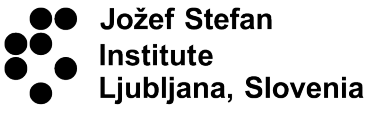
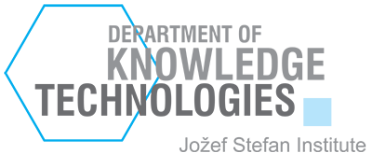
IJS delovno poročilo
DP-15651
2026

Introducing Dynamic DEX Models

Marko Bohanec

Jožef Stefan Institute, Department of Knowledge Technologies
marko.bohanec@ijs.si





Abstract

DEX is a qualitative multi-criteria decision modelling method in which attributes are organised hierarchically and aggregated by decision tables. This report introduces DynaDex, an extension of DEX that allows cycles in the structure of a model, so that feedback among attributes can be modelled. With cycles, evaluation becomes an iterative process whose outcome depends on the schedule of updates and on the initial state, and which may end in a fixed point or in a recurring cycle of states. We define DynaDex models, their node types, schedules, evaluation results and delays formally, show that DynaDex is a conservative extension of DEX, and characterise its computational power: with input sequences, DynaDex models are exactly as expressive as finite Mealy and Moore machines. We present exact algorithms for evaluation and for the identification of all attractors and their basins, including a compositional search along strongly connected components and a reduction for cyclic models built in standard DEX software. The approach is applied to synthetic models, to models reconstructed from economics, management, the psychology of decision making and the WINGS method, and to DEX models extended with feedback. The results show that fixed points do not depend on timing whereas cycles and basins do, that the timing of a model must match the modelled process, and that multiple stable states and path dependence provide new, decision-relevant information. The study draws extensively on the theory of multi-valued logical networks, discrete iterations and automata; its contribution is to bring these results to qualitative decision modelling, decision analysis and decision support. The experimental software is open source.

Keywords: DEX method, qualitative multi-criteria decision modelling, decision tables, feedback, cyclic models, discrete dynamical systems, multi-valued logical networks, attractors, update schedules, decision support

Contents

Abstract **ii**

1 Introduction **1**

2 Related Work **4**
2.1 Multi-criteria methods with feedback 4
2.2 Boolean and multi-valued logical networks 5
2.3 Update schedules, discrete iterations and fixed points 6
2.4 Automata and cellular automata 6
2.5 System dynamics and synchronous languages 7
2.6 Other relatives 8
2.7 Summary 8

3 Formal Model **10**
3.1 From DEX to DynaDex 10
3.2 Structure, scales and decision tables 10
3.3 Node types 11
3.4 State, update and fixed points 12
3.5 Schedules 13
3.6 Instances, trajectories and attractors 14
3.7 Delays 15
3.8 Stocks: cycles in tree-based DEX software 16
3.9 Basic properties 17
3.10 Computational nature 18

4 Algorithms **20**
4.1 Setting 20
4.2 Evaluation 21
4.3 Cycles of a functional graph 22
4.4 Exhaustive attractor search 22
4.5 SCC-compositional attractor search 22
4.6 Stock-map reduction 23
4.7 Occupancy distributions 24
4.8 Drawing structural graphs 25
4.9 Validation 26
4.10 Controlling evaluation, and limits 26

5 Models **28**
5.1 Synthetic models 28
5.1.1 Singletons 28
5.1.2 Double loops 29
5.1.3 Classical small networks 30

5.1.4 Counters and accumulators . . . 32
5.1.5 Delays . . . 34
5.1.6 Findings . . . 35
5.2 Reconstructed and decision-oriented models . . . 36
5.2.1 Game of Life . . . 36
5.2.2 Cobweb market . . . 38
5.2.3 Investment with a supply line . . . 38
5.2.4 Trust and cooperation . . . 39
5.2.5 Coherence-based choice . . . 39
5.2.6 A WINGS model of strategic offers . . . 40
5.2.7 Findings . . . 42
5.3 Extended DEX models . . . 43
5.3.1 Models derived from the Grignon model . . . 43
5.3.2 Employee2: assessment of academic staff . . . 46
5.3.3 Project2: assessment of projects . . . 47
5.3.4 Employee2 with lags . . . 48
5.3.5 Findings . . . 48
5.4 Overview of the models . . . 49

6 Conclusions **51**

References **55**

1. Introduction

Multi-criteria decision models (MCDM) [26], [69] are formal frameworks for evaluating and ranking decision alternatives against multiple, often conflicting criteria. In a typical MCDM problem, several attributes, such as cost, reliability, environmental impact, or risk, are assessed and aggregated into a single evaluation of alternatives. Most widely used MCDM methods are "linear" in a structural sense: they map input attributes directly to an output assessment without intermediate levels of aggregation. The weighted sum model [38] and TOPSIS [30], for example, operate on a flat decision matrix in which each criterion contributes directly to the final score.

More advanced methods introduce hierarchical structures. The Analytic Hierarchy Process (AHP) [53] organizes criteria into a tree, with pairwise comparisons at each level producing local priorities that are aggregated upward. The Multiple Criteria Hierarchy Process (MCHP) [14] extends this logic to a forest of criteria trees, enabling preference elicitation and recommendation at multiple levels of specificity. MACBETH [2] and related hierarchical approaches similarly decompose complex evaluations into structured sub-criteria. These methods remain acyclic: influence flows from leaves toward roots, and the evaluation of an alternative proceeds in a single pass from inputs to outputs.

In the broader MCDM landscape, however, several methods incorporate cycles, either implicitly or explicitly. AHP itself, while structurally hierarchical in its standard form, has been extended into the Analytic Network Process (ANP) [55], which explicitly permits feedback and interdependence among criteria and alternatives. DEMATEL [59] constructs a total influence matrix that captures direct and indirect causal relationships among factors, including feedback loops, and identifies influential versus influenced criteria. WINGS [40] builds on DEMATEL by incorporating nonlinear effects and the "power" of influencing elements, modelling reciprocal dependencies in complex systems. Beyond MCDM proper, cycles are a defining feature of several formal modelling traditions: Markov models [10] represent state transitions that may return to prior states; Petri nets [48] model concurrent systems with places and transitions that can form cycles; and Bayesian networks, while typically acyclic in their graph structure, are often coupled with decision or temporal extensions that introduce iterative inference [11]. Collectively, these approaches can be described as *dynamic models* and *dynamic systems* in which the evaluation state evolves through iteration rather than being computed in a single forward pass.

DEX (Decision EXpert) [6] is a methodology for qualitative multi-attribute decision modelling that shares the hierarchical, tree-structured character of AHP and MCHP. In a standard DEX model, attributes are organized as a tree: leaves represent input attributes, internal nodes represent aggregate attributes, and the root represents the overall evaluation. Each internal node is defined by a decision table that maps combinations of child values to a parent value. As in the tree-based methods discussed above, standard DEX models are strictly acyclic. Influence propagates from leaves to root, and evaluating an alternative is a one-pass, bottom-up computation yielding a single, stable result.

