A Primer on Economic Choice Automata

Mark R. Johnson

Abstract This paper presents a development of the transformation semigroup of economic choice automata as a subgroup of the semigroup (monoid) of partial functions defined over the states of a finite state machine. The classes of consistency behavior considered are those rationalized by linear orders, weak orders, quasi-transitive relations and non-rationalizable path independent choice functions. For each of these classes of choice behavior, a particular class of lattice is identified as the action semigroup that drives the automaton. Given these characterizations, several features of the choice behavior are considered. In particular, the simplifying interval property of path independent choice, the importance of the distributive property of quasi-transitive rational choice in reducing the complexity of dynamic choice is addressed. Based on the algebraic structure of semiautomata implementing path independent choice functions it is possible to rank these semiautomata by the mathematical power required to implement a particular class of choice functions. This provides a means for ranking these machines by their "implementation complexity". Dually, the computational complexity of constructing a semiautomaton that implements a particular class of choice functions is investigated. It is seen that these complexities are inversely related.

Keywords Automata • Choice functions • Computational complexity • Implementation complexity • Semiautomata

1 Introduction

Notions of complexity lie at the core of our thinking about economic decision making. Generally, there is a belief that differences in complexity will affect the behavior of individuals or the structure of economic institutions. For example, in an elementary, unchanging world, a simple rule of thumb may be an adequate and appropriate method for making decisions. In a more complicated, rapidly changing

M.R. Johnson (✉)
Department of Finance and Economics, A. B. Freeman School of Business, Tulane University, New Orleans, LA 70118, USA
e-mail: mjohnso@wave.tulane.edu

C. Binder et al. (eds.), *Individual and Collective Choice and Social Welfare*, Studies in Choice and Welfare, DOI 10.1007/978-3-662-46439-7_5

world, the same simple rule of thumb may be quickly over come and lead to mistakes. Similarly, an overly complicated corporate decision structure involving many people and elaborate departmental check-offs might be too expensive and reduce the firm profits. The discussion below identifies the limitations imposed on the complexity of choice by economic consistency axioms.

Simon was one of the first economists explicitly to incorporate constraints on the information processing capacities of individuals and firms. For Simon [56], there were several ways that information processing limitations could affect economic activity and he categorized models incorporating these limitations as "theories of bounded rationality". While Simon was one of the early articulators on the importance of information processing to economic behavior and structure, other economists have appealed to these arguments as well. Most have adhered to Simon's view that different economic structures might have different complexities, and their writings have reflected a shared belief that complexities can either be scaled or, at least, compared. Among other early examples of information processing considerations in the model of individual choice is Strotz's [57, 58] exchange with Gorman [25] on the separability property for utility functions. In that exchange, Strotz maintained that separability was desirable because it simplified the consumer budgeting problem. A more recent example of information processing impacts on economic models is Auman's [4] suggestion that players in a game be modeled as automata and that the complexity measure based on the number of states in the automaton required to implement each strategy be used to separate classes of strategies.[1] Rubinstein [54] was one of many to follow up on this suggestion. With respect to organizational structure, Radner [53] has suggested that information processing is a major part of a corporation's "management" activities and that the hierarchical structure of a firm arises, at least in part because of this structure's efficiency at processing decentralized information.[2] Other authors also have made links between collective decision-making structures and complexity issues.[3]

A number of authors have suggested that one way information processing costs affect economic behavior is through the consistency axioms either satisfied by or

[1] An early survey of the literature growing out of this suggestion is offered in Kalai [40] while Chatterjee and Sabourian [15] offer a more recent treatment.

[2] Radner [53] also suggested that "the costliness of information processing contributes to organizational economies or diseconomies of scale". This idea that there are information costs contributing to the operational costs of a decision making organizations is very similar to Hurwicz's [30] suggestion of a resource cost to allocation mechanisms.

[3] For example, Bartholdi and Orlin [7], Bartholdi et al. [8, 9] address matters of computational complexity and the complexity of strategic manipulation in voting schemes. Johnson [31] references the relationship between the distribution of power and notions of choice complexity in Arrow type choice procedures.

imposed on the choices made by the individual or institution being modeled.[4] In particular both Arrow [2] and Plott [52] envisioned choice as an algorithm or process in which the elements of the feasible set were examined and a choice set determined.[5] Plott specifically suggested that there was a "computational efficiency" aspect to consistency axioms in general and to his path independence axiom in particular. Especially relevant to the results presented here, Plott provided a link between computing machines and path independent choice functions by proving that path independent choice functions form a semigroup under a naturally defined operation. The significance of this result is that, every semigroup can be used to define a semiautomaton (the basic building block of computing machines) and every semiautomaton has an associated semigroup.[6] More recently, Johnson [32, 34] has suggested that the link between the computational efficiency of a choice function and the consistency axiom it satisfies is captured by the algebraic structure of a subsemigroup of the semigroup originally identified by Plott.

Because the consistency axioms impose limitations on the relationship between the choice made on one set and that made on other sets, Plott's suggestion is intuitively appealing. Especially in the case of the most commonly used consistency requirements (i.e., rationalizability by either linear orders or weak orders and quasi transitivity for his path independence axiom), Plott observed that consistency axioms removed the requirement of re-examining previously considered alternatives. For the most commonly used axioms, it is the case, also, that the classes of choice functions satisfying these axioms form a nested set. Thus, the expectation is that, if consistency axioms are related to information processing efficiencies, then the choice function complexity should grow as the consistency axioms are relaxed. This intuition provides the same complexity ranking as the ordering by algebraic structure suggested by Johnson [31].

In these early discussions there, often, was an ambiguity in the use of the term "complexity" with imprecision in distinguishing between what information

[4]Virtually all economic models of individual choice assume that the individual satisfies the Weak Axiom of Revealed Preference. Often, as in the case of theorems following out of Arrow's general Possibilities Theorem [3], group decision structures are assumed to satisfy some type of consistency axiom. Many of these structures are surveyed in Kelly [42].

[5]Several other authors have formalized the search/process aspects of choice, which Arrow and Plott left somewhat unspecified, as an algorithm and investigated the relationship between these algorithms and the act of choice. Of particular note are Campbell's [13, 14] work on the existence of desirable algorithms for computing choice functions, Kelly's [43] look at the computability of collective choice rules and Bandyopadhyay's [5] characterization of a class of choice functions by means of a specific algorithm. Also contributing to this line of research are Lewis's [49, 50] investigations into the ability of a Turing machine (see Turing [59]) to compute economic choice automata on non-finite sets.

[6]More precisely, the link is between transformation semigroups and semiautomata. The transformation semigroup representation of a semiautomaton is discussed in Sect. 2.3. The relationship between transformation semigroups and semiautomata is discussed in Holcombe [28, pp. 31–34]. Holcombe [28, p. 145] also has a discussion of the relationship between semiautomata and automata.

scientists call "implementation complexity" and "computational complexity". The difference between these two complexities is that implementation complexity is determined by the difficulty of making an already determined choice. That is, given a choice function defined on a finite set, how much "computing power" is required by a semiautomaton to make choices consistent with the already specified choice function. Johnson [31] presents an intuitive example of this aspect of choice "difficulty". There, the difficulty of implementing a choice function is scaled by the number of entries in the incidence matrix representing the binary relation that rationalizes the choice function that are required to be known in order to assure the correct choice is made as the feasible set is expanded. There, this difficulty was called the "bit cost" of making choice. This implementation complexity is contrasted with "computational complexity" which reflects the difficulty of making the choice function. Until recently this "computational complexity" has arisen most explicitly in game theory.

In game theoretic discussions, the complexity of computing the best response (e.g., [10, 24, 45, 51]) has been substantially discussed. In these papers, the computational complexity is treated in the classical manner; usually some tool such as a time or memory space bound on determining a solution. Typically each of these is a computational complexity measure satisfying what are known as the Blum axioms [12].[7] One attraction of a computational complexity measure satisfying the Blum axioms is that the resulting scaling is independent of the machine performing the calculations. Similarly, the computational complexity measure introduced in Sect. 4 also satisfies the Blum axioms.

In contrast, implementation complexity has been interpreted in a number of disparate manners. Most commonly the scaling of implementation complexity is by the number of states in the semiautomaton implementing the desired strategy. The number-of-states measure of implementation complexity has some appeal in that it is numerical, simple to use, and as a result, it is easy to obtain results. However, even Abreu and Rubinstein [1] note limitations to their use of the number-of-states measure;

> Various features of the model presented below, such as the complexity measure we use, are rather special. Our results are therefore regarded as suggestive, and we strongly emphasize the exploratory nature of the present paper.

Kalai and Stanford [41] amend the basic number-of-states measure to reflect the fact that a semiautomaton can have extraneous states and appeal to the minimum number-of-states machine implementing a strategy. Banks and Sundaram [6] tried to capture the fact that there is more to machine complexity than the number of states in the machine by using a criterion that depends on both the number of states as well as the number of edges in the directed graph representation of the machine. Johnson [33] pointed out that, on machines with two states and two transitions,

[7] The reader is referred directly to Blum's classic paper for a discussion of the issues and techniques. The paper is short and readable.

there are, up to isomorphism, four distinct semiautomata and that these machines are categorized as (1) a machine that can't count, and can't detect differences in order of play, (2) a machine that can count but can't detect differences in the order of play, (3) a machine that can't count but can detect differences in the order of play, and, finally (4) a machine that can count *and* detect differences in the order of play. Johnson obtained these observations by using algebraic techniques similar to those employed in the results presented here.

