MATHEMATICAL PROBLEMS IN COMPUTATION THEORY
BANACH CENTER PUBLICATIONS, VOLUME 21
PWN – POLISH SCIENTIFIC PUBLISHERS
WARSAW 1988

COALGEBRAIC DECISION THEORY*

J. R. B. COCKETT

Department of Computer Science, University of Tennessee, Knoxville, U.S.A.

Decision trees are terms of a coalgebraic theory, called a *decision theory*. The terms of a decision theory may be manipulated into two important forms: by simple reduction into a simply reduced form and by reduction into an irreducible form. Simple reduction provides a method for determining equality of terms. An irreducible term is always optimally efficient for some problem. Irreducibility provides a key to the more general problem of code optimization.

0. Introduction

Decision trees have traditionally been used for the representation of taxonomic knowledge and to drive interactive systems (for recent work in this direction see [1], [8] and [9]). With the advent of rule-based systems, this method of representation has become less popular. One reason for this decline in popularity has been the lack of a well-known calculus of manipulations for decision trees. This is regrettable as many applications have a natural representation as a decision tree. Furthermore there is a very well-behaved calculus of manipulations.

A decision corresponds to an attribute map. The value of an attribute gives a choice of branching at a decision. Conversely, a choice of branching at a decision gives an attribute value. Thus decision trees are closely allied to information systems, which provides another application of decision trees. They are an efficient data structure for holding database information. The

* This research was supported by the U.S. Department of Energy, Office of Energy Storage and Distribution, Electric Energy Systems Program, through the Oak Ridge National Laboratory, Systems Engineering for Power Program. Activities at the University of Tennessee where supported by sucontract 37B-07685C/S13.

retrieval properties and the database aspects of decision trees have been the subject of several investigations (see [2], [3] and [4]).

Decision trees, in their most general form, occur as the control structure (flow diagram) of a program. Thus every computer scientist and programmer uses decision trees. Yet at the same time little is generally known about these structures and it is hard to find a reference which talks substantially about decision trees.

The theory of discrete decisions has two significant merits. Firstly, it provides a description of decision trees which is purely algebraic. This provides a good basis from which properties can be proven and the correctness of algorithms can be established. Secondly, it introduces some basic forms for decision trees. The most important of these forms is the irreducible form, which is central to optimization results for decision trees (see [5] and [6]) and whence, more generally, for program code.

The problem of optimization has, of course, been the subject of a considerable volume of research (see references in [10]). Traditional treatments have been stymied by the lack of a calculus to directly manipulate the terms. They rely on general techniques, such as searching using branch and bound, and do not taken full advantage of the structure inherent in the situation.

The optimization results mentioned above assume that the decisions are independent in the sense that the possible outcomes of any given decision are not affected by the outcomes of other decisions. This is rarely the case in practice (e.g. in expert systems). Current work indicates that all the basic results may be extended to the case in which dependencies are present.

This paper provides a brief survey of the current state of the theory of discrete decisions [7]. That it has bearing on program code optimization, database design, and interactive system design implies that it may well be a rather fundamental part of that theory which underpins practical computer science.

1. Coalgebraic preliminaries

A decision tree may be viewed as an expression or term of a coalgebraic theory. A coalgebra, for such a theory, consists of a base set S together with a collection Q of cooperations which act on the set. A cooperation, q, consists of map from the base set S to an n-fold coproduct, or disjoint union, of S:

$$q\colon\ S \rightarrow S + \ldots + S.$$

Cooperations can be manipulated in the same way as operations. The primary difference is that their composition is the "wrong way round" (that is they are the formal dual of operations). However, there are significant differences between coalgebras in Sets (or indeed in any topos) and algebras in Sets as cooperations interact with the underlying structure of Sets.

A common way in which a cooperation can arise is when an attribute map is present. Thus, a cooperation q may result from the presence of a map f from the base set S to the product of that set with a finite cardinal, $|n|$. This is because this product is isomorphic, via k (see below), to the n-fold coproduct (disjoint union) of S:

$$f\colon S \to S \times |n|, \qquad k\colon S \times |n| \to S + \ldots + S, \qquad q = f \cdot k\colon S \to S + \ldots + S.$$

Conversely given a cooperation, q, by composing with the inverse of the isomorphism k a map, $f = q \cdot k^{-1}\colon S \to S^* |n|$ can be obtained. Thus cooperations and maps to this product correspond precisely.