The question that motivates this study is what happens when this constraint is relaxed. *Dynamic DEX models* (*DynaDex* for short) permit cycles within the attribute structure, allowing higher-level attributes to influence those lower in the hierarchy. This seemingly minor modification has substantial consequences. When cycles are introduced, the clear distinction between

input, intermediate, and output attributes dissolves. Evaluation is no longer a one-pass computation but an iterative process. The final evaluation may, but need not converge, oscillating among multiple values. In effect, the DEX model ceases to be a simple function from inputs to outputs and becomes a computational automaton whose behaviour depends on the dynamics of iteration, not merely on the static decision tables at each node.

These changes raise fundamental questions about the nature and utility of DEX models with cycles. What properties of standard DEX models are preserved, and which are fundamentally altered? Does the introduction of cycles enable new forms of decision modelling? What are the practical implications for decision analysis and decision support?

Specifically, the objectives of this study are:

- Study related work. Investigate whether similar model types already exist. Where similar approaches exist, compare them with DynaDex: identify similarities and differences, and determine what DynaDex can benefit from knowledge established in those domains.
- Define suitable mathematical concepts and notation for describing DynaDex models, their components, and related algorithms. Propose a formal framework comprising concepts and definitions of model components, including formal definitions of the model, state, schedule, and evaluation, using consistent mathematical notation.
- Determine what the introduction of cycles brings to DEX models: what is essentially changed and what needs redefinition.
- Examine how the concepts of input, aggregate, and root attributes change in DynaDex models relative to standard DEX models, and clearly categorize node types in the model.
- Propose a suitable evaluation algorithm for DynaDex models, specifying what is required before evaluation begins and what kinds of results can be expected. Study various evaluation artifacts, such as stopping criteria, oscillations, evaluation traces, convergence, and attractors.
- Propose an efficient algorithm for identifying attractors.
- Determine how the execution and convergence of the evaluation algorithm can be controlled.
- Assess the limits of the proposed approach, for example in terms of the reasonable number of nodes or states it can handle.
- Determine the computational power of DynaDex models. Determine what type of automata DynaDex models are, and how are they related to Turing machines.
- Suggest typical DynaDex models and study their properties. Examples should include singleton models (A affects A), singleton models with input (I to A, A to A), and double-loop models.
- Suggest typical use cases for DynaDex models. Specifically, suggest possible applications in decision making, including decision analysis and decision support.
- Investigate whether components beyond attributes, scales, and decision tables can add to the functionality and usefulness of DynaDex models. Specifically, study the effects of adding a "delay" node to slow down the evaluation of an overly fast model cycle.
- Develop experimental open-source DynaDex software.

The study was exploratory, and its methods combined literature research, formal analysis and computational experiments. It proceeded in six phases. First, related approaches were identified and reviewed, with an emphasis on multi-valued logical networks, discrete iterations and fixed-point theory, automata, and multi-criteria methods with feedback; every reference was checked against the publisher's or an index record. Second, a formal framework was defined, and its main properties were stated as propositions with proof sketches. Third, algorithms for evaluation and attractor identification were designed. Each algorithm was implemented twice, as a reference implementation in Python and as the main implementation in RemObjects Oxygene, and the two were cross-checked on random models and on all models of the study. Fourth, a series of models of increasing size and realism was analysed: synthetic models whose behaviour can be predicted by hand, models reconstructed from the literature, and DEX models extended with cycles. For each model, all attractors and their basins were computed exhaustively or with an exact reduction, under several schedules and input combinations, so that the reported results are complete rather than sampled. Fifth, additional model elements, in particular delays, were studied theoretically and tested on the same models. The sixth phase is this report.

The resulting experimental software is open source and available at https://repo.ijs.si/markobohanec/dynadex.

The report is organised as follows. Section 2 reviews related work and states the similarities and differences between DynaDex and its closest relatives. Section 3 defines DynaDex models, their node types, schedules, evaluation results and delays, derives their basic properties, and characterises their computational power. Section 4 presents the algorithms for evaluation, attractor identification, the stock-map reduction and occupancy distributions, together with their validation and limits. Section 5 presents the models studied: synthetic models (Section 5.1), reconstructed and decision-oriented models from the literature (Section 5.2), and extended DEX models (Section 5.3), followed by an overview of all models and results (Section 5.4). Section 6 summarises the findings and the contributions of the study and outlines further research.

The study was carried out by the author in an iterative dialogue with the AI assistant Claude (Anthropic), mainly the Opus 5.5 model, which was used for literature search, formalisation, proof sketches, design and implementation of the algorithms and reference code, running the experiments, and drafting parts of the text. All results, code and references were reviewed and verified by the author, who takes full responsibility for the content.

2. Related Work

DynaDex lies at the intersection of two fields that have developed largely independently. On one side are multi-criteria decision methods, some of which allow feedback among criteria but treat it algebraically, with numeric weights. On the other side are discrete dynamical systems, in particular logical networks and automata, which have a mature theory of cycles, schedules and attractors but are not used for evaluating decision alternatives. This section reviews both, together with several more distant relatives, and states for each the main similarities and differences with DynaDex and what DynaDex adopts from it. Throughout, only deterministic evaluation is considered: the random asynchronous updating that dominates parts of the logical-network literature, probabilistic networks and sampling-based inference are mentioned only to delimit the scope. Table 2.1 at the end of the section summarises the comparison.

2.1 Multi-criteria methods with feedback

The Analytic Network Process (ANP) generalises the AHP from hierarchies to networks with dependence and feedback, both among elements of the same cluster and between clusters [54], [55], [56]. Local priorities obtained from pairwise comparisons are collected in a column-stochastic *supermatrix*, which is weighted by the importance of the clusters and raised to powers until it converges to a *limit supermatrix*, from which the global priorities are read. The relation between AHP and ANP is similar as between DEX and DynaDex: in both cases a hierarchical method is extended by allowing feedback, and evaluation becomes the iteration.

DEMATEL, developed at the Battelle Geneva Research Centre in the 1970s, structures a problem by expert ratings of the direct influence of every factor on every other, on a small ordinal scale [59]. The normalised direct-influence matrix D is summed over all powers into a total-influence matrix $T = D + D^2 + \cdots = D(I - D)^{-1}$, which includes the indirect influences through chains and loops; row and column sums of T then separate cause factors from effect factors. WINGS [40] builds on DEMATEL and adds the *strength* of each component, placed on the diagonal of the matrix, to its influences on the others. The matrix is scaled by the sum of the absolute values of its elements, and the total strength-influence matrix is again the convergent series $T = S + S^2 + \cdots = S(I - S)^{-1}$ of the scaled matrix S [3]. For independent criteria, WINGS reduces to additive aggregation [40].

Fuzzy cognitive maps (FCMs) [33] are signed, weighted directed graphs of concepts with activation levels. The state is updated iteratively by a weighted sum of the activations of the predecessors passed through a squashing function, and is clamped for what-if analysis. An FCM may settle in a fixed point, a limit cycle or chaotic behaviour [46]. Among the decision-oriented methods, FCMs are the closest to DynaDex in their use of iteration and in the kinds of results they produce.