Within algebraic complexity, the Krohn-Rhodes [47, 48] measure is commonly used.[8] This measure is based on the minimum number of simple groups in a wreath product that forms a semigroup covering of the semigroup generated by the machine of interest. Gottinger [26] in particular has suggested using the Krohn-Rhodes measure for some economic applications. In applications, however, there is a general problem in that it is not known how to determine what is the minimum number of simple groups in the decomposition of a particular semigroup. For specific application to choice functions, the measure is not very useful because there are no groups in the semigroup associated with the machines implementing path independent choice functions and, as a result, the Krohn-Rhodes measure would give an implementation complexity of zero to all path independent choice function implementing semiautomata. In the following, the power required to implement a choice function is ordered by the algebraic class of the machine implementing the choice function. Because these algebras are nested, comparison of the power of the different systems is straightforward.

From an economic perspective, one interpretation of these different complexities is that the computational complexity can be viewed as a fixed cost. This is the cost of determining or constructing the choice machine that will be used to implement choices. The computational complexity cost is born only once when the machine is made. This interpretation is consistent with classical views on computational complexity and with the way computational complexity is used in game theory. In contrast, the implementation complexity is more like a marginal cost. In order to implement a particular choice function, you need to maintain a machine with the requisite power to implement the choice function. This intuition is similar to the justification Rubinstein [54] gave in support of the number of state based measure of implementation complexity he used in his introduction of complexity costs into repeated play games.[9]

The first results presented here identify the structure of semiautomata constrained to satisfy Path Independence. The key requirement is to demonstrate that a particular semigroup is a subsemigroup of the semigroup of partial functions defined on the feasible sets. The particular subsemigroup also is a the subsemigroup of the semigroup identified by Plott in his first demonstration of the link between path independent choice functions and semigroups. In the past, Johnson [34] has

[8] See Krohn-Rhodes [47, 48] directly or Eilenberg [20, 21] for a more modern treatment.

[9] While not investigated here, this intuition naturally raises the possibility of trade offs between fixed costs and marginal costs in the design of choice machines/institutions.

demonstrated that choice semiautomata can be constructed by means of Eilenberg's embedding theorem. Demonstrating that the required semigroup already is a subsemigroup of the semigroup of partial functions, both simplifies the presentation and tightens the link between path independent choice functions and automata. Given this, it can be seen that choice semiautomata have elementary structure. For example, no "memory" is required to implement path independent choice.

Subsequent results identify a complexity ordering of path independent choice functions defined on finite sets. The classes of choice functions considered are those satisfying the Strong Axiom of Preference (SAP), the Weak Axiom of Revealed Preference (WARP), Quasitransitive rational Path Independence and (not necessarily rational) Path Independence. Each of the classes of choice functions are ordered by the mathematical power of the system implied by the defining consistency axiom. These comparisons confirm Plott's conjecture that consistency axioms contain information processing implications as well as conforming to Johnson's [31] ranking of choice function complexity by algebraic structure.

In order to focus the presentation a number of simplifications are made. For example, it is assumed that the choice functions are complete. This simplification will imply that the resulting automata, also are complete. In fact, the semiautomata model is perfectly capable of, and, in many ways, ideally suited for dealing with situations where completeness is not present but the exposition of the choice function implementing automata is simpler in the complete case. In addition, in the complete case, a natural nesting arises that eases discussion of implementation complexity rankings. If completeness is not assumed, then, depending on the precise nature of the incompleteness, the nesting of systems seen in this presentation may or may not be visible.[10] Another example of a simplification is that, while it is necessary to implement choice both as the feasible set *expands* as well as when it *contracts*, this presentation details only those machines that implement choice as the feasible set expands.[11]

Following specification of choice semiautomata and identifying the implementation complexity for each class, the matter of "computational complexity" is addressed. In this treatment a very simple approach is adopted. First, a result is presented that allows construction of all path independent choice functions on a finite set by means of a series of contractions.[12] This result provides the intuition for a partial order of the computational complexity of different choice functions.

[10] As a particular example, one of the referees inquired about restricting choice to sets that have null intersection. For the Path Independent choice functions considered, the mathematical structure that arises is a particular class of lattice. In a finite lattice, the meet and join of any two elements in the lattice must be well defined. If a choice operation from sets that have a non-empty intersection were not allowed, the lattice structure exploited here may not obtain.

[11] Choice functions that implement choice as the feasible set both expands and contracts are addressed in Johnson [34].

[12] I thank my co-author Richard A. Dean for permission to use our previously unpublished, original proof of this result. The appeal of this proof is that it is based on traditional lattice theoretic tools and, as a result, some may find it more accessible that other proofs.

Loosely, the scaling for the computational complexity is related to the number of elements removed from a particular feasible set in determining its choice set. This interpretation arises because each contraction deletes a single alternative as a possible choice element from a particular feasible set. It is seen that it is possible to order the classes of choice functions by their computational complexity. And, as noted earlier, the method satisfies the Blum axioms.

Most intriguingly, in a very natural manner, it turns out that the implementation complexity and the computational complexity are dual in the sense that classes of choice functions that have higher implementation complexity have lower computational complexity. Because of the number and range of different choice functions in each class, there is some overlap in the computational complexities of specific choice functions but, for the most computationally intensive and least computationally intensive representatives of each class, the duality holds.[13]

Section 2 presents the basic definitions and tools. The semiautomaton model and the transition to path independent choice function implementing machines are summarized in Sect. 3. A principle focus is in demonstrating that the mappings for the choice semiautomaton form a subsemigroup of the semigroup of partial functions among the possible subsets. This section also reports on the results for implementation complexity. Section 4 presents the results on computational complexity. Conclusions are in Sect. 5 and the original proof of the contraction theorem is provided in the Appendix.

2 Definitions and Notation

The technical tools used in the results presented below are choice functions, the elementary algebra of sets, primarily semigroups and lattices, and the transformation semigroup representation of a semiautomaton. The definitions and prerequisites of choice functions and consistency requirements on choice functions are presented in Sect. 2.1. Algebras are covered in Sect. 2.2. Semiautomata and the links to algebra are covered in Sect. 2.3.

2.1 Notation, Choice Functions and Consistency Requirements

The universal set V is composed of a finite number of distinct alternatives, and 2^V is the power set of V. Subsets of V, denoted by v, are elements of 2^V. Unless otherwise stated, the cardinality of V, denoted by $|V|$, is t, and the cardinality of $v \in 2^V$ is n; note $n \leq t$. Distinct subsets of V are subscripted with an integer $i \in \{1, \ldots 2^t\}$; where $\{v_i\} = \{v_j\}$ if and only if $i = j$.

[13] Previous results existed only for the least computationally intensive representative of each class.

A *choice function* is a mapping $C : 2^V \to 2^V$, such that $C(v) \subseteq v$ and $C(v) = \emptyset$ if and only if $v = \emptyset$. A choice function C defined on V is *discriminating* if there is some $v \in 2^V$ for which $C(v) \neq v$. A choice function C is *rational* if and only if there exists a relation R such that, for every $v \in 2^V$, $C(v) = G(v; R)$ where, $G(v; R) = \{x \in v | xRy, \forall y \in v\}$. The function $G(v; R)$ selects the R-maximal elements.

The classes of choice functions considered are those satisfying the Strong Axiom of Preference, the Weak axiom of Revealed Preference, the conjunction of Path Independence and Extension, and Path Independence (alone). The consistency axioms are defined formally as follows.

Strong Axiom of Preference (SAP):

(i) $\forall x, y \in V, x \in C(\{x, y\}) \Rightarrow y \notin C(\{x, y\})$, and

(ii) $\forall v_1, v_2 \subseteq V, v_1 \subseteq v_2 \Rightarrow \{v_1 \cap C(v_2)\} = \begin{cases} \emptyset, \text{or} \\ C(v_1) \end{cases}$.

Weak Axiom of Revealed Preference (WARP)

$\forall v_1, v_2 \subseteq V, v_1 \subseteq v_2 \Rightarrow \{v_1 \cap C(v_2)\} = \begin{cases} \emptyset, \text{or} \\ C(v_1) \end{cases}$.

Rational Path Independence (RPI)

(i) $\forall v_1, v_2 \subseteq V, C(C(v_1) \cup C(v_2)) = C(v_1 \cup v_2)$, and

(ii) Extension (E): $\forall v \subseteq V, (x \in v \text{ and } (\forall y_{y \in v}, x \in C(\{x, y\})) \Rightarrow x \in C(v))$.

Path Independence (PI)

$\forall v_1, v_2 \subseteq V, C(C(v_1) \cup C(v_2)) = C(v_1 \cup v_2)$.

The first two of these classes always can be rationalized by a complete, reflexive and transitive binary relation [2]. Choice functions satisfying the Strong Axiom are always single-valued and rationalized by linear orders while choice functions meeting WARP need not be single-valued and are rationalized by weak orders. The two classes of path independent choice functions are distinguished by whether or not they are rationalizable; choice functions satisfying both PI and E are rationalizable by a quasitransitive relation while choice functions satisfying PI need not be rationalizable [52].[14]

2.2 Algebras

The definitions of binary systems and system properties are provided in terms of an arbitrary finite non-empty set N, which is used as both the domain and the range,

[14] A complete, reflexive relation R, where the strict preference part is denoted by P, is *quasitransitive* if for all $x, y, z \in V, xPy$ and $yPz \to xPz$. Thus strict preference is transitive while indifference need not be.

and a binary operation denoted by $(\cdot)$. Thus, $\cdot : N \times N \rightarrow N$, and the binary system for N under the operation $(\cdot)$ is $\langle N; \cdot \rangle$. Algebraic properties defined for all $v_i, v_j, v_k \in N$ are

(B-1) Closure: $v_i \cdot v_j \in N$,
(B-2) Associativity: $v_i \cdot (v_j \cdot v_k) = (v_i \cdot v_j) \cdot v_k$,
(B-3) Commutativity: $v_i \cdot v_j = v_j \cdot v_i$, and
(B-4) Idempotence: $v_i \cdot v_i = v_i$.