The effect of the map f as it is a map to a product, may be split into two parts. The first effect is an endomorphism of S, which is called the *trace*, $\mathrm{tr}(q)$, of the cooperation q. The second effect is a splitting of S into n buckets, which is called the *attribute assignment*, $\mathrm{at}(q)$, of the cooperation q. The set $|n|$ may be regarded to be a set of attributes. These two effects are obtained by composing with the projections from the product:

$$\mathrm{tr}(q) = f \cdot p_0\colon S \to S, \qquad \mathrm{at}(q) = f \cdot p_1\colon S \to |n|.$$

The map f may then equivalently be expressed as the composition:

$$\langle i(S), \mathrm{at}(q)\rangle \cdot \langle \mathrm{tr}(q) \times i(|n|)\rangle\colon S \to S \times |n|,$$

where $i(S)$ and $i(|n|)$ are the identity maps on S and $|n|$ respectively.

The first component of this composition $\langle i(S), \mathrm{at}(q)\rangle$ has codomain $S \times |n|$ and gives rise to a cooperation $\mathrm{dc}(q) = \langle i(S), \mathrm{at}(f)\rangle \cdot k$. Notice that $\mathrm{dc}(q)$ relies only on the attribute assignment. This cooperation is idempotent. For this reason it is called the *idempotent factor* of the cooperation q.

It is idempotent in the normal algebraic sense, that is

$$\mathrm{dc}(q) \cdot \langle x'x' \ldots 'x\rangle = x,$$

where the list is the codiagonal map. The dual of this, the idempotence law for an operation, should be familiar: lattice conjunction is idempotent. A cooperation is clearly idempotent if and only if its trace is the identity map. The trace of $\langle i(S), \mathrm{at}(q)\rangle \cdot k$ is obviously the identity map thus it is idempotent.

An idempotent cooperation is a decision. Decisions arise from attribute assignments alone. If A is a finite set then, given an attribute assignment $a\colon S \to A$ an idempotent cooperation $\langle i(S), a\rangle \cdot k\colon S \to S + \ldots + S$, can be formed. The converse is also true: an idempotent cooperation gives rise to an attribute map.

LEMMA. *Any cooperation q on a set S may be uniquely factored into a decision* $\mathrm{dc}(q)$ *followed by the application of the trace* $\mathrm{tr}(q)$ *to each component*:

$$\mathrm{dc}(q)\cdot\langle \mathrm{tr}(q)+\ldots+\mathrm{tr}(q)\rangle.$$

Proof. $q = f\cdot k = \langle i(S), \mathrm{at}(q)\rangle\cdot\langle \mathrm{tr}(q)\times i(|n|)\rangle$
$= \langle i(S), \mathrm{at}(q)\rangle\cdot k, \quad k^{-1}\cdot\langle \mathrm{tr}(q)\times i(|n|)\rangle\cdot k$
$= \mathrm{dc}(q)\cdot\langle \mathrm{tr}(q)+\ldots+\mathrm{tr}(q)\rangle.$ ∎

The existence of this unique factorization implies that every coalgebraic "theory" can be "decision extended" in the sense that decisions can be added so that every factorization can be expressed within the theory. Thus, an extension of the theory can be formed in which the unique idempotent factors are already present. From this it can be seen that idempotent cooperations form a rather natural focus of study. In particular all coalgebraic theories give rise to a related theory consisting of the idempotent cooperations of this "decision completion". A theory in which all the cooperations are idempotent, and thus decisions, is called a decision theory.

An idempotent cooperation is the abstract notion of a discrete decision. The study of theories whose cooperations are all idempotent is thus the study of discrete decision processes.

A fundamental result of these theories is that the expressions, or terms, of the theory are also decisions which themselves satisfy all the identities which the primitive decisions do. While it is clear that every decision must satisfy the idempotence law,

[D.1] $$q\cdot\langle x'x'\ldots'x\rangle = x,$$

the fact that every decision is idempotent actually forces two other important identities to hold.