Similarities and differences. All these methods accept that criteria or factors influence each other in loops, and they are used in decision analysis, which shows that decision makers can work with such models. They differ from DynaDex in three respects. First, they are numeric and, except for the squashing function of FCMs, linear: influences are weights, and aggregation is a

weighted sum. DEX decision tables are qualitative and can express non-additive interactions, such as "high only if both inputs are high". Second, ANP, DEMATEL and WINGS treat feedback *statically*: the effect of all loops is summed in closed form, which gives one result per model. Their results do not depend on an initial state or on the order of updates, and they cannot show several stable states. Section 5.2.6 shows this on a WINGS model, in which a reinforcing risk loop has two stable states in DynaDex, and the evaluation of an alternative depends on which of them the model is in. Third, in ANP and WINGS the alternatives are elements of the network or are evaluated against the resulting weights, whereas in DEX and DynaDex an alternative is an input assignment, and the same model is evaluated for many alternatives. FCMs are dynamic, like DynaDex, but their results depend on the choice of the squashing function and of its parameters, and convergence is not guaranteed; in the finite setting of DynaDex, every evaluation ends in an exactly identified attractor.

What DynaDex adopts. The positioning of DynaDex with respect to DEX, by analogy with ANP and AHP; the practice of analysing a model's influence structure before evaluating alternatives, which in DynaDex corresponds to the condensation and the classification of nodes (Section 3.3); and the way FCM users report clamped scenarios and steady states, which corresponds to the evaluation of an alternative and its attractor. The WINGS model of Section 5.2.6 was reconstructed from a published WINGS study [3].

2.2 Boolean and multi-valued logical networks

The closest formal relatives of DynaDex are logical networks used in systems biology. Kauffman [31] modelled genes as binary nodes, each controlled by a few others through a Boolean function, and updated all nodes synchronously; since the state space is finite, every trajectory ends in an attractor, which he interpreted as a cell type. Thomas [65] introduced a logical formalism with asynchronous updating, later extended to multi-valued variables, and formulated rules linking feedback circuits to behaviour: a positive circuit is necessary for multistationarity, and a negative circuit for sustained oscillation [66]. These rules have been proved in several settings. For Boolean networks with asynchronous dynamics, Remy, Ruet and Thieffry [49] showed that two fixed points require a positive circuit in the local interaction graph of some state, and that an attractive cycle requires a negative circuit in the union of the local interaction graphs along the cycle. For multi-valued networks, Richard and Comet [51] gave the corresponding necessary condition for multistationarity, and Richard [50] proved that a negative circuit is necessary for sustained oscillations in asynchronous automata networks; as a corollary, a network without negative circuits has a fixed point. The theory of automata networks, including convergence results and the role of the update mode, is presented by Goles and Martínez [25], and Schwab et al. [58] give an overview of the standard concepts: synchronous and asynchronous updating, state transition graphs, attractors, basins and robustness. Algorithms for finding attractors in large networks include SAT-based search [16], model checking [32] and the SCC-based decomposition [41], [71] on which the compositional search of Section 4.5 is based.

Similarities. A DynaDex model is a multi-valued logical network: every node has a finite set of values, every non-input node has a table that maps the values of its direct inputs to its own value, the graph may contain cycles, and evaluation is iteration. A full standard decision table is exactly such a node table. States, trajectories, attractors, basins and transients have the same meaning in both, and networks with clamped input nodes correspond to the DynaDex treatment of inputs.

Differences. The differences lie in the interpretation and in the use of the models. DEX scales are ordered, small and semantically meaningful, and DEX tables are designed by experts

to express preferences, whereas network functions are usually Boolean or threshold functions, inferred from data or chosen at random. DEX models are built to evaluate many alternatives (input assignments) with the same model, with designated outputs, whereas biological networks usually describe one system whose inputs are fixed conditions. Finally, much of the theory concerns asynchronous, non-deterministic dynamics, in which any unstable node may be updated next, and the circuit rules do not carry over to a single deterministic schedule (Remark 1). They also require tables with a definite sign in each argument, which DEX tables do not have.

What DynaDex adopts. The vocabulary and the formal objects (state transition graph, attractor, basin, update schedule); the circuit-sign rules as an interpretation aid for monotone models; and the algorithmic idea of decomposing the network along its SCCs.

2.3 Update schedules, discrete iterations and fixed points

Robert's theory of discrete iterations [52] studies maps on finite products of finite sets and their iteration in parallel (synchronous), serial (sequential) and block-sequential modes, the discrete analogues of the Jacobi and Gauss–Seidel methods of numerical analysis. Its best-known result is that a network whose interaction graph is acyclic has a unique fixed point, which every iteration reaches in a number of steps bounded by the depth of the graph. Standard DEX evaluation is exactly this acyclic case. A second classical result is that the fixed points do not depend on the block-sequential schedule [47], while the cycles and basins do; Proposition 2 extends this to state-dependent schedules and to delays. Aracena et al. [1] characterised the block-sequential schedules that produce the same dynamics by a labelled *update digraph*, which is the view of schedules as arc delays used in Section 3.5. Gadouleau [20] generalised Robert's theorem and related bounds from Boolean networks to maps on arbitrary complete lattices, which include the products of ordered scales used in DEX.

Fixed-point theory supplies the results for monotone models. By Tarski's theorem [63], a monotone map on a complete lattice has a least and a greatest fixed point, and on a finite lattice both are reached by iteration from the bottom and from the top element; this is the basis of Proposition 4. For non-monotone maps, logic programming offers several semantics of programs with recursion through negation. Fitting's Kripke–Kleene semantics [19] and the well-founded semantics [68] are three-valued: atoms whose value depends on a cycle through negation remain *undefined*, which can be read as an interval between the lowest and the highest possible value.

Similarities and differences. These theories treat exactly the objects of DynaDex: finite maps, their iteration under different schedules, and their fixed points. They do not, however, address decision making, and logic programming is concerned with which fixed point defines the meaning of a program rather than with trajectories and cycles. DynaDex keeps both views: the fixed points, which do not depend on timing, and the attractors, which do.

What DynaDex adopts. Robert's theorem as the formal statement that DynaDex is a conservative extension of DEX (Proposition 1); the invariance of fixed points under schedules; the update-digraph view of schedules; Tarski and Kleene iteration for monotone models; and, for future work, three-valued or interval outcomes for cycles that do not settle.

2.4 Automata and cellular automata

Finite-state machines map a state and an input to a next state and an output. In a Mealy machine [39] the output depends on the state and the current input, in a Moore machine [42] on the state alone. Cellular automata are networks of identical finite-state cells on a regular lattice,

updated synchronously by one local rule; Wolfram [70] classified their behaviour and related them to dynamical systems and computation. Conway's Game of Life [22] is the best-known two-dimensional example, and the one-dimensional Rule 110 is computationally universal [12]. The effect of the update mode on cellular automata was studied by Schönfisch and de Roos [57]. In digital circuits, combinational logic with cycles but without registers was analysed by Malik [36], who showed how to decide whether such a circuit has a well-defined behaviour.