A binary system satisfying (B-1) and (B-2) is called a *semigroup*, and a semigroup satisfying (B-3) is called a *commutative semigroup*. A semigroup for which every element satisfies (B-4) is called an idempotent semigroup. A commutative idempotent semigroup is a *semilattice*. Conceptually, some may find it easier to consider these semilattices diagrammatically under the natural partial ordering of the semigroup where the *natural partial ordering* is defined as follows: $a \cdot b = b \Leftrightarrow a \leq b$.[15]

For application to choice functions, the power set of the universal set V is used as both the domain and the range, and the binary operation $(\bullet)$ is adopted from Plott [52].

Definition 1 Given a path independent choice function C, the *Plott Product* $(\bullet)$ is defined as follows, $\bullet : 2^V \times 2^V \rightarrow 2^V$, where $\forall v_1, v_2 \in 2^V, v_1 \bullet v_2 = C(C(v_1) \cup C(v_2))$.

Formally, the operation $(\bullet)$ should be subscripted by the choice function used in its definition, however, to avoid excess notation, this subscripting is omitted where the choice function can be inferred from the context. The binary system for V under the operation $(\bullet)$ is denoted by $\langle 2^V; \bullet \rangle$. Plott [52] proved that this system is a commutative semigroup.

In addition to the properties of the operation, it is useful to identify two special members of binary systems.

Definition 2 Given a binary system $T = \langle N; \cdot \rangle$ an element z such that $x \cdot z = z \cdot x = z, \forall x \in T$ is called a *zero*, and an element e such that $t \cdot e = e \cdot t = t, \forall t \in T$ is called an *identity*.

A semigroup that has an identity is a *monoid*. An idempotent commutative monoid with a zero is a *lattice*. Johnson [32] identified a subsemigroup of Plott's semigroup that has precisely these properties. Further, Johnson conjectured that this subsemigroup might be relevant to economic applications of automata theory. The results below validate that conjecture. While initially identified by means of Plott's single operation $(\bullet)$, lattices actually have two operations, typically called the *join* denoted by $\vee$ and the *meet* denoted by $\wedge$. A lattice L is denoted by $\langle L; \vee, \wedge \rangle$. A lattice L has a *dual* denoted by D obtained by interchanging the roles of the meet and join operations so that if $a \vee b = a$ in L then $a \vee b = b$ in D. Given a

[15] The natural partial ordering is adopted from Clifford and Preston [16, 17].

lattice $L = \langle L; \wedge, \vee \rangle$ for which there is a set K such that $\emptyset \neq K \subseteq L$ and $a, b \in K$ implies $a \wedge b \in K$ and $a \vee b \in K$ then $K = \langle K; \wedge, \vee \rangle$ is a *sublattice* of L. An element x in a lattice is called *join-irreducible* if $a \vee b = x$ implies $x = a$ or $x = b$. By convention, bottom elements of a lattice are not called join-irreducible. Dually, an element y in a lattice is called *meet-irreducible* if $a \wedge b = y$ implies $y = a$ or $y = b$. For both the join irreducibles and the meet irreducibles, the partially ordered set of irreducibles $\langle P; \leqslant \rangle$ may be important. In a partially ordered set P, x *covers* y if $x > y$ and for no $a \in P$, $x > a > y$. Lattices are well covered in such classics as Birkhoff [11] and Davey and Priestly [18]; however, a few especially useful properties are summarized here. One important property of some lattices is the distributive law. A lattice $\langle L; \vee, \wedge \rangle$ is a *distributive lattice* if it satisfies the *distributive law*:

$$a \vee (b \wedge c) = (a \vee b) \wedge (a \vee c) \text{ for all } (a, b, c \in L).$$

Given a lattice L with a zero 0 and an identity 1, and some element $a \in L$, for which there is an element $b \in L$ such that $a \wedge b = 0$ and $a \vee b = 1$, then a is said to have a *compliment*. If a has a unique compliment, then the compliment is denoted by a'. Taking the compliment is a *unary* operation. A *Boolean algebra* is a system $\langle B; \wedge, \vee, ', 0, 1 \rangle$ such that (1) $\langle B, \wedge, \vee \rangle$ is a distributive lattice, (2) $a \wedge 1 = a$ and $a \vee 0 = a$ for all $a \in B$, and (3) $a \wedge a' = 0$ and $a \vee a' = 1$ for all $a \in B$. The finite Boolean algebras considered here are isomorphic to 2^V under set union, intersection and complementation.

Within the Boolean algebra 2^V for sets $V \supseteq T \supseteq B$, the collection of sets K such that $T \supseteq K \supseteq B$ is called an *interval*, and the interval is denoted by T/B. T is the *top* of the interval, and B is the *bottom* of the interval. An interval T/B is called *proper* if $T \neq B$. If $T = B \cup \{x\}$, then T *covers* B and the interval T/B is a *prime* interval.[16] It will turn out that intervals are significant in path independent choice functions. In fact, little else is required in addition to the interval property in order to define a path independent choice function.[17] Two examples of the presence of intervals in the domain of the choice function being mapped into a particular choice element are presented in Fig. 1.

A particular class of lattices initially identified by Dilworth [19] and now known as *lower locally distributive lattices* (LLDs) is relevant for choice functions. A lattice is an LLD if every element in the lattice has a unique irredundant representation as the join of join irreducibles. Here a representation of an element a in a lattice as the *irredundant* join of join irreducibles means that if $a = x_1 \vee x_2 \vee \ldots \vee x_k$ then a is not the join of any proper subset of $\{x_1, \ldots, x_k\}$. This representation also is *unique* in the sense that if $a = y_1 \vee y_2 \vee \ldots \vee y_h$ as well as

[16] Birkhoff attributes this use of the term prime interval to Morgan Ward.

[17] See Johnson and Dean [39] for the characterization of path independent choice functions result using partitions of the domain satisfying the interval property and little else.

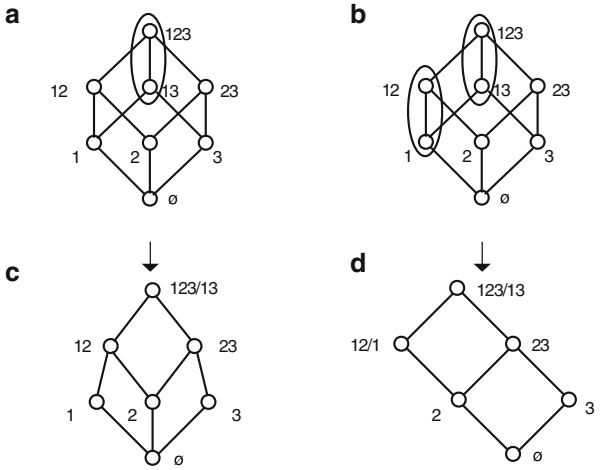

Fig. 1 Demonstration of relationship between identified intervals and lattice representations of Path Independent choice functions; (**a**) Boolean algebra with one interval identified, (**b**) Boolean algebra with two intervals identified, (**c**) image lattice of (**a**) with interval 123/13 identified, and (**d**) image lattice of (**b**) with intervals 123/13 and 12/1 identified

$a = x_1 \vee x_2 \vee \ldots \vee x_k$ then $h = k$ and $\{x_1, \ldots, x_k\} = \{y_1, \ldots, y_h\}$.[18] Johnson and Dean [35, 36] and, independently, Koshevoy [46] demonstrated a direct link between path independent choice functions and LLDs in that every PI choice function has a representation as an LLD lattice and for every LLD lattice, there is an associated PI choice function.[19] Further, Johnson and Dean demonstrated characterization results between the predominant classes of PI choice functions and subclasses of LLDs. Significantly, not all of these LLDs are distributive.

2.3 *Semiautomata, Transformation Semigroups and Action*

Although employed here only as a link to other literature, a common means for representing a semiautomaton, or finite state machine, in economics is through the directed graph. A directed graph representation of a semiautomaton $\mathscr{M} = (Q, \Sigma, F)$ consists of a finite number of states Q, an alphabet Σ and partial functions F. The partial functions F are the transitions so that $F : Q \times \Sigma \to Q$. If the partial functions F are functions then the semiautomaton is a *complete*

[18]The partition lattice on five elements is an example of a lattice that fails to meet this requirement.

[19]Koshevoy [46] used convex geometries to obtain results related to a subset of the Johnson and Dean [35, 36] results. Here the full range of the Johnson and Dean characterizations is used.

semiautomaton. In the directed graph, the states become the vertices, the partial functions F are the edges, and the alphabet labels the edges.[20]

Within information science and mathematics, a standard alternative to the directed graph representation is the transformation semigroup. This fundamental semigroup of algebraic automata theory consists of an underlying set and an action semigroup (see Eilenberg [20, 21] or Holcombe [28]). The transformation semigroup and its component parts are defined as follows.[21]

Let Q be a finite set, and let $PF(Q)$ be the monoid of partial functions $Q \to Q$ with composition of partial functions as multiplication. The identity partial function is the unit denoted by 1_Q. (In this paper, the situation is simplified because the mappings considered are functions.) A *transformation semigroup* $X = (Q, S)$ is a finite set Q and S is a subsemigroup of $PF(Q)$. The set Q is called the *underlying set* of X, and the members of Q are called *states*. The semigroup S is called the *action semigroup* of X, and the elements of S are called the *transformations* of X.

The transformation semigroup X is *complete* if the following two conditions are met.

(a) $Q \neq \emptyset$
(b) $qs \neq \emptyset$ for all $q \in Q, s \in S$.

Condition (a) assures that the underlying set is not empty and condition (b) requires that each transformation be defined at every state. Thus, as with the directed graph representation, the transformation semigroup representation is complete if the transformations of X are functions.