PROPOSITION. *In any coalgebra on the set S*:

(i) *The composition of decisions is a decision,*

(ii) *Every decision distributes over every other decision, that is* [D.2] (*below*) *is satisfied,*

(iii) *Every decision satisfies the repetition law,* [D.3] (*below*). ∎

It is not hard to see that a distibutive law holds between decisions. In the binary case this is the following identity:

$$q_1\cdot\langle x_1{}'q_2\cdot\langle y_1{}'y_2\rangle\rangle = q_2\cdot\langle q_1\cdot\langle x_1{}'y_1\rangle{}'q_1\cdot\langle x_1{}'y_2\rangle\rangle.$$

The generalization of this to arbitrary arity produces the general distributive law:

[D.2] $$q_1\cdot\langle x_1{}'\ldots{}'x_{(r-1)}{}'q_2\cdot\langle y_1{}'\ldots{}'y_m\rangle{}'x_{(r+1)}{}'\ldots{}'x_n\rangle$$
$$= q_2\cdot\langle q_1\cdot\langle x_1{}'\ldots{}'x_{(r-1)}{}'y_1{}'x_{(r+1)}{}'\ldots$$
$$\ldots x_n\rangle{}'\ldots{}'q_1\,\langle x_1{}'\ldots{}'x_{(r-1)}{}'y_m{}'x_{(r+1)}{}'\ldots{}'x_n\rangle\rangle.$$

The repetition law is the following identity:

$$[\text{D.3}] \quad q_1 \cdot \langle x_1{}' \ldots {}' x_{(r-1)}{}' q_1 \cdot \langle y_1{}' \ldots {}' y_n \rangle' x_{(r+1)}{}' \ldots {}' x_n \rangle = q_1 \cdot \langle x_1{}' \ldots {}' x_{(r-1)}{}' y_r{}' x_{(r+1)}{}' \ldots {}' x_n \rangle.$$

These identities [D.1]–[D.3] are really commonsense facts about decisions. [D.1] states that if making a decision makes no difference, then there is no need to make it. [D.2] allows the order of decisions to be changed. While [D.3] says there is no point in asking for the same decision twice in a row: the same answer will be obtained both times.

These three identities [D.1]–[D.3] are actually sufficient to describe all possible valid manipulations of decisions, this is proven in [7]. For this reason expressions which are [D.1]–[D.3] equivalent shall be said to be *decision equivalent*.

It is reasonable to be curious as to how these other identities arose. Clearly one cannot somehow perform some algebraic trick to obtain the repetition law from idempotence. This highlights the significant difference between algebraic and coalgebraic theories. In the coalgebraic setting the idempotent identity does imply distributivity and repetition. This is because the underlying structure of **Sets** interacts with the cooperations.

The significance of the completeness of the identities [D.1]–[D.3] is now clear. A complete characterization of the manipulations of decision expressions in terms of the algebraic identities which hold is essential. This avoids recourse to pulling some vital "fact" about coalgebras out of the air whenever the going gets tough. It is certainly not obvious that such a complete characterization exists. However, it does and it is given by [D.1]–[D.3].

The use of the term coalgebra suggests the existence of a right-adjoint to the obvious underlying functor from the category of coalgebras. In fact such an adjoint does exist. Furthermore the category of coalgebras is a well known category. It is the slice category **Sets**/Arg: this is decribed in [1]. The underlying functor is therefore a left-adjoint and so preserves colimits. It should not be expected to preserve limits and does not in general. These matters are discussed in more detail in [7].

There may be further identities which are satisfied by a particular decision coalgebra. These can be used to determine a subvariety of decision theories. In practice these varietal identities are rather important as they provide the description of the dependencies which hold between attributes. Although this paper does not deal with the effect of these identities, all the basic results can be generalized to the situation in which they are present.

2. Simple forms for decision expressions

In fact the proposition, above, has some further rather important ramifications. It is intuitively obvious in a decision tree that there is no point in repeating a question on any path from root to leaf. However, it is not immediately obvious how this fact can be obtained from the identities [D.1]–[D.3]. Repetition refers only to an immediate repetition not to a case in which there are a series of intermediate decisions. In fact this more general notion of repetition is a consequence of the algebraic identities and this may be seen from a rather subtle application of this proposition.

LEMMA (REPEAT REDUCTION LEMMA).

$$q\cdot\langle x_1{}'\ldots{}'x_{(r-1)}{}'\,W\cdot\langle y_1{}'\ldots{}'q\cdot\langle z_1{}'\ldots{}'z_n\rangle'\ldots{}'y_m\rangle'\,x_{(r+1)}{}'\ldots{}'x_n\rangle$$
$$= q\cdot\langle x_1{}'\ldots{}'x_{(r-1)}{}'\,W\cdot\langle y_1{}'\ldots{}'z_r{}'\ldots{}'y_m\rangle'\,x_{(r+1)}{}'\ldots{}'x_n\rangle,$$

where W is any expression, or term, of a decision theory in variables $v_1, \ldots, v_n$ and $W\cdot\langle t_1{}'\ldots{}'t_n\rangle$ denotes that term with variable v_i substituted by t_i.