Similarities and differences. A DynaDex model with input sequences is a finite Mealy or Moore machine, and every finite machine can be represented as a DynaDex model (Theorem 1). The difference lies in the representation: an automaton enumerates its states, whereas a DynaDex model factors the state into attributes with meaningful scales and the transition function into small decision tables, in the same way as a synchronous sequential circuit. Cellular automata are DynaDex models with a regular structure and identical tables; the Game of Life is analysed as such in Section 5.2.1. A cycle without delay in DynaDex corresponds to a cyclic combinational circuit, and its meaning is fixed by the schedule (Section 3.5).

What DynaDex adopts. The characterisation of its computational power (Section 3.10), the separation of a model into a combinational part and registers (the normal form of Section 3.7), the construction for Turing universality of unbounded cellular automata, and the Game of Life as a test case for the role of the schedule.

2.5 System dynamics and synchronous languages

System dynamics [61] models stocks, flows and feedback loops with explicit delays, usually as differential or difference equations. Causal loop diagrams, its qualitative layer, classify loops as reinforcing or balancing, which corresponds to the positive and negative circuits of logical networks. Experiments with decision makers show that people systematically misjudge the dynamics of feedback systems with delays, for example by ignoring the supply line of orders already placed [60]. Qualitative simulation [34] describes the behaviour of such systems with ordinal, landmark-based values, which is close in spirit to the qualitative scales of DEX, but it predicts the set of all possible behaviours, i.e. it is non-deterministic.

Synchronous programming languages such as Lustre [28] describe a system as a set of equations over streams that are evaluated once per clock tick; the operator `pre` gives the value of a stream at the previous tick, and every cycle of dependencies must pass through such a delay. This is the well-formedness condition of Section 3.7: every cycle must have a positive total delay, and each step is the evaluation of an acyclic graph. In synchronous circuits, retiming moves registers across combinational logic without changing the input–output behaviour [35]. This suggests the open question, raised in Section 5.1.6, whether the periods of a DynaDex model depend mainly on the total delay around its cycles rather than on where the delays are placed.

Similarities and differences. System dynamics shares the purpose of DynaDex, the analysis of feedback in managerial and policy decisions, and its stocks and delays have direct counterparts in DynaDex (Section 3.8). It is, however, numeric and continuous. Synchronous languages share the formal structure of DynaDex with delays, but they are programming languages for reactive systems and have no notion of a decision alternative or an attractor.

What DynaDex adopts. Stocks and explicit delays as model elements, the well-formedness condition for cycles, loop polarity as a way of explaining results to decision makers, and decision-oriented feedback structures such as the cobweb and supply-line models of Sections 5.2.2 and 5.2.3.

2.6 Other relatives

Several other formalisms deal with cycles or time, but are further from DynaDex, mostly because they are stochastic.

Markov models [10] describe transitions between states with probabilities. A DynaDex model under a fixed schedule is a deterministic special case: every state has exactly one successor.

Dynamic Bayesian networks [15] represent time by copies of a Bayesian network for consecutive time slices. The arcs within a slice and the unrolled network as a whole remain acyclic, and feedback appears only as a dependence on the previous slice, which corresponds to a DynaDex model in which every cycle passes through a delay. Activator dynamic Bayesian networks [43] allow cyclic dependencies at a coarse time scale, under a modified acyclicity condition. Dynamic Bayesian networks have also been combined with the AHP for time-dependent multi-criteria decisions [11]. Unrolling is also a useful view of DynaDex: a trajectory of T sweeps is the DEX evaluation of the model unrolled T times.

Petri nets [44], [48] model concurrent systems by places, transitions and tokens. Their cycles describe the flow of resources rather than of evaluations, and the firing of transitions is non-deterministic, so they are only a distant relative.

Spreadsheets evaluate cells in topological order and normally treat a circular reference as an error; some programs can optionally iterate circular references up to a maximum number of iterations or until the change is small. A DEX model is in effect a spreadsheet of lookup tables, and DynaDex is its circular version, with the advantage that in a finite state space non-convergence is detected exactly.

Coherence-based reasoning describes decisions as the settling of a network of mutually constraining propositions [64]. Experiments show that during a decision, the evaluations of the attributes shift towards the option that is about to be chosen [24], [29], which is a feedback from the overall evaluation to its inputs. A qualitative DynaDex version of this process is analysed in Section 5.2.5.

2.7 Summary

Table 2.1 summarises the comparison. None of the formal results used in this report is new in itself: DynaDex models are multi-valued logical networks, finite automata and discrete iterations, and their theory is well developed. What has been missing is the connection to decision analysis. On the decision-analysis side, the methods that accept feedback are numeric and, with the exception of FCMs, static; they compute the effect of loops in closed form and cannot show several stable states, oscillations, or a dependence of the evaluation on history and timing. On the dynamical-systems side, the models are not used for evaluating alternatives against preferences. DynaDex connects the two: it keeps the qualitative, rule-based and transparent modelling of DEX and adds the dynamics of logical networks, with deterministic and explicitly specified timing.

Table 2.1: Summary comparison of DynaDex with related approaches.

Approach	Values and local rule	Cycles and time	Result	Adopted by DynaDex
ANP, DEMATEL, WINGS	numeric weights; linear aggregation	loops summed in closed form; no time	priorities, influence indicators	positioning; influence structure
Fuzzy cognitive maps	numeric activations; weighted sum and squashing	iteration	fixed point, cycle or chaos	reading clamped scenarios
Logical networks	Boolean or multi-valued; table per node	iteration; synchronous or asynchronous	attractors, basins	formal objects; circuit rules; SCC algorithms
Discrete iterations, fixed-point theory	maps on finite products or lattices	parallel, serial, block-sequential iteration	fixed points; convergence	conservative extension; schedule invariance; monotone case
Automata, cellular automata	finite states; transition table or uniform local rule	synchronous steps; registers	state sequences, behaviour classes	computational power; normal form; Game of Life
System dynamics, synchronous languages	numeric stocks and flows; stream equations	explicit delays	trajectories	stocks, delays, well-formedness
Markov models, DBNs, Petri nets	probabilities; tokens	stochastic transitions; unrolling; firing	distributions, reachability	unrolling view (deterministic part only)
Coherence-based reasoning	activations; constraint satisfaction	settling of a network	coherent choice	feedback from evaluation to attributes

3. Formal Model

This section defines DynaDex models and their evaluation. The definitions are chosen so that standard DEX is exactly the acyclic special case, and so that every notion needed later (schedule, attractor, basin, delay) has a precise meaning. Throughout, evaluation is *deterministic*: the same model, inputs, initial state and schedule always give the same result. Stochastic update rules, including random asynchronous updating as used in much of the logical-network literature, are deliberately excluded. Decision tables are assumed to be fully defined, so that each combination of their input values maps to a defined single outcome value.