Both the transformation semigroup and the action semigroup are important items in the study of automata theory. However, while the transformation semigroups characterize semiautomata, all of the mathematical power or algebraic complexity is contained in the action semigroup [21]. For this reason, much of the remaining analysis is focused on the action semigroup.

3 From Generic Semiautomata to Choice Semiautomata

As described above, semiautomata can be represented either as a directed graph or as a transformation semigroup. Two small examples of the directed graph representations are depicted in Fig. 2. In the left most example (Fig. 2a), there are three states and two letters in the alphabet labeling transitions among the states. The transitions labeled "a" form a cycle among the three states while the transitions labeled "b" flip the triangle about the state 1. The algebra associated with the

[20] For perspective, the more commonly employed automaton is a semiautomaton that has been augmented by identification of an *initial* state i and a collection of *terminal* states T. Thus an automaton $\mathscr{A} = (\mathscr{M}, i, T)$.

[21] This summary borrows from Eilenberg [20, 21].

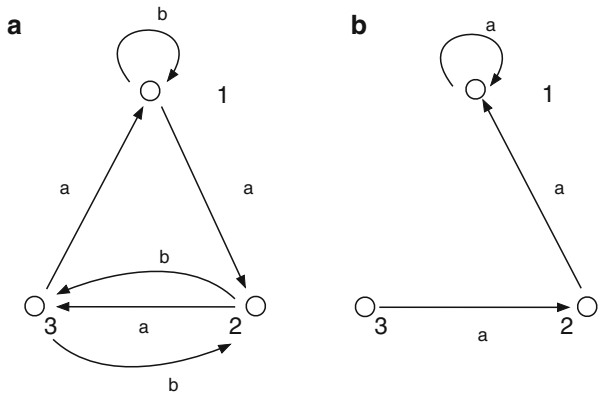

Fig. 2 Example of two three-state semiautomata, (**a**) on the *left* with two-letter alphabet labeling transitions, and (**b**) on the *right* with a single-letter alphabet

semiautomaton is the dihedral group. This system is one of the most "powerful" systems on three states with two transitions. To information scientists, what makes this example "powerful" is that it is a group.[22] The group structure endows this system with the ability to count repeatedly. Moreover, this particular group is not commutative, thus, the machine associated with it has the ability to detect differences in the sequence of action. Standard algebraic complexity holds that any system without a group structure (one or more groups in the algebra associated with the machine) has *zero* algebraic complexity (also called implementation complexity). Significantly, none of the structures implementing path independent choice posses a group structure. All of the "choice semiautomata" considered rely only on the lattices we will see as we progress. To make this semiautomaton an automaton what needs to be done is to identify one of the states as the "initial" state and some subset of the states as potential "terminal states". On the right side of Fig. 2, (Fig. 2b) is another three-state semiautomaton with a single letter alphabet. In this case the algebra associated with this machine is a chain. This semiautomaton is one of the "simplest" on three states. It is simple, in part, because, lacking a group structure, it can count only once and, being a commutative system, it does not have the ability to detect differences in sequence. An automaton is achieved once again by specifying an initial state and possible terminal states.

There are several ways to determine the algebra associated with a particular directed graph representation of a semiautomaton. One useful technique is demonstrated here using the Fig. 2a directed graph representation as a start. This method works by representing the transformations by transformation matrices and then working out the operation table for the semigroup obtained as the multiplicative

[22] A *group* is a semigroup in which there is an identity e and for which every element a in the semigroup has an *inverse* a^{-1} such that $a \cdot a^{-1} = a^{-1} \cdot a = e$.

closure of all possible products of the transformation matrices. For example, the transformation a takes state 1 into state 2 and state 2 into state 3 and state 3 into state 1. From there the cycle repeats. If we represent the initial state as the row and the destination state as the column, this transformation is represented by the following matrix,

$$a = \begin{bmatrix} 0\,1\,0 \\ 0\,0\,1 \\ 1\,0\,0 \end{bmatrix}.$$

Similarly, the transformation b is represented by this matrix,

$$b = \begin{bmatrix} 1\,0\,0 \\ 0\,0\,1 \\ 0\,1\,0 \end{bmatrix}.$$

The letters a and b are the letters of the alphabet for this Fig. 2a semiautomaton. Taking the multiplicative closure of these two matrices generates the four words of this machine. These words are presented below.

$$a^2 = \begin{bmatrix} 0\,0\,1 \\ 1\,0\,0 \\ 0\,1\,0 \end{bmatrix}, a^3 = b^2 = I = \begin{bmatrix} 1\,0\,0 \\ 0\,1\,0 \\ 0\,0\,1 \end{bmatrix},$$

$$ba = \begin{bmatrix} 0\,1\,0 \\ 1\,0\,0 \\ 0\,0\,1 \end{bmatrix}, ab = \begin{bmatrix} 0\,0\,1 \\ 0\,1\,0 \\ 1\,0\,0 \end{bmatrix}$$

This information is compactly summarized by the operation table presented in Fig. 3.

To complete our description of this machine as a transformation semigroup, note that the underlying set $Q_3 = \{1, 2, 3\}$ while the action semigroup is the dihedral group on three elements S_{D3} with the operation table as depicted in Fig. 3. Thus, the transformation semigroup $X_D = (Q_3, S_{D3})$. Evident in the operation table is the

Fig. 3 Operation table for S_{D3} dihedral group associated with the directed graph in Fig. 2a

	I	a	a^2	b	ab	ba
I	I	a	a^2	b	ab	ba
a	a	a^2	I	ab	ba	b
a^2	a^2	I	a	ba	b	ab
b	b	ba	ab	I	a^2	a
ab	ab	b	ba	a	I	a^2
ba	ba	ab	b	a^2	a	I

Fig. 4 Operation table for S_{C3} the chain associated with the directed graph of Fig. 2b

	a	a^2
a	a^2	a^2
a^2	a^2	a^2

cyclic counting from the interaction of the "a" mapping while interaction between a and b or between a and products of a and b provide the sequence detecting ability. For example, the sequence ba followed by ab leads to a different result from the sequence ab followed by the sequence ba.

Another technique for constructing a machine algebra is exposited using the Fig. 2b semiautomaton.[23] The semiautomaton depicted in Fig. 2b is less rich. By examination, it can be seen that the single mapping a has the property that once it is applied to any state twice ($aa = a^2$) the result is that, independent of where the semiautomaton is started, it will be in state 1. This situation is represented formally in the state transition table below. On the top is a listing of the states; 1, 2,and 3. Subsequent rows specify what happens when the transition a is applied once (first row) and twice (second row).

	1 2 3
a	1 1 2
a^2	1 1 1

The corresponding operation table is presented in Fig. 4. Notably, because this semiautomaton does not have an identity mapping and, as a consequence, neither does it's algebra the operation table is not especially informative. These mappings do endow the algebraic system with a zero (a^2) and this is evident in the operation table. And, as before, we can identify the transformation semigroup as follows. The underlying set is the same $Q_3 = \{1, 2, 3\}$ but the action semigroup is different S_{C3} with the transitions as described in the state transition table and operation table as in Fig. 4. Given this, we see the transformation semigroup is $X_C = (Q_3, S_{C3})$.

Of course, these are just two of the many semiautomata that can be defined on three states and having one or two transitions. In fact, these two are both subsemigroups of the semigroup of partial functions on these three states. There are many more subsemigroups including the null machine and the identity machine and all other possibilities of partial functions mapping a state to itself or some other state or to no state at all (often called the null map). Each of these semiautomata has an associated semigroup and every semigroup has at least one semiautomaton that is associated with it. The number of semigroups varies on the precise class meant (e.g., mononids, commutative, only non-isomorphic semigroups, etc.). One nice result provided by Kleitman et al. [44] offers an asymptotic approximation for

[23] Yet another technique using permutation notation for representing the partial functions can be seen in Howie [29].

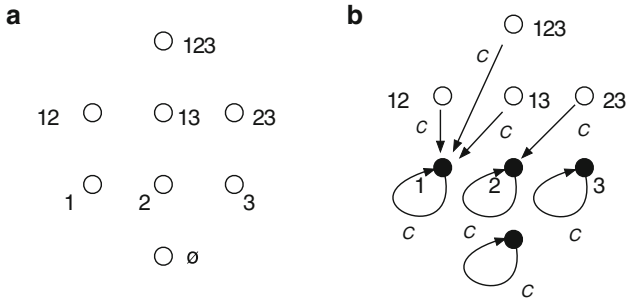

Fig. 5 Start at defining a "choice semiautomaton". (**a**) Depicts the eight states labeled as elements of the Boolean algebra on three alternatives. (**b**) Depicts the same eight states and one particular set of transformations C

the number of semigroups $S(n)$ on n elements as $S(n) = \left[\sum_{i=1}^{n} f(t)\right](1 + o(1))$ where $f(t) = \binom{n}{t} t^{1+(n-t)^2}$. Other estimates under different assumptions range from the very early Forsythe [22] to the more recent Grillet [27].

Of all the possible semiautomata on n states, we are interested only in those that implement Path Independent choice. From Johnson [31] we know that not all semiautomata meet the requirements for Path Independent choice. In Fig. 5a, b we see both the initial layout of the eight states representing the possible choice sets on three alternatives. Of course, three alternatives means there will be *eight* different sets (including the empty set from which the choice only can be the empty set) from which we might have to choose and thus, eight states in the semiautomaton.[24] Each of the states is labeled as a subset of a feasible set $\{1, 2, 3\}$. In Fig. 5a there are no transitions while in Fig. 5b there are a number of transitions. The intent is to build a semiautomaton that implements the choice function rationalized by the linear order where 1 is preferred to 2 and both 1 and 2 are preferred to 3. Clearly visible is the interval property of Path Independent choice functions; for example, every state between 1 and 123 in the Boolean algebra on $\{1, 2, 3, \}$ is mapped into 1.[25] What the interval property reflects is an equivalence class of the input signals to the semiautomaton. Specifically, because any of the input signals (here, the signal are new sets of available alternatives) $\{1, 2, 3\}$, $\{1, 2\}$, $\{1, 3\}$, and $\{1\}$, has the same effect as the choice element from each of these sets, viz. $\{1\}$. After a little further development, this choice function is used as a example in specifying the choice semiautomaton.