Proof. Notice that $W\cdot\langle v_1{}'\ldots{}'v_n\rangle$ is a decision and therefore satisfies all the identities of decisions. Thus we may distribute the inner occurrence of q forward to get from the RHS:

$$q\cdot\langle x_1{}'\ldots{}'x_{(r-1)}{}'\,q\cdot\langle W\cdot\langle y_1{}'\ldots{}'z_1{}'\ldots{}'y_m\rangle'\ldots\rangle'\,x_{(r+1)}{}'\ldots{}'x_n\rangle.$$

Now use the repetition law to obtain the LHS. ∎

This shows that arbitrary repetitions can be eliminated. It also shows that the idea of eliminating repeated decisions which does not seem at first sight to be an algebraic concept is in fact a direct result of algebraic identities.

This raises the question of whether it is possible to rewrite an arbitrary decision expression into a canonical form in which there are no repetitions. This, among other things, can be done. Consider the following forms and associated rewriting schemes for decision expressions:

A. *Idempotent reduced*:
– no subexpression of the form $q\,\langle x'\,x'\ldots{}'x\rangle$,
– term rewriting rule: $q\,\langle x'\,x'\ldots{}'x\rangle \Rightarrow x$,
– confluent: $t \overset{*}{\Rightarrow} \mathrm{id}(t)$.

B. *Repetition reduced*:
– no repeated cooperations on any path to the leaf,
– schema term rewriting rule:

$$q\cdot\langle x_1{}'\ldots{}'x_{(r-1)}{}'\,W\cdot\langle y_1{}'\ldots{}'q\cdot\langle z_1{}'\ldots{}'z_n\rangle'\ldots{}'y_m\rangle'\ldots{}'x_n\rangle$$
$$\Rightarrow q\cdot\langle x_1{}'\ldots{}'x_{(r-1)}{}'\,W\cdot\langle y_1{}'\ldots{}'z_r{}'\ldots{}'y_m\rangle'\ldots{}'x_n\rangle,$$

– confluent: $t \overset{*}{\Rightarrow} \mathrm{rpt}(t)$

A+B. *Simply reduced*:
– no subexpressions which can be idempotent reduced, no repetitions,
– not confluent: $t \overset{*}{\Rightarrow} \mathrm{id}(\mathrm{rpt}(t))$.

It is easy to show that A, B are confluent. However, A+B does not give a confluent rewriting scheme. In order to produce an expression in simply reduced form it is usual, therefore, to first perform repeat reduction and then apply idempotent reduction.

Obviously if two expressions happen to have the same simply reduced form then the expressions must have been decision equivalent. However, it is not the case that if two expressions are decision equivalent that they will necessarily have the same simply reduced form. Despite this, simple reduction does provide a method of determining equality.

Consider two simply reduced decision expressions:

$$E_1 \cdot \langle x_1' \ldots ' x_n \rangle \quad \text{and} \quad E_2 \cdot \langle x_1' \ldots ' x_n \rangle.$$

By idempotence we have:

$$E_1 \cdot \langle E_2 \cdot \langle x_1' \ldots ' x_n \rangle' \ldots ' E_2 \cdot \langle x_1' \ldots ' x_n \rangle \rangle = E_2 \cdot \langle x_1' \ldots ' x_n \rangle.$$

However, if the LHS is repetition reduced then it is clear that the structure of E_1 will be left intact under the remnants of the E_2's at the leaves. Equality occurs precisely when the remnants are such that an idempotent reduction collapses the remnants down to the appropriate variable at the leaf of E_1.

LEMMA. *E_1 and E_2 are decision equivalent if and only if*

$$\operatorname{id}(\operatorname{rpt}(E_1 \cdot \langle x_1' \ldots ' x_n \rangle))$$

$$\equiv$$

$$\operatorname{id}(\operatorname{rpt}(E_1 \cdot \langle E_2 \cdot \langle x_1' \ldots ' x_n \rangle' \ldots ' E_2 \cdot \langle x_1' \ldots ' x_n \rangle \rangle)). \blacksquare$$

Here "$\equiv$" means structural equality. A full proof of this result is given in [6] or [7]. Simple reduction, besides providing the basis for an algorithm for determining decision equivalence, affords the builder of an interactive system an inexpensive tool to make the interface more productive. However, it is a rather trivial step compared to the steps which are introduced in the next section.