3.1 From DEX to DynaDex

A *DEX model* [6] consists of attributes organised in a directed acyclic graph (DAG), discrete ordered value scales, and decision tables. Basic attributes (inputs) receive values from the decision alternative being evaluated. Every aggregate attribute obtains its value from those of its direct subordinates by a decision-table lookup. Roots are the primary outputs. Because the graph is acyclic, the attributes can be evaluated once, in topological order, and the result is unique.

DynaDex keeps all ingredients of DEX except acyclicity: the structure may be any finite directed graph, including self-loops and longer cycles. Several things that the DAG made unnecessary must then be defined explicitly:

- there is no topological order, so evaluation becomes *iteration* of table lookups;
- the order and grouping of updates, the *schedule*, becomes part of the semantics;
- aggregate attributes need *initial values*, because a node on a cycle may read its own earlier value;
- the result is no longer a single value per attribute: an evaluation may end in a fixed point or in a recurring cycle of states;
- the familiar classification of attributes into inputs, aggregates and roots no longer covers all cases.

3.2 Structure, scales and decision tables

Let V be a finite set of n nodes (attributes) and $E \subseteq V \times V$ a set of directed arcs, where $(u, v) \in E$ means that u directly affects v. Self-loops (v, v) are allowed. For a node v we write

$$S(v) = \{u : (u, v) \in E\}, \qquad T(v) = \{w : (v, w) \in E\}$$

for its direct inputs and direct dependents. As in DEX, $S(v)$ is ordered, which fixes the column order of the decision table of v. $S^*(v)$ denotes the non-reflexive transitive closure of S, i.e. the set of nodes that affect v along a directed path of length at least one. A node v is *cyclic* if $v \in S^*(v)$. In a DEX model no node is cyclic.

Every node v has a value scale X_v, a finite totally ordered set of $m_v = |X_v|$ values, identified with $\{0, 1, \ldots, m_v - 1\}$. The order may be a preference order ("better than or equal") but need not be. For $U \subseteq V$ we write $X_U = \prod_{u \in U} X_u$, and $X = X_V$ is the state space.

Definition 1 (DynaDex model)**.** *A DynaDex model is a tuple*

$$M = (V, E, (X_v)_{v \in V}, (D_v)_{v \in A}),$$

where (V, E) is a finite directed graph, $I = \{v : S(v) = \emptyset\}$ is the set of input *nodes, $A = V \setminus I$ the set of* aggregate *nodes, and for every $v \in A$*

$$D_v : X_{S(v)} \to X_v$$

is a full standard decision table (FSDT): it assigns exactly one value of X_v to every combination of values of the direct inputs of v.

A *standard DEX* model is a DynaDex model whose graph is acyclic. DynaDex is currently restricted to using FSDTs, while standard DEX allows using incompletely defined decision tables that map to value intervals [6], and there are extended variants of the method that employ probabilistic and fuzzy value distributions [67]. These features are candidates for future DynaDex extensions.

The decision tables are assumed to be given; their acquisition is not considered here. A table D_v is *monotone* if it is non-decreasing in each argument with respect to the scale orders (ties allowed). A model is monotone if all its tables are. Most DEX tables are monotone, but DynaDex does not assume it: monotone models are treated as a special case with additional properties (Proposition 4).

3.3 Node types

In DEX, the position of a node in the hierarchy determines its role: basic attributes are inputs, internal attributes are aggregates and the root is the output. With cycles this trichotomy breaks down. A cyclic graph need not have a node without dependents, and a node may be at the same time "above" and "below" another. We therefore use two independent classifications (Table 3.1).

The *local* classification uses only the degrees of a node. Inputs have no direct inputs and are clamped to the values of the alternative being evaluated; aggregates have at least one direct input and are computed by their tables; a sink has no dependents. In a DEX model the sinks are the roots.

The *global* classification is based on the cycle structure of the graph, for which we use the standard notion of strongly connected components [13], [62]. Two nodes u and v are *strongly connected* if each can be reached from the other along directed paths, i.e. if $u = v$ or both $u \in S^*(v)$ and $v \in S^*(u)$. This is an equivalence relation on V, and its classes are the *strongly connected components* (SCCs) of (V, E); every node belongs to exactly one SCC. An SCC is *cyclic* if it contains a cycle, that is, if it has more than one node or consists of a single node with a self-loop; otherwise it is *trivial*. A node is cyclic exactly when its SCC is cyclic. All SCCs of a graph can be found in time linear in $|V| + |E|$, for example with Tarjan's algorithm [62].

Contracting every SCC to a single node, and drawing an arc from one component to another whenever some arc of E leads from a node of the first to a node of the second, gives the *condensation* of the graph. The condensation is always acyclic: two components on a common cycle would be strongly connected and hence a single component. The condensation thus exposes the hierarchical, DEX-like structure that every DynaDex model retains: a DAG whose nodes are either single non-cyclic nodes or cyclic subsystems. A DEX model is the special case in which all components are trivial.

The position of a component in the condensation determines its role. A *source* component receives no arcs from other components, a *sink* component sends no arcs to other components,

and all other components are *intermediate*. Every input node forms a trivial source component on its own. The behaviour of a component depends only on its own initial values and on the values that reach it from components upstream. A cyclic source component receives nothing from outside, so its behaviour is fully determined by its own initial values; we call it *autonomous*. All other cyclic components are *driven*: they are influenced from upstream, ultimately by the inputs or by autonomous components.

The union of the cyclic components is the *cyclic core* of the model. The remaining nodes are *feed-forward* and are classified by their position relative to the core. A feed-forward node is *upstream* if it affects some cyclic node but is not affected by any (for example an input, or an aggregate that feeds a loop); *downstream* if it is affected by some cyclic node but does not affect any (for example an output computed from the values of a loop); *between* if it is affected by one cyclic component and affects another, i.e. lies on a path connecting two cyclic components; and *independent* if it neither affects nor is affected by any cyclic node, in which case it belongs to a part of the model that behaves exactly as in DEX.

Table 3.1: Classification of nodes in a DynaDex model.

Classification	Type	Definition
Local (degrees)	input	$S(v) = \emptyset$; value clamped to the alternative
	aggregate	$S(v) \neq \emptyset$; value computed by D_v
	sink	$T(v) = \emptyset$ (the roots of a DEX model)
Global (cycles)	cyclic	$v \in S^*(v)$, i.e. v lies in a cyclic SCC
	upstream	not cyclic; affects the cyclic core, is not affected by it
	downstream	not cyclic; affected by the cyclic core, does not affect it
	between	not cyclic; on a path between two cyclic SCCs
	independent	not cyclic; unrelated to the cyclic core
Role	output	member of the designated set $R \subseteq V$

Because a cyclic model may have no sink, the *output* nodes are not derived from the structure but designated: $R \subseteq V$, with the sinks as the default. Inputs are never cyclic, since they have no direct inputs. Feed-forward nodes upstream of the core are evaluated as in DEX once their own inputs are known, but under some schedules their initial values can still influence the result (Remark 2).

3.4 State, update and fixed points

A *state* is a vector $s \in X$ that assigns a value to every node. An *input assignment* $\iota \in X_I$ describes the alternative being evaluated; the states compatible with it form $X^\iota = \{s \in X : s|_I = \iota\}$, a set of size $\prod_{v \in A} m_v$.