[24] Satoh et al. [55] calculate the number of non-equivalent semigroups of order 8 at around 1.85 billion.

[25] As in Fig. 1, the soft brackets are omitted to simplify notation.

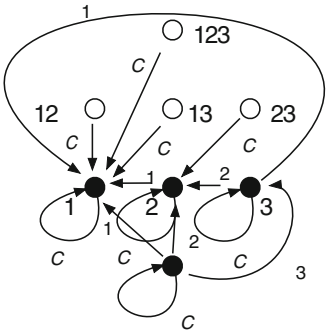

Fig. 6 Figure 5 mappings under C with additional mappings added as required for meeting the implications of Plott's product (•)

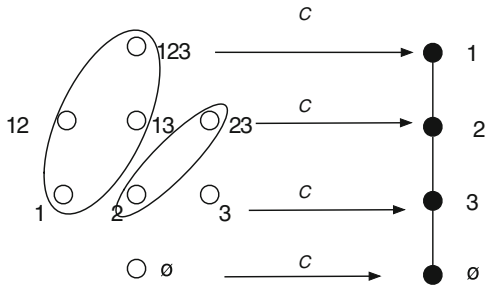

Fig. 7 On the *left* is the Boolean algebra on three alternatives and on the *right* is the image of the mapping C combined with the information of Plott's "•" to produce the chain I_{C3} on the *right*

Now, there are a number of possible labelings for the partial functions in Fig. 5 but one useful way of labeling is to use the notation C, for *choice* for each of the transitions. In Fig. 6, the additional requirements imposed by the ranking of 1 over 2 and both of these over 3 is incorporated. At this point, the diagram is getting fairly confusing. A, perhaps, more easily read representation is presented in Fig. 7. On the left in Fig. 7a is the same diagram as in Fig. 5b with the labeling and in Fig. 7b the presentation I find most useful. In the presentation, the underlying set Q is on the left and the subset of the underlying set to which all states are mapped is on the right. Here, these elements are ordered in the manner required by the Plott operation (•) while the elements $\{1, 2, 3, \emptyset\}$ in Fig. 5 use only the information of the mapping C. The ordering in Fig. 7 is a chain which is labeled I_{C3} for future use.

This representation is very useful in understanding the operation of a "choice semiautomaton". In particular, the C mapping incorporates the interval property present in all path independent choice functions and the operation (•) helps provide the ordering. In fact, for all path independent choice functions it is useful to recognize that both the mappings C and the impact of the interactions resulting from aggregating choice sets leads to a representation as a lattice.

The following remarks summarize salient points of this discussion.

Remark 1 Note that the mappings in Fig. 6 are a subset of all possible partial functions among the elements of the underlying set. While demonstrated only on this small example, the observation extends to all finite sets.

Remark 2 As a result of Remark 1, the semigroup obtained from the transitions will be a subsemigroup of the semigroup of all partial functions on the underlying set. Again, while this result is demonstrated only on a small example, the result extends to all finite sets.

In addition, because the choice functions are *functions* this is a complete semiautomaton. Analytically, it is useful to think of choice functions and Plott's (•) operation in the Boolean algebra domain and the meet and join operations in the lattice domain. And, conveniently, the lattices that form the range of the mapping have been characterized.[26] These results are summarized in the following proposition.

Proposition 1 *Let C be a discriminating choice function that satisfies PI on V, let* (•) *be the Plott product, and let* $T = (2^V; J)$ *be the complete transformation semigroup derived from C. Then*

1. *C satisfies PI if and only if J is an LLD lattice,*
2. *C satisfies PI and E if and only if J is a distributive lattice,*
3. *C satisfies WARP if and only if J is a chain of Boolean algebras, and*
4. *C satisfies SAP if and only if J is a chain.*

Examples of each of these systems on a three alternative domain are represented in Fig. 8. Figure 8a is the canonical seven-element LLD lattice that is a sublattice of every LLD lattice, Fig. 8b is a six-element distributive lattice, Fig. 8c is a chain

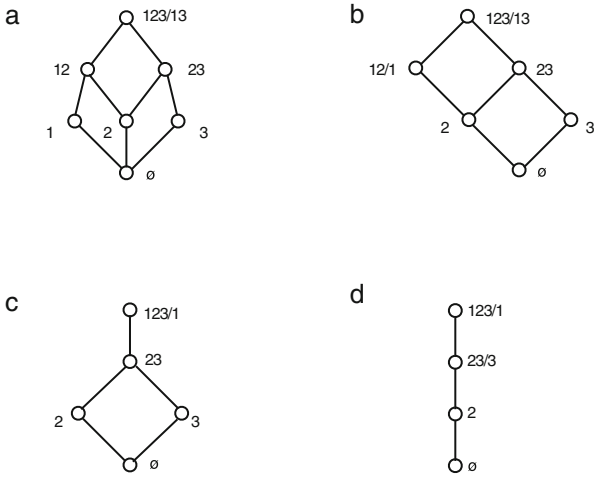

Fig. 8 (**a**) is an LLD lattice, (**b**) is a distributive lattice, (**c**) is chain of Boolean algebras and (**d**) is a chain

[26] See Johnson and Dean [35, 36] for the complete set of characterizations and Koshevoy [46] for a subset of these characterizations.

of Boolean algebras and Fig. 8d is a chain. For these lattices derived from choice functions defined on three alternatives, it turns out that the representatives of each class of lattice are differentiated by the number of elements in the image lattice. When the domain has four or more alternatives, this is no longer true.[27] On four or more alternatives, there is substantial overlap in the number of elements of the representatives of each class. Most important, independent of the number of alternatives in V, the class of distributive lattices is a strictly contained in the class of LLDS lattices and the class of chains of Boolean algebras is strictly contained in the class of distributive lattices, while the class of chains is strictly contained in the class of chains of Boolean algebras. Thus, the relevant algebras for the action semigroup are nested.

Notably the systems identified are nested so that comparison of the mathematical powers is direct. A chain is capable only of ordering a set; it does not have the ability to allow for indifference among alternatives. Similarly, a chain of Boolean algebras has the ability to permit indifference but cannot handle the case where strict preference is transitive but indifference need not be transitive. The distributive lattices have the power to allow for intransitive indifference but can not handle the case where the final choice can depend on the *sequence of expansions and contractions*. Finally, an LLD lattice has sufficient power to handle choice situations where there may not be an underlying relation rationalizing choice and where the final choice may depend of the sequence of expansions and contractions.[28] Given this nesting, it is seen that non-rationalizable path independent choice functions have the highest requirement for implementation while choice functions rationalized by linear orders have the lowest mathematical requirement.

Example Building on the choice function rationalized by the linear order of 1 preferred to 2 and 2 preferred to 3 the transformation semigroup X of choice semiautomaton can be specified. First, the underlying set, Q, is the Boolean algebra on $\{1, 2, 3\}$. As specified in Proposition 1, the action semigroup of the lattice of idempotents that, in this case, is a chain. For this choice function the idempotents are $I = \{\emptyset, 1, 2, 3\}$. Thus, the transformation semigroup is $X = (2^V, I)$. In this representation, the underlying set is not the "minimum number of states" that will implement the relevant choice. In the minimum number of state machine, only the "representative" states are required; these "representative" states are the same elements as the idempotents, I. Thus, the transformation semigroup of the minimum-number-of-state semiautomaton, $\bar{X}$, is $\bar{X} = (I, I)$. To see the operation of this choice function, consider the following choice problem

$$\{2, 3\} \bullet \{1, 3\} = ?$$

[27] See Johnson and Dean [37] for an atlas of unique LLDs on four alternatives.

[28] The key issue here is that LLD lattices are not distributive so that the final choice can depend on the interactions of the meet and join operations. Most critically, in the non-rational path independent choice functions, the choice from the intersection of two sets depends on the largest sets from which the relevant choice sets are drawn. See Johnson [34] for a concrete example of this "path dependence".

In this case, the representative state for $\{2, 3\} = 2$ (with the brackets left off the representative element). Similarly, for $\{1, 3\} = 1$ and in the lattice domain, the join operation yields $2 \vee 1 = 1$. $\diamond$

Similar examples can be constructed for the other choice functions presented in this section.

In the next section it will be seen that this ordering by implementation complexity is reversed when computational complexity is considered.

4 Computational Complexity of Path Independent Choice Functions

In general, the number of steps required to solve a problem of a particular type can depend on the nature of the computer applied to the problem. For this reason many approaches to computational complexity look for some more fundamental aspect of what is required to solve the problem. Frequently, the goal that is sought is some scaling that is independent of the particular machine or algorithm applied to the problem.[29] The measure that is used here fits within that framework. For path independent choice functions the effort that is expended to create a choice implementing semiautomaton must identify the collection of sets from which the same choice will be made. For path independent choice functions, an attractive computational complexity measure of a particular choice implementing semiautomaton is the number of prime intervals defining the collection of sets from which the same choice will be made. As noted earlier, each of these collections of sets will be an interval in the domain of the choice function being implemented.