3. Irreducible decision expressions

A criterion for a decision theory is a cost function on the expressions of the theory. It associates a real number with each decision expression to indicate the cost of evaluation, storage or testing [10]. The problem of optimizing a decision expression, with respect to a criterion, is that of finding a decision equivalent expression of minimal cost. Clearly this problem is of considerable computational interest.

A further identity of decision expressions, which turns out to be central to this optimization problem and whose significance was, for example, exploited by K. Chen and Z. Ras [3], is transposition:

$$[\text{D.4}] \quad q_1 \cdot \langle q_2 \cdot \langle x_{11}' \ldots ' x_{1n} \rangle' \ldots ' q_2 \cdot \langle x_{m1}' \ldots ' x_{mn} \rangle \rangle$$
$$= q_2 \cdot \langle q_1 \cdot \langle x_{11}' \ldots ' x_{m1} \rangle' \ldots ' q_1 \cdot \langle x_{1n}' \ldots ' x_{mn} \rangle \rangle.$$

As the identities [D.1]–[D.3] are complete they imply this identity. In fact the distributive law, [D.2], could be replaced by this identity to give an alternate axiomatization of decision theories.

For any expression the set of all paths from the root to a leaf can be considered. Each path must be regarded as a pair: a list of decision outcomes and the variable at the leaf to which those outcomes lead. If each list of decision outcomes is regarded to be an unordered bag, then it may reasonably be asked what expressions with equivalent sets of path bags have in common.

PROPOSITION (RAS'S LEMMA). *Two repeat reduced expressions have equivalent sets of path bags if and only if they are transposition equivalent.* ∎

This result gives a simple algebraic characterization of a complex property which two trees may have in common. In trying to arrange a productive interaction, or computation, it is often not the particular order of the decisions which matters but rather which decisions will actually have to be made at the end of the day to reach the various conclusions. Thus interactions (or computations) which always result in the same path bags will be equally desirable. Such decision expressions may be obtained from each other by using transposition alone and are thus called transposition equivalent. If expressions are transposition equivalent, then they are decision equivalent, however, the converse is certainly not true. It is clear that in trying to optimize a decision expression it is necessary only to optimize up to transposition equivalence.

In an actual application it is often difficult to specify a precise criterion. Thus, optimization to a particular criterion is often of little practical value. An alternative approach is to optimize with respect to a class of criteria. The problem with such a strategy is that two different particular criteria of the class may have very different optimal solutions. This implies that such an optimization should allow for the possibility that there are many possible optimal solutions.

A preorder is a transitive and reflexive but not necessarily anti-symmetric relation. A preorder may be generated from any relation by forming the transitive-reflexive closure. As a preorder is not necessarily anti-symmetric; it is possible for two different elements to be both greater and less than each other. Such elements are said to be equivalent in the preorder.

A preorder may be generated on the expressions of a decision theory as follows:

DEFINITION. If D is a decision theory and $T(D)$ its terms, then the reasonable preorder ${}_r\!\leqslant$ is generated by:

[RP.1] $q_1 \cdot \langle \ldots 'x_1' \ldots \rangle {}_r\!\leqslant q_1 \cdot \langle \ldots 'x_2' \ldots \rangle$ whenever $x_1 \leqslant x_2$ (composable),

[RP.2] $x_1 {}_r\!\leqslant q_1 \cdot \langle x_1' \ldots 'x_1 \rangle$ (idempotent well-ordered),

[RP.3] $q_1 \cdot \langle q_2 \cdot \langle x_{11}' \ldots 'x_{1n} \rangle' \ldots 'q_2 \cdot \langle x_{m1}' \ldots 'x_{mn} \rangle \rangle$
$${}_r\!\leqslant q_2 \cdot \langle q_1 \cdot \langle x_{11}' \ldots 'x_{m1} \rangle' \ldots 'q_1 \cdot \langle x_{1n}' \ldots 'x_{mn} \rangle \rangle$$
(transposition invariant),

[RP.4] $q \cdot \langle \ldots 'x_{r-1}'W \cdot \langle \ldots 'q \cdot \langle \ldots 'y_r' \ldots \rangle' \ldots \rangle' \ldots \rangle$
$$\geqslant_r q \cdot \langle \ldots 'x_{r-1}'W \cdot \langle \ldots 'y_r' \ldots \rangle' \ldots \rangle$$ (repeat reducing).