The *local update* of node v is

$$f_v(s) = \begin{cases} D_v(s|_{S(v)}), & v \in A, \\ s_v, & v \in I, \end{cases}$$

so inputs are clamped. The *synchronous update map* $F : X \to X$ updates all nodes at once, $F(s) = (f_v(s))_{v \in V}$. For a set $B \subseteq V$ the *partial update* F_B updates only the nodes in B, all of them reading the same (old) state:

$$F_B(s)_v = \begin{cases} f_v(s), & v \in B, \\ s_v, & v \notin B. \end{cases}$$

A state s is a *fixed point* if $F(s) = s$, i.e. if every aggregate agrees with its table: $D_v(s|_{S(v)}) = s_v$ for all $v \in A$. The set of fixed points is denoted $\mathrm{Fix}(F)$, and $U(s) = \{v : f_v(s) \neq s_v\}$ is the set of *unstable* nodes of s. In a DEX model, $\mathrm{Fix}(F) \cap X^\iota$ contains exactly one state, the DEX evaluation of ι.

3.5 Schedules

A schedule determines which nodes are updated at each step. To allow both fixed periodic orders and orders that depend on the current state while keeping the whole system finite and deterministic, a schedule carries a small state of its own.

Definition 2 (Schedule)**.** *A deterministic schedule is a tuple $\Sigma = (Q, q_0, \kappa, \nu)$, where Q is a finite set of schedule states, $q_0 \in Q$ the initial one, $\kappa : Q \times X \to 2^V \setminus \{\emptyset\}$ a selector of the nodes to update, and $\nu : Q \times X \to Q$ a successor function. One* micro-step *maps a configuration (s, q) to*

$$\big(F_{\kappa(q,s)}(s),\ \nu(q, s)\big).$$

Most of the study uses *periodic block-sequential* schedules: a sequence of blocks $B_0, \ldots, B_{k-1} \subseteq A$ whose union is A, applied cyclically. This is the special case $Q = \mathbb{Z}_k$, $q_0 = 0$, $\kappa(q, s) = B_q$ and $\nu(q, s) = (q + 1) \bmod k$. One pass through all k blocks is a *sweep*.

The standard instances are:

- *synchronous*: $k = 1$, $B_0 = A$; all aggregates are updated at once from the old state;
- *sequential* in the order $a_1, \ldots, a_k$: $k = |A|$, $B_j = \{a_{j+1}\}$; each update already sees the values updated earlier in the same sweep;
- *block-sequential*: anything in between, for example *SCC-stratified* schedules whose blocks follow a topological order of the condensation.

These update modes are standard in the theory of discrete iterations and of Boolean and multi-valued networks, where synchronous (parallel), sequential (serial) and block-sequential iterations were studied systematically by Robert [52]; see [58] for an overview of update schemes in Boolean network modelling and [1] for their effect on the dynamics. These works also consider random asynchronous schemes, which are excluded here. State-dependent (*event-driven*) schedules are also covered by Definition 2, for example "update the first unstable node in a fixed order of A".

A schedule is *faithful* if every state that remains unchanged forever under it is a fixed point of F. A periodic block-sequential schedule whose blocks cover A is faithful: if a sweep leaves s unchanged, every block has left it unchanged, so every aggregate agrees with its table. The event-driven schedule above is faithful as well. An unfaithful schedule would, for example, never update some aggregate v (a periodic schedule whose blocks do not cover A): a state in which only v disagrees with its table would then remain unchanged forever without being a fixed point. Only faithful schedules are considered.

Schedules as arc delays. A periodic block-sequential schedule can be read as a labelling of the arcs. Let $b(v)$ be the index of the block that contains v. Within a sweep, v reads the *new* value of u if $b(u) < b(v)$ and the value from the previous sweep otherwise. Writing

$$\delta(u, v) = \begin{cases} 0, & b(u) < b(v), \\ 1, & b(u) \geq b(v), \end{cases}$$

the synchronous schedule is $\delta \equiv 1$, and the sequential schedule with order π has $\delta(u, v) = 0$ if $\pi(u) < \pi(v)$ and 1 otherwise. Self-loops always have $\delta = 1$. The arcs with $\delta = 0$ form an

acyclic graph, because b strictly increases along them. One sweep is therefore an ordinary DEX evaluation of this acyclic graph, in which the arcs with $\delta = 1$ act as additional inputs carrying values from the previous sweep. This labelled graph is the *update digraph* of Aracena et al. [1], who used it to define equivalence classes of block-sequential schedules that produce the same dynamics. The same view underlies the treatment of explicit delays in Section 3.7.

3.6 Instances, trajectories and attractors

Definition 3 (Instance)**.** *An instance is a tuple* (M, ι, s^0, Σ) *of a model, an input assignment, a* seed $s^0 \in X^\iota$ *(the initial state) and a schedule.*

The seed has no counterpart in DEX: in a DAG the initial values of the aggregates are overwritten before they are read, while on a cycle they are not. Unless stated otherwise, the seed sets all aggregates to the lowest value of their scales.

The *trajectory* of an instance is the sequence of configurations (s^t, q^t), $t = 0, 1, \ldots$, with $(s^0, q^0) = (s^0, q_0)$ and each configuration obtained from the previous one by a micro-step. Inputs never change, so the trajectory stays in the finite set $X^\iota \times Q$, and since each micro-step is a function of the configuration alone, the trajectory is eventually periodic: there are a least $\tau \geq 0$ (the *transient*) and a least $p \geq 1$ (the *period*) such that $(s^{\tau+p}, q^{\tau+p}) = (s^\tau, q^\tau)$, and

$$\tau + p \leq |X^\iota| \cdot |Q|.$$

Definition 4 (Attractor and evaluation result)**.** *The* attractor *of an instance is the cycle of configurations* $(s^\tau, q^\tau), \ldots, (s^{\tau+p-1}, q^{\tau+p-1})$*. It is a* fixed point *if* s^t *is the same state for all* $t \geq \tau$*, and a* limit cycle *otherwise. The* evaluation result *of the instance is the transient* τ*, the attractor, and its projection onto the output nodes* R*.*

For a periodic schedule with k blocks the schedule state alone has period k, so p is a multiple of k and p/k is the period in sweeps. Periods are reported in sweeps throughout, which makes them comparable between schedules; for the synchronous schedule the two units coincide. Transients are likewise reported in sweeps, rounded up.

Since the trajectory is eventually periodic, evaluation can always be completed exactly by detecting the first repeated configuration (Section 4); a limit cycle is a legitimate result, not a failure to converge. A step budget is only a practical safeguard; when it is exhausted the result is reported as *unresolved*.

Definition 5 (Basin)**.** *The* basin *of an attractor, for fixed* M*,* ι *and* Σ*, is the set of seeds* $s^0 \in X^\iota$ *whose trajectory, started in* (s^0, q_0)*, ends in that attractor.*

The basins of all attractors partition X^ι, so their sizes sum to $|X^\iota|$. Starting all trajectories in q_0 loses nothing: in a periodic schedule every attractor passes through q_0, and counting seeds rather than configurations keeps basin sizes comparable between schedules. In a DEX model there is a single attractor, a fixed point whose basin is the whole of X^ι. For large models full basins are out of reach, and the reported basins are counted over *settled starts* only (Section 4); this convention is stated wherever it is used.