In addition to being independent of a particular algorithm or machine, using the prime intervals has two attractive economic intuitions. First, identifying a prime interval selects two subsets of the feasible set from which the same choice will be made. Effectively, identifying a prime interval answers the question, "Do you want to make the same choice from these two sets?". This idea is firmly rooted in the view that consistency axioms are concerned with the relationship between choice made on one set and choices made on other sets. Second, the size of the two sets related by a prime interval differ by a single alternative with the superset being larger than the subset. Thus, identifying a prime interval also specifies some particular alternative that will not be in the choice from the larger of these sets.

The foundation for this approach is the following lemma from Johnson and Dean [36] which provides conditions under which a contraction of an interval will result in a new path independent choice function. Combined with a related result that assures

[29] As noted earlier computational complexity measures that have this property satisfy the Blum [12] axioms.

that the contractions are reversible, this lemma provides a means of constructing all path independent choice functions on a finite set.[30]

Lemma 1 *Let C be a path independent choice function on a set V. Let B be a meet irreducible element in the lattice of idempotents of C that is not equal to $C(V)$ or $\emptyset$. Let A be the unique element covering B in this lattice. Suppose that $A = B \cup \{x\}$. Let the function C^* defined as:*

$$C^*(S) = C(S) \text{ if } C(S) \neq A$$

$$C^*(B) = B \text{ if } C(S) = A$$

The function C^* is a path independent choice function on V. We say that C^* is obtained from C by contracting the quotient A/B in the lattice of idempotents under C.

Proof See Appendix.

While the measure of computational complexity is independent of any particular algorithm, it is useful to examine a concrete example of how the sequence of contractions described in the lemma above actually works. In particular, reviewing the operation of the contractions provides a clear understanding of what really happens to the prime intervals in the process of identifying a particular action semigroup.

Example Construction of the five discriminating choice lattices on three elements is diagramed in Fig. 9. Application of the contraction procedure is direct. In most cases, so is identifying the computational complexity. The exceptions are the sequences of contractions resulting in the two chains of Boolean algebras. Special note will be made of features of those contractions and how that relates to the computational complexity of the choice function. Consider PI choice functions defined on $V = \{1, 2, 3\}$. To help in the exposition of the process as well as to identify the prime intervals that will be the scale for the computational complexity Fig. 9 presents the domain of the choice function 2^3 on the left side, the five discriminating choice functions in the center and the intermediate lattices on the far right side. The Boolean algebras presented on the left are used to identify and keep track of the total number of prime intervals that are identified in the domain. The intermediate lattices on the right are used to identify the single prime intervals that are contracted in each application of Lemma 1.

The algorithm begins with the Boolean algebra on three elements presented at the top left of Fig. 9. The first discriminating lattice, labeled L7 at the center top of the diagram, is constructed by contracting one of the three intervals 123/12, 123/13, 123/23. Each of these contractions will result in a lattice isomorphic to the lattice L7. In L7, the interval 123/13 has been contracted. Note that *one* prime interval has

[30] See Johnson and Dean [36].

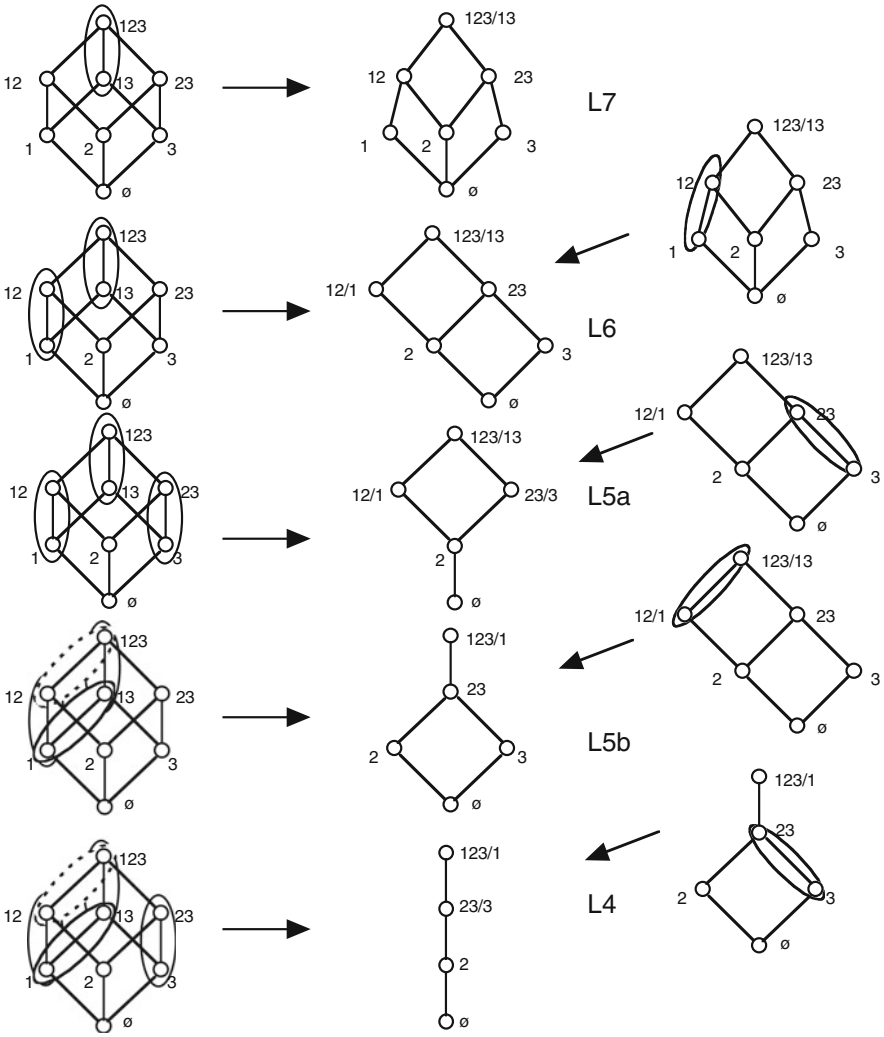

Fig. 9 Sequence of contractions leading to construction of all non-isomorphic LLD lattices that are possible action semigroups of semiautomata implementing Path Independent Choice Functions on a three alternative domain using the contraction process

been contracted and it has been determined that the alternative 2 will not be chosen from the set $\{1, 2, 3\}$.

The second lattice constructed, L6 is obtained by contracting one of two intervals in L7 meeting the conditions of Lemma 1. In this case the two intervals that could have been contracted: 12/1 or 23/3. Either contraction will result in a lattice that is isomorphic to the six element distributive lattice second from the top of the middle column labeled L6. In this case the interval 12/1 has been contracted as indicated

in the seven element lattice to the upper right of L6. It has been determined that the alternative 2 will not be selected from the set $\{1, 2\}$. As can be seen in the Boolean algebra to the left of L6, two prime intervals in the domain have been identified.

In L6, there are again two intervals that can be contracted. This time however, the contractions will not result in isomorphic lattices (in fact, they will be dual lattices). One of possible contractions, 23/3, is easy to see because it is just like the earlier contractions. Contracting this interval leads to the lattice L5a in the middle of the center column. This lattice has a two-element Boolean algebra on top of a singleton. As can be seen in the Boolean algebra to the left of L5a, in this lattice, *three* prime intervals have been contracted.

The other interval in L6 that can be contracted is an interval that involves two previously contracted intervals; this interval consists of the lattice point labeled 12/1 and the lattice point labeled 123/13 (see the copy of L6 to the upper left of L5b). The result of this contraction is the lattice with a two-element Boolean algebra on the bottom, labeled L5b. Looking to the left of this lattice, it can be seen that, although only three contraction operations have been made, a total of *four* prime intervals have been contracted. This is because the interval being contracted consisted of previously contracted intervals. Even though only three contraction operations have been made, the fourth prime interval has been contracted because of the implication of the two contractions made previously. The induced contraction is depicted by the dashed loop in the Boolean algebra to the left of L5b. Notice that this mapping is the first case of an interval that is not just a prime interval. Here, the interval consists of all the sets between $\{1\}$ and $\{1, 2, 3\}$ in the Boolean algebra for a total of four prime intervals. Equally important, the cumulative impact of the individual prime interval contractions is that alternative 1 is the only alternative chosen from any of the sets in the interval $1/123$. Observe that identifying the first prime interval determined that alternative 2 would not be chosen from the set $\{1, 2, 3\}$, identifying the second prime interval determined that alternative 2 would not be chosen from the set $\{1, 2\}$ and the final prime interval, determined that alternative 3 would not be chosen from the set $\{1, 3\}$ with the implication that only alternative 1 will be chosen from the set $\{1, 2, 3\}$.

The final contraction is performed on L5b and results in the chain labeled L4. In this case the interval $23/3$ is contracted in lattice L5b. As can be seen in the Boolean algebra to the left of L4 this lattice has *five* prime intervals that have been contracted. If instead of working with L5b we had stayed with L5a where the Boolean algebra is on top, then there are two intervals that can be contracted, and both of them involve previously contracted intervals. Contracting either of them results in a lattice isomorphic to the chain in L4, and that chain must have *five* prime intervals that have been contracted. $\diamond$

Now, the computational complexity measures can be formalized. For a standard problem it is common to consider three different computational complexities; (1) the best case for determining the answer, (2) the average case for determining the answer, and (3) the worst case for determining the answer. Currently, not enough is known about the distribution of the numbers of each type of path independent

choice function to be able to determine the average number of prime intervals that must be identified for the class. It is, however, possible to determine the best and worst cases and the results below accomplish that task.

First to simplify notation, let P denote the class of LLD lattices, D the class of distributive lattices, W the class of chains of Boolean algebras and S the class of chains. Then let P^* be the class of LLD lattices that are not distributive, D^* the class of distributive lattices that are not chains of Boolean algebras or chains and W^* the class of chains of Boolean algebras that are not chains. Note that $S = S^*$. Thus, the starred classes are in some sense, "pure" representatives of their class.

