *Proceedings of the Ninth International Joint Conference on Artificial Intelligence.* Los Altos, CA: Kaufmann, (1985), pp. 650-658.

16 Ritchie O.D. and Hanna P.K., AM: a case study in A.I. methodology. *Artificial Intelligence.* 23:3 (1984), pp 249-268.

17 Rumelhart D.E. and McClelland J.L., Learning internal representations by error propagation. *Parallel Distributed Processing. Explorations in the Micro-Structure of Cognition. Vol. 1: Foundations.* Cambridge, MA: MIT Press, (1986), pp. 318-362.

18 Sammut C. and Banerji R.B., Learning concepts by asking questions. *Machine Learning: An Artificial Intelligence Approach. Vol. 2.* Eds. Michalski R.S., Carbondl J.O and Mitchell R.M. Los Altos, CA: Kaufmann, (1986), pp. 167-192.

19 Shapiro A.D., *The Role of Structured Induction in Expert Systems.* Edinburgh: Edinburgh University (Ph.D. thesis), 1983.