Occupancy distributions. A node that oscillates on a limit cycle has no single final value. Its behaviour can be summarised by the share of time it spends at each value. For an attractor with configurations $c_0, \ldots, c_{p-1}$, the *occupancy distribution* of node v is

$$\mu_v(x) = \frac{\left|\{\, j \in \{0, \ldots, p-1\} : c_j|_v = x \,\}\right|}{p}, \qquad x \in X_v.$$

A fixed point gives a 0/1 distribution, and a node alternating between two values gives $(1/2, 1/2)$. The distribution is a deterministic time share (the marginal of the unique invariant measure on the cycle), not a degree of belief, which is why it is not called a probability distribution. Some care is needed in its use:

- it depends on the schedule and on the counting unit: counting every micro-step weights nodes by how often they happen to be updated, whereas counting only configurations with $q = q_0$ gives one observation per sweep (the two coincide for the synchronous schedule);
- marginal distributions lose correlations between nodes, so the joint distribution on R should be kept where it matters;
- a distribution cannot be fed into a downstream decision table without an additional assumption (for example independence); it is an output of evaluation, not a new kind of table outcome;
- if several attractors are reachable, combining them requires weights, for example basin sizes, which amounts to an explicit assumption about the seed.

The possible representations of an evaluation result thus form a hierarchy, from the most to the least informative: the cyclic sequence of states on R, the joint occupancy distribution on R, the marginal distributions, their supports (the sets of values visited), and summary values such as a range, median or mode. Interval-, set- and distribution-valued *table outcomes* are a different matter and are left to a separate study.

3.7 Delays

A schedule assigns a delay of 0 or 1 sweep to every arc, and the same rule applies to all arcs between two blocks. Some models require more: a specific arc that acts with a lag of several periods, or timing that is fixed by the model rather than by the choice of schedule. For this purpose DynaDex models may contain *delay cells*.

Definition 6 (Delay chain)**.** *Let Σ be a periodic schedule. A delay chain of length $d \geq 1$ on a source node u consists of cells $c_u^1, \ldots, c_u^d$ with scale X_u. The cells are part of the state but belong to no block of the schedule. At the end of every sweep (when the schedule state returns to q_0) all cells latch simultaneously:*

$$c_u^1 \leftarrow s_u, \qquad c_u^j \leftarrow c_u^{j-1} \quad (j = 2, \ldots, d),$$

where the right-hand sides are the values before the latch. A node v that reads the tap c_u^d instead of u is connected to u by an arc with a delay of d sweeps*.*

A chain is shared by all readers of its source, and different readers may use different taps. Because the cells are outside the schedule, the delay of an arc read through a tap is exactly d sweeps under every periodic schedule: an explicit delay fixes the timing of its own arc, while the schedule decides the timing of all other arcs. Delay cells enlarge the state space by a factor m_u^d per chain, and they need initial values; by default each cell starts with the value of its source, which corresponds to a *settled history*.

Unified view. Schedules and delays can be described together by a delay label $\delta(u, v) \in \mathbb{N}$ on every arc: the value of u read by v in sweep t is the value u had at the end of sweep $t - \delta(u, v)$, with $\delta = 0$ meaning the current sweep. A labelled model is well formed if every cycle has a positive total delay, which is equivalent to the arcs with $\delta = 0$ forming a DAG. Each sweep is then a DEX evaluation of that DAG, with the delayed arcs supplying values from earlier sweeps.

A block-sequential schedule yields labels in $\{0, 1\}$ as above; an explicit delay of d sweeps gives $\delta = d$.

Explicit delays add two things that schedules cannot provide. First, delays of two or more sweeps, which require additional memory. Second, per-arc timing: the labelling $\delta(x, y) = 0$, $\delta(x, z) = 1$, $\delta(z, y) = 1$ on the arcs $x \to y$, $x \to z$, $z \to y$ (with x, y, z aggregates) would require $b(x) < b(y)$, $b(x) \geq b(z)$ and $b(z) \geq b(y)$, i.e. $b(x) < b(y) \leq b(z) \leq b(x)$, so no block-sequential schedule realises it. Conversely, schedules allow the timing of a whole model to be changed without editing its structure, which is useful in experiments (Section 5, Game of Life).

Neither element adds expressive power. A delay chain can be compiled into ordinary identity nodes placed in one additional final block of the schedule, and a schedule with state set Q can be compiled into a synchronous model with one additional *clock* node (scale Q, with a self-loop) and tables of the form "update if active, otherwise hold". The cost is larger tables and additional arcs. In this sense every DynaDex model has a *normal form*: a DEX DAG plus a set of registers that are updated once per sweep.

3.8 Stocks: cycles in tree-based DEX software

DEX modelling software, such as DEXi [5] and DEXiWin [7], represents a model as a tree of attributes (with linked attributes for DAGs) and cannot express backward arcs directly. In this study, cycles were introduced into DEX models by a naming convention: an aggregate attribute named `*X` together with a basic attribute named `_X` denotes a backward arc from `*X` to `_X`. On import into DynaDex, `_X` becomes a *stock* node whose table is the identity on the value of `*X`. The model itself, stored in a `.dxi` file, remains an ordinary DEX model and can still be edited and evaluated in DEXi or DEXiWin, where `_X` simply acts as an input attribute.

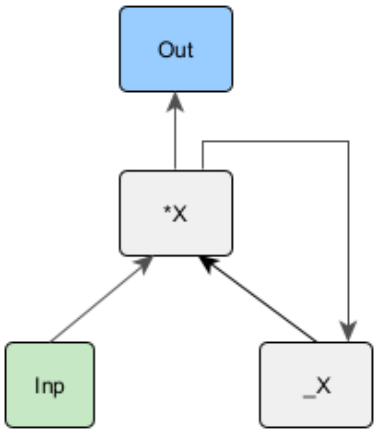


Figure 3.1: DynaDex graph of a minimal DEX model with one stock. The DEX tree `Out` ← `*X` ← {`Inp`, `_X`} becomes cyclic through the backward arc `*X` → `_X` (stock convention). Input in green, output in blue, other nodes in grey.

Figure 3.1 shows the smallest example. In the DEX model, the root `Out` has the single subordinate `*X`, whose subordinates are the basic attributes `Inp` and `_X`. After import, the stock arc `*X` → `_X` closes the loop `*X` → `_X` → `*X`. In the terms of Section 3.3, `*X` and `_X` form a cyclic component driven by the upstream input `Inp`, and `Out` is a downstream node. Apart from the explicit stock node, this is the "singleton with input" model of Section 5.1.