Definition 3 Let C be a path independent choice function defined on V and let J be the associated idempotent action semigroup. The *computational complexity* of J, $k(J)$, is the number of prime intervals in the Boolean algebra 2^V that are contracted in J.

Here we see that the computational complexity of a particular choice function is measured by the number of prime intervals that must be contracted in the Boolean algebra in order to construct the action semigroup for the choice implementing semiautomaton. This measure simply reflects the effort required to identify the collections of sets from which the same choice will be made.

In economic applications, a major focus is on the computational complexity of the classes of choice functions and their associated semiautomata rather than the complexity of a specific semiautomaton. This is a common event in computational complexity. Where the complexity of the a class of problems is considered, the standard approach is to identify separate complexities for the minimum complexity of the class, the average complexity of the class and the maximum complexity of the class. For choice functions, it is possible to define each of these computational complexities. For the minimum computational complexity and the maximum computational complexity of a class, the class computational complexity measure can be identified. At this stage, however, not enough is know about the number of members of each class to be able to calculate the average computational complexity.

Definition 4 For path independent choice functions defined on V with cardinality t satisfying a consistency axiom A and action semigroups J with t join irreducibles belonging to the class of *LLD* lattices B, let the *minimum computational complexity* K_{min} *of a class B* of *LLD* lattices be defined as follows:

$$K_{min}(B) = (r|r = min_{J \in B}(k(J))).$$

Definition 5 For path independent choice functions defined on V with cardinality t satisfying a consistency axiom A and action semigroups J with t join irreducibles belonging to the class of *LLD* lattices B, let the *maximum computational complexity* K_{max} *of a class B* of *LLD* lattices be defined as follows:

$$K_{max}(B) = (r|r = max_{J \in B}(k(J))).$$

Definition 6 For path independent choice functions defined on V with cardinality t satisfying a consistency axiom A and action semigroups J with t join irreducibles belonging to the class of *LLD* lattices B, let the *average computational complexity* K_{max} *of a class B* of *LLD* lattices be defined as follows:

$$K_{ave}(B) = (r|r = ave_{J \in B}(k(J))).$$

This sequence of definitions identifies: (1) a computational complexity measure for a semiautomaton implementing a particular choice function based on the number of prime intervals that must be identified in order to construct the action semigroup, (2) for a class of LLD action semigroup lattices, the computational complexity of the class by either of the minimum number of prime intervals that must be identified in order to construct a member of the class, the maximum number of prime intervals identified for a member of the class and the average number of intervals identified for the non-isomorphic members of the class. Note $K_{min}(B) < K_{ave}(B) < K_{max}(B)$ so that $K_{min}(B)$ and $K_{max}(B)$ bound $K_{ave}(B)$.[31]

Remark 3 Let V be a collection of $t \geq 3$ join irreducibles and let 2^V be the Boolean algebra on V. For discriminating choice functions defined on V, the minimum computational complexities of the following classes of action semigroups P^*, D^*, W^*, and S^* are ordered as follows:

$$K_{min}(P^*) < K_{min}(D^*) < K_{min}(W^*) < K_{min}(S^*).$$

Remark 4 Let V be a collection of $t \geq 3$ join irreducibles and let 2^V be the Boolean algebra on V. For discriminating choice functions defined on V, the maximum computational complexities of the following classes of action semigroups P^*, D^*, W^*, and S^* are ordered as follows:

$$K_{max}(P^*) < K_{max}(D^*) < K_{max}(W^*) < K_{max}(S^*).$$

Formal proofs of these results are presented in Johnson [34]. The primary technique used in the proofs is to provide general examples on t join irreducibles for each of the distinguishing cases. The remarks above demonstrate that both $K_{min}(B)$ and $K_{max}(B)$ provide the same ordering of the computational complexities of the classes of choice functions. Finally, while it is not yet possible to determine $K_{ave}(B)$ for arbitrary sized domains, direct computation on three and four element domains confirms that $K_{ave}(B)$ orders choice function classes the same as $K_{min}(B)$ and $K_{max}(B)$ for those domains.

Referring back to the example of this section, in these sample construction there is only one member of P^* which has a $K_{min}(P^*) = K_{max}(P^*) = 1$. Similarly,

[31] A joke I heard frequently from information scientists working on computational complexity problems is that "the average case is almost always the worst case".

there is only one member of D^* and it has a $K_{min}(D^*) = K_{max}(D^*) = 2$. The class W^* has two members and we see, for the first time, that $K_{min}(W^*) = 3 \neq 4 = K_{max}(W^*)$. Finally, for S^*, $K_{min}(S^*) = K_{max}(S^*) = 5$. Of course, these numbers are only for the particular choice functions in the example. A complete treatment would have to include all possible choice functions on three elements, however, in this case, up to isomorphism, all members are represented.[32]

5 Conclusions

While differing from previous approaches to addressing the structure of economic choice automata (for example, see Futia [23] or Gottinger [26]), the results presented here characterize the structure of choice implementing semiautomata when constrained to satisfy standard economic consistency axioms. The approach is to combine previous algebraic results on choice functions of Plott [52], Johnson [31, 32] and Johnson and Dean [35–38] with work on classic algebraic automata theory results from Eilenberg [20, 21] and Holcombe [28]. Notably, the characterization results are tight in that each class of PI choice function is associated with a specific class of action semigroup.

For these choice implementing semiautomata, two different complexities are identified. The first, deriving directly from the characterization results, is implementation or algebraic complexity, which reflects the mathematical power required of the semiautomaton in order to correctly implement the choice rule being effected. When ranked by algebraic complexity, the broadest class of choice functions is identified as requiring the highest power in order to be correctly implemented. As the class of choice functions becomes increasingly restricted, the power required correctly to implement the choice rule is reduced. When viewed as choice implementing semiautomata, one intuition is that as the class of choice functions becomes more restrictive, the environments in which the semiautomata operate becomes "simpler" .

In contrast, the computational complexity which is determined by the effort required to make the action semigroups of the choice implementing semiautomaton is demonstrated to be lowest for the broadest class of choice functions and increasingly higher for the more restrictive classes. The class of choice functions with the highest computational complexity is the class of choice functions rationalized by linear orders.

Perhaps most intriguingly, the two complexities are dual with algebraic complexity being highest when the computational complexity is lowest.

[32]Note here that both of the choice machines in the class W^* have the same number of lattice points and, yet, have different computational complexities.

Appendix

Proof of Lemma 1 [33]

Proof We omit the verification that C^* is a choice function on V. We shall verify that C satisfies properties Q and CA. As a preliminary recall from Lemma 11 that if, under C, $arc(B) = B^\wedge/B$ and $arc(A) = A^\vee/A$ then $A^\vee \supseteq B^\vee$. Then it follows because B is meet irreducible in the lattice of idempotents under C, that if $K \in A^\wedge/B$ then $C(K) = A$ or $C(K) = B$.

To see this, the computation: $C(K) \bullet B = C(C(K) \cup C(B)) = C(K \cup B) = C(K)$ means that $C(K) \geq B$ and so either $C(K) = B$ or $C(K) \geq A$ in the lattice. In the latter case, $C(K) = C(K) \bullet A$ while the computation $C(K) \bullet A = C(C(K) \cup C(A)) = C(K \cup A) = C(A)$ since $K \cup A \in A^\wedge/A$. This means that $K \in A^\wedge/B$ implies $K \in A^\wedge/A$ or $K \in B^\wedge/B$ and hence that $B^\wedge$ is a relative compliment of A in the quotient $A^\wedge/B$ in 2^V. It also means that $K \in A^\wedge/B$ implies that $C^*(K) = B$.

First we verify that the inverse images under C^* are quotients in 2^V. Now the inverse images of C are unchanged under C^* unless $C^*(S) = B$. So it must be verified that the inverse image of B under C^* is an interval. We prove that $\{S : C^*(S) = B\} = A^\wedge/B$. We have just shown that for $K \in A^\wedge/B, C^*(K) = B$. Conversely, as we have shown $A^\wedge \supseteq B^\wedge$ [34] so if $C(X) = B$ then $X \in A^\wedge/B$. Now $C^*(S) = B$ if and only if $C(S) \neq A$ and $C(S) = B$, in which case $S \in B^\wedge/B$ or $C(S) = A$, in which case $S \in A^\wedge/A$. So the inverse image of B under C^* is contained in $A^\wedge/B$.

Second we verify the condition: $D \supseteq E$ implies $C^*(E) \supseteq C^*(D) \cap E$. Because C is path independent, $C(E) \supseteq C(D) \cap E$. There are four cases to check:

Case 1. Neither D nor E belong to $A^\wedge/A$.

In this case there is no change from C to C^* and so the condition holds.

Case 2. Both D and E belong to $A^\wedge/A$.

Then $C^*(D) = C^*(E) = B$ and the condition holds.

Case 3. $D \in A^\wedge/A$ and $E \notin A^\wedge/A$.

In this case $C^*(E) = C(E), C(D) = A$ and $C^*(D) = B$, so the condition to be verified is $C(E) \supseteq B \cap E$. but this is true because $C(E) \supseteq C(D) \cap E = A \cap E \supset B \cap E$.

Case 4. $D \notin A^\wedge/A$ and $E \in A^\wedge/A$.

[33] The proof reproduced here is the original proof of Johnson and Dean [35]. This proof is offered here because it is more algorithmic and instructive for this application to computational complexity. Additionally, this proof is founded on basic principles instead of relying on additional constructs as in Johnson and Dean [36].

[34] The claim is from Lemma 11 of Johnson and Dean [35] restated at the end of this proof.