Notice that two expressions are comparable only if they express the intuitive notion of when one decision expression is better than another. Thus, if a subexpression has been improved then it is reasonable to expect that the whole expression will be no worse, [RP.1]. Similarly eliminating a decision by repetition or idempotence should improve the situation, or at least make it no worse, [RP.2] and [RP.3]. If t_1 and t_2 are two expressions and $t_1 \geqslant_r t_2$ and $t_1 {}_r\!\leqslant t_2$, then they are reasonable equivalent, denoted $t_1 \simeq_r t_2$, and this occurs if and only if the terms are transposition equivalent.

The idea is that any reasonable criterion must preserve this reasonable preorder. Thus we may study the problem of optimizing a decision expression given the assumption that the criterion is reasonable in this sense. It is easy to see that, for criteria which are reasonable, simple reduction will never increase the cost of an expression. Thus the cost of an expression t is never less than the cost of its simply reduced form $\mathrm{id}(\mathrm{rpt}(t))$. This is a desirable property as simple reduction seems intuitively to be an obvious source of improvement.

A minimal element in a preorder is an element for which every element below is actually equivalent. A solution to optimization with respect to a reasonable criterion will always fall on a minimal element in the reasonable preorder. Finding decision equivalent terms which are minimal in this preorder will therefore be a crucial step in optimization with respect to any reasonable criterion. A term which is minimal in the reasonable preorder is said to be *irreducible*.

A sensible step in optimizing a decision expression would therefore be to find a decision equivalent irreducible term. It is natural to inquire whether any further steps can be taken in this general optimization problem. That is whether irreducibles can be ordered in a nontrivial way to agree with every reasonable cost criterion. The answer is no. Thus one cannot optimize at this general level beyond finding irreducibles.

For the class of reasonable criteria the problem of optimization amounts to finding a suitable decision equivalent irreducible. It would therefore be useful if they could be easily identified. If a term is repeat reduced then any further reduction in the reasonable preorder must use idempotent reduction. This actually gives rise to a characterization of irreducibles.

LEMMA. *An expression is irreducible if and only if every transposition equivalent term is simply reduced.* ∎

Notice that this means that the expression must itself be simply reduced as it is transposition equivalent to itself. This indicates how one can test for irreducibility: generate all the transposes of the expression and check that they are all simply reduced. In practice this may be a rather difficult thing to do as the number of transposes can be very large. In [6] a more efficient approach to this problem is described. It suffices here to observe that irreducibles can be identified relatively easily.

Given any decision expression there is always an irreducible which is below it in the reasonable preorder. A process which finds such an irreducible is called a reduction. One way of doing this is by simply reducing transposes. Given an expression t_1 it is clear that any transpose t_2 will have the same cost. If such a transpose t_2 is not simply reduced, then without increasing cost it can be simply reduced to t_3. Clearly the cost of t_3 is no more than the cost of t_1. Thus, by successively simply reducing tranposes, any term can eventually be reduced to an irreducible which is guaranteed to cost no more than the original expression. This is the idea behind the reduction algorithm described in [6].

A sophisticated step which can be undertaken when designing an interaction (or computation) is to ensure that it is irreducible. If it is not, then it can be reduced. Reduction is a nondeterministic optimization of a decision tree and can also be applied to any acyclic code segment. The computational cost of this process is dominated by the size of the tree. This makes it a computationally feasible step to undertake.

While intuitively a reasonable criterion has appeal there is no guarantee that in practice the criteria which are used will be reasonable. However, there is a important criterion whose relation to reasonableness has been investigated. This criterion is the expected testing cost criterion.

If, in an expression, the probability that any path will be followed and the cost associated with making each decision is known, then the expected testing, or evaluation, cost can be calculated. This value is the sum over each path of the costs of the decisions on the path times the probability of the path itself. The resulting criterion is reasonable and is called the expected testing cost criterion.