The condition $D \supseteq E$ entails $C(E) \supseteq C(D) \cap E$ since C is path independent. $C(E) = A$ in this case so $A \supseteq C(D) \cap E$. We are to verify that $C^*(E) \supseteq C^*(D) \cap E$, or in this case, that $B \supseteq C(D) \cap E$. Since $A = B \cup \{x\}$ this condition will hold if $x \notin C(D)$, so we suppose for the remainder of the discussion of Case 4 that $x \in C(D)$ and derive a contraction. We prove that $D \in A^\wedge/A$ contrary to the Case 4 hypothesis.

In any even we have $D \supseteq E \supseteq A$. Let $y \in D, y \notin A$, in particular $y \neq x$. Consider $B \cup \{y\}$. The computation $B \bullet C(B \cup \{y\}) = C(B \cup C(B \cup \{y\})) = C(B \cup B \cup \{y\}) = C(B \cup \{y\})$ shows that $C(B \cup \{y\}) \geq B$ in the lattice of idempotents under C. Because B is meet irreducible, either $C(B \cup \{y\}) = B$ or $C(B \cup \{y\}) \geq A$ in the lattice.

The second alternative cannot hold as we now argue. If it did $C(B \cup \{y\}) = C(B \cup \{y\}) \bullet A = C(A \cup C(B \cup \{y\})) = C(A \cup B \cup \{y\}) = C(A \cup \{y\})$. Now $D \supseteq A \cup \{y\}$ so $C(A \cup \{y\}) \supseteq C(D) \cap (A \cup \{y\})$ and since $x \in C(D) \cap A$, it follows that $x \in C(A \cup \{y\}) = C(B \cup \{y\})$; but $x \notin B \cup \{y\}$, a contradiction.

Thus for all $y \in D, y \notin A, C(B \cup \{y\}) = B$, or $B \cup \{y\} \in B^\wedge/B$, hence for these y's, $A^\wedge \supseteq B \cup \{y\}$. But if $y \in A$, then $A^\wedge \supseteq B \cup \{y\}$ anyway, so for all $y \in D, A^\wedge \supset \{y\}$; i.e. $A^\wedge \supseteq D$, but that means $D \in A^\wedge/A$, contrary to Case 4. □

Lemma 11 [35] *If C is PI and $A > B$ in the lattice of idempotents then $A^\wedge \supset B^\wedge$.*

References

1. Abreu D, Rubinstein A (1988) The structure of Nash equilibrium in repeated games with finite automata. Econometrica 56:1259–1281
2. Arrow KJ (1959) Rational choice functions and orderings. Econometrica 26:121–127
3. Arrow KJ (1963) Social choice and individual values, 2nd edn. Yale University Press, New Haven
4. Auman RJ (1981) Survey of repeated games. In: Aumann et al. (eds) Essays in game theory and mathematical economics in Honor of Oskar Morgenstern, vol 4. of Gesellschaft, Recht, Wirtschaft, Wissenschaftsverlag Bibliographisches Institute, Mannheim, pp 11–42
5. Bandyopadhyay T (1988) Revealed preference theory, ordering and the axiom of sequential path independence. Rev Econ Stud 55:343–351
6. Banks JS, Sundaram RK (1990) Repeated games, finite automata and complexity. Games Econ Behav 2:97–117
7. Bartholdi III JJ, Orlin JB (1991) Single transferable vote resists strategic voting. Soc Choice Welf 8:341–354
8. Bartholdi III JJ, Tovey CA, Trick MA (1989) Voting schemes for which it can be difficult to tell who won the election. Soc Choice Welf 6:157–165
9. Bartholdi III JJ, Tovey CA, Trick MA (1989) The computational difficulty of manipulating an election. Soc Choice Welf 6:227–241
10. Ben-Porath E (1990) The complexity of computing a best response automaton in repeated games with mixed strategies. Games Econ Behav 2:1–12
11. Birkhoff G (1979) Lattice theory, 3rd edn. American Mathematical Society, Providence

[35] See Johnson and Dean [35].

12. Blum M (1967) A machine-independent theory of complexity of recursive functions. J Assoc Comput Mach 14:322–336
13. Campbell DE (1978) Realization of choice functions. Econometrica 46:171–180
14. Campbell DE (1978) Rationality from a computational standpoint. Theor Decis 9:255–266
15. Chatterjee K, Sabourian H (2009) Game theory and strategic complexity. Encyclopedia of complexity and systems science. Springer, New York, pp 4098–4114
16. Clifford AH, Preston GB (1961) The algebraic theory of semigroups, vol 1. American Mathematical Society, Providence
17. Clifford AH, Preston GB (1961) The algebraic theory of semigroups, vol 2. American Mathematical Society, Providence
18. Davey BA, Priestly HA (1990) Introduction to lattices and order. Cambridge University Press, Cambridge
19. Dilworth RP (1950) A decomposition theorem for partially ordered sets. Ann Math 51:161–166
20. Eilenberg S (1974) Automata, languages and machines, vol A. Academic, New York
21. Eilenberg S (1976) Automata, languages and machines, vol B. Academic, New York
22. Forsythe GE (1955) SWAC computes 126 distinct semigroups of order 4. Proc Am Math Soc 6:443–447
23. Futia C (1977) The complexity of economic decision rules. J Math Econ 4:289–299
24. Gilboa I (1988) The complexity of computing best response automata in repeated games. J Econ Theory 45:342–352
25. Gorman WM (1959) Separable utility and aggregation. Econometrica 27:469–481
26. Gottinger HW (1978) Complexity in social decision rules. In: Gottenger HW, Leinfellner W (eds) Decision theory and social ethics, pp 251–269. D. Reidel, Dordecht
27. Grillet PA (1995) The number of commutative semigroups of order N. Semigroup Forum 50:317–326
28. Holcombe WML (1982) Algebraic automata theory. Cambridge University Press, Cambridge
29. Howie JM (1991) Automata and languages. Oxford University Press, New York
30. Hurwicz L (1986) On informational decentralized systems. In: McGuire CB, Radner R (eds) Decision and organization. North-Holland, Amsterdam
31. Johnson MR (1990) Information, associativity and choice. J Econ Theory 52:440–452
32. Johnson MR (1995) Ideal structures of path independent choice functions. J Econ Theory 65:468–504
33. Johnson MR (1996) Algebraic complexity of strategy-implementing semiautomata for repeated-play games. Mimeo presented southeast economic theory and international trade meetings 1–3 Nov 1996, Florida International University, Miami, FL
34. Johnson MR (2005) Economic choice semiautomata; structure, complexities and aggregations. Presented at Econometric Society 2005 World Congress, London
35. Johnson MR, Dean RA (1996) An algebraic characterization of path independent choice functions. Presented at the third international meeting of the Society for Social Choice and Welfare, Maastricht, 1996. Available at http://markrjohnson.net/read-me/
36. Johnson MR, Dean RA (2001) Locally complete path independent choice functions and their lattices. Math Soc Sci 42:53–87
37. Johnson MR, Dean RA (2001) The construction of all lower locally distributive lattices on 4 elements. Available at http://markrjohnson.net/read-me/
38. Johnson MR, RA Dean (2002) Construction of finite lower locally distributive lattices. Presented at American Mathematics Society Meetings, San Diego, CA. Available at http://markrjohnson.net/read-me/
39. Johnson MR, Dean RA (2005) Designer path independent choice functions. Econ Theory 26:729–740
40. Kalai E (1990) Bounded rationality and strategic complexity in repeated games. In: Ichiishi T, Neyman A, Tauman Y (eds) Game theory and applications, pp 131–157. Academic, San Diego
41. Kalai E, Stanford W (1988) Finite rationality and interpersonal complexity in repeated games. Econometrica 56:397–410
42. Kelly JS (1978) Arrow impossibility theorems. Academic, New York

43. Kelly JS (1988) Social choice and computational complexity. J Math Econ 17:1–8
44. Kleitman JK, Rothschild BR, Spencer JH (1976) The number of semigroups of order n. Proc Am Math Soc 55:227–232
45. Knoblauch V (1994) Computable strategies for repeated prisoner's dilemma. Games Econ Behav 7:381–389
46. Koshevoy G (1999) Choice functions and abstract convex geometries. Math Soc Sci 38:35–44
47. Krohn KB, Rhodes JL (1962) Algebraic theory of machines. In: Proceedings of symposium on the mathematical theory of automata, pp 341–378. Polytechnic Institute of Brooklyn, New York
48. Krohn KB, Rhodes JL (1965) Algebraic theory of machines, I Prime decomposition theorem for finite semigroups and machines. Trans Am Math Soc 116:L450–L464
49. Lewis A (1985) On effectively computable realizations of choice functions. Math Soc Sci 10:43–80
50. Lewis A (1985) The minimum degree of recursively representable choice functions. Math Soc Sci 10:179–188
51. Papadimitriou CH (1992) On players with a bounded number of states. Games Econ Behav 4:122–131
52. Plott CR (1973) Path independence, rationality and social choice. Econometrica 41:1075–1091
53. Radner R (1993) The organization of decentralized information processing. Econometrica 61:1109–1146
54. Rubinstein A (1991) Comments on the interpretation of game theory. Econometrica 59:909–924
55. Satoh S, Yama K, Tokizawa M (1994) Semigroups of order 8. Semigroup Forum 49:7–29
56. Simon HA (1972) Theories of bounded rationality. In: McGuire CB, Radner R (eds) Decision and organization. North-Holland, Amsterdam
57. Strotz RH (1957) The empirical implications of the utility tree. Econometrica 25:269–280
58. Strotz RH (1959) The utility tree a correction and further appraisal. Econometrica 27:482–488
59. Turing AM (1937) On computable numbers, with an application to the entscheidungs problem. Proc Lond Math Soc 42:230–265