In principle, there is an expected testing cost associated with every applied decision expression. Thus it would be comforting if there was a welldefined relation between irreducibles and the various expected testing cost criteria which might apply. Such a result exists:

PROPOSITION (SEPARATION OF IRREDUCIBLES). *If t_1 and t_2 are two decision equivalent irreducibles which are not transposition equivalent, then there is an expected testing cost criterion C such that*

(i) $C(t_1) > C(t_2)$ *and*

(ii) $C(t_2)$ *is the minimal cost achievable.* ∎

This means that given that one does not know, for one reason or another, the cost and probability distributions required to obtain the expected testing cost one cannot do better than to reduce the term to an irreducible.

4. Concluding remarks

The description of decisions in algebraic terms has a number of benefits. Perhaps the most significant is an algebraic approach to code optimization. This seems to have significant advantages over the more traditional graph theoretical and general searching techniques. Current research is being undertaken to extend this approach to a full programming environment.

The fact that it is rarely possible to provide a full specification of an expected testing cost criterion makes this approach very attractive. This difficulty cannot be emphasized enough as the following two examples illustrate.

First consider the situation for code optimization. The primitive decisions, being basic machine operations, have a definite time stamp. However, the expected cost of a computation is also determined by the probabilities of the paths of the computation. These probabilities are determined by use and the number of different paths involved may well inhibit any attempts at data collection. Furthermore, if the decisions are actually subroutines, then the embedded decisions in these may make the actual cost of the decisions context sensitive. Thus, in code optimization, associating costs to nonprimitive decisions can be difficult while determining path probabilities may simply not be feasible.

For interactive systems the situation is no better: the cost of a decision should be a measure of how much difficulty a user has in providing that decision. This is subjective and difficult to measure. Thus again the information to calculate a precise expected testing cost is hard to obtain if not simply unavailable.

In the latter case it may be feasible to collect path statistics. However, these statistics can, unfortunately, fail to give sufficient information to justify an optimization. The reason for this is that a rearrangement of the process can produce paths whose probabilities are not determined by the usage of the old process.

It would seem that the best that can be achieved is to perform a reduction. However, the situation is slightly more complex. Often to force an irreducible to be optimal it is necessary to assume a heavily biased probability and cost function (they may be extremely sparse). Thus some irreducibles are more robust than others. A current area of research concerns robust optimization.

An irreducible is a robust optimal if a perturbation of the cost functions and the probabilities will not cause its optimal properties to evaporate. This means, in particular, that there must be an open set in the space of cost/probability assignments in which it is optimal. The concept of robustness can cover many different models of cost/probability perturbation: for example path probability disappearance. The robustness properties of irreducibles depends on the perturbation model.

In the end, out of the optimization, one wishes to obtain irreducibles which are robust to as many perturbation in the cost description as possible and understand the relative advantages of each irreducible. There remains much work to be done to elucidate these relative advantages.

References

[1] J. D. Birdwell, J. R. B. Cockett, A. J. Laub, M. Athans and L. Hatfield, *Expert Systems Techniques in a Computer-Based Control Systems Analysis and Design Environment*, 3rd IFAC Symposium on Computer Aided Design and Engineering Systems, Lyngby, Denmark 1985.

[2] K. Chen and Z. Ras, *Homogeneous Information Trees*. Fundamenta Informaticae 8 (1985), 123–149.

[3] –, –, *An Axiomatic Theory of Information Trees*, Proceedings of the 1985 CISS, 636–640.

[4] –, –, *Dynamic Hierarchical Data Bases*, Proceedings of the 1983 ICAA in Taipei, Taiwan 1983, 450–456.

[5] J. R. B. Cockett and J. Herrera, *Prime Rule-Based Systems Give Inadequate Control*, Univ. Tenn. at Knoxville, Dept. of Comp. Sci., Tech. Report CS-84-60 (1985).

[6] J. R. B. Cockett, *Decision expression optimization*, Fund. Inform. 10 (1987), 93–114.

[7] –, *Discrete decision theory: Manipulations*, Theoret. Comput. Sci. 54 (1987), 215–236.

[8] –, *File handling for detail, extent and subtasks in the implementation of decision processes*, Inform. Sci. 37 (1985), 157–168.

[9] S. R. Heard, DECIDE: *A Decision Expression Interpreter*, Univ. Tenn. at Knoxville, Dept. of Comp. Sci. (1985), M. S. Thesis.

[10] B. M. E. Moret, *Decision Trees and Diagrams*, ACM Comp. Surv. 14, 4 (1982).

Presented to the semester
Mathematical Problems in Computation Theory
September 16–December 14, 1985