With the node order inputs, aggregates (children before parents), stocks, the sequential schedule evaluates in each sweep the DEX tree with the current stock values as additional inputs, and then copies each `*X` into its stock. Each cycle through a stock thus carries a delay

of one sweep; with this node order the delay sits on the arcs that read the stock, while with the stocks numbered first it would sit on the stock arc `*X` → `_X`. Both orders produce the same sequence of stock vectors, and the dynamics are fully described by the map from one stock vector to the next. This reduction is exploited in Section 4.

A stock arc can be given a longer lag $n \geq 2$, written in the `.dxi` file as the description `lag=`n (or as a `DELAY` element) of the leaf occurrence of `_X`. The importer then attaches a delay chain to the stock `_X` and connects the reader to tap n, which makes the path from `*X` through `_X` to the reader $n-1$ sweeps slower than with a plain stock, under every schedule. With a plain stock, this path has a total delay of one sweep under the sequential schedule (`_X` is updated after `*X` in the same sweep and read in the next one) and of two sweeps under the synchronous schedule (both arcs are delayed). A lag of n therefore means exactly n sweeps under the sequential schedule and $n+1$ under the synchronous one. This definition was chosen so that stocks keep their usual meaning under both schedules and models without lags behave exactly as before.

3.9 Basic properties

The following properties follow from the definitions. Proofs are given as sketches; full proofs are left for a later publication.

Proposition 1 (Conservative extension of DEX)**.** *Let M be acyclic, and let ℓ be the length of its longest path. For every input assignment ι, every seed and every faithful periodic schedule, evaluation ends in a fixed point, which is the DEX evaluation of ι. Under the synchronous schedule $\tau \leq \ell$; under any periodic block-sequential schedule the fixed point is reached within ℓ sweeps.*

Proof sketch. By induction on the length of the longest path from an input to v: a node at level j has its final value after at most j sweeps, because all its direct inputs have theirs after at most $j-1$ sweeps, and a sweep updates every aggregate at least once. The resulting state satisfies every table, and in a DAG such a state is unique. This is the acyclic case of Robert's convergence theorem [52]. □

Proposition 2 (Fixed points do not depend on timing)**.** *For every faithful schedule, the states that are stationary under the schedule are exactly the fixed points of F. With delay cells, the stationary configurations are the fixed points of F with every cell equal to its source. In particular,* $\mathrm{Fix}(F)$ *is the same for all schedules and all delay assignments.*

Proof sketch. If $F(s) = s$ then $F_B(s) = s$ for every B, so s is stationary; the converse is the definition of faithfulness. A stationary configuration with delays has every cell equal to its predecessor in the chain, hence to the source, so every node reads the current values of its inputs, and the condition reduces to $F(s) = s$. □

What the schedule and the delays do change are the transients, the limit cycles, the basins, and *which* fixed point a given seed reaches. For example, in the two-node model $a := b$, $b := a$ with seed $(0, 1)$, the synchronous schedule produces a cycle of period 2, the sequential order a, b reaches the fixed point $(1, 1)$, and the order b, a reaches $(0, 0)$. Proposition 2 extends the classical invariance of fixed points under block-sequential update schedules [47], [52] to state-dependent schedules and to delays.

Proposition 3 (Unavoidable oscillation)**.** *If* $\mathrm{Fix}(F) \cap X^{\iota} = \emptyset$*, every evaluation of M with input ι ends in a limit cycle, for every seed, faithful schedule and delay assignment.*

This follows directly from Proposition 2. The simplest example is the toggle $a := \neg a$, which oscillates with period 2 under every schedule.

Proposition 4 (Monotone models)**.** *Let M be monotone, let ι be fixed, and order X^ι componentwise, with bottom $\bot$ (all aggregates at their lowest values), top $\top$ and height $h = \sum_{v \in A}(m_v - 1)$. Then:*

1. $\mathrm{Fix}(F) \cap X^\iota$ *is non-empty and has a least element* lfp *and a greatest element* gfp *[63];*
2. *synchronous evaluation from $\bot$ increases monotonically to* lfp*, and from $\top$ decreases to* gfp*, each within h steps;*
3. *for every seed s and every t, $F^t(\bot) \leq F^t(s) \leq F^t(\top)$, so every synchronous attractor lies between* lfp *and* gfp*;*
4. *if* lfp $=$ gfp*, every seed reaches this unique fixed point synchronously within h steps;*
5. *under every faithful schedule, evaluation from $\bot$ reaches* lfp *and evaluation from $\top$ reaches* gfp*.*

Proof sketch. F is monotone on the finite lattice X^ι, so (1) is Tarski's theorem. For (2), $\bot \leq F(\bot)$ and monotonicity give an ascending chain, which stabilises within h steps at a fixed point; by induction it stays below every fixed point, so it is the least one (Kleene iteration). The dual argument applies from $\top$. (3) follows by applying F t times to $\bot \leq s \leq \top$, and (4) is immediate. For (5), the invariant $s \leq F(s)$ holds at $\bot$ and is preserved by every partial update F_B, so the trajectory is ascending, stabilises, and by faithfulness ends in a fixed point. The invariant $s \leq x$ for every fixed point x is also preserved, since $F_B(s) \leq F_B(x) = x$; hence the limit lies below lfp and equals it. The case of $\top$ is dual. □

Proposition 4 gives a cheap test: two evaluations, from the lowest and from the highest seed, bound all fixed points and all synchronous attractors, and if they agree, the result does not depend on the seed. If some tables are non-increasing in some arguments, reversing the scale order of some nodes restores monotonicity exactly when the signed graph is balanced, i.e. when every cycle of the underlying undirected graph has an even number of negative arcs.

Remark 1 (Feedback signs). In logical networks, feedback circuits are classified as positive or negative, and it is known that a positive circuit is necessary for several stable states and a negative circuit for sustained oscillation [65], [66]. These rules were proved for Boolean networks with local interaction graphs [49] and for multi-valued networks [50], [51] (Section 2.2). These results concern asynchronous, non-deterministic dynamics, and they apply only to tables with a definite sign in each argument; for general FSDTs the sign of an arc is not defined. Under a single deterministic schedule they do not carry over directly: in the positive three-node ring $a := c$, $b := a$, $c := b$, the sequential order c, b, a with seed $(1, 0, 0)$ produces a cycle of period two sweeps (six micro-steps), in which a block of 1s circulates around the ring, although the model is monotone, while the order a, b, c reaches $(0, 0, 0)$. A negative circuit is therefore not necessary for oscillation under a deterministic schedule.

Remark 2 (Seed dependence outside the cyclic core). In the limit, only the values of cyclic nodes carry information from the seed. With a schedule that is not SCC-stratified, however, the initial values of feed-forward nodes upstream of the core are read by the core before they have settled, and can push it into a different basin. An SCC-stratified schedule, which first settles each component and then proceeds downstream, removes this effect; this is also the idea behind the SCC-based algorithms of Section 4.

3.10 Computational nature

With clamped inputs, an instance is an *autonomous* finite dynamical system: the micro-step is a function on the finite set $X^\iota \times Q$ (extended by the delay cells, if any), and its graph is a

functional graph whose cycles are the attractors. If instead the inputs are allowed to change from one sweep to the next, the model reads a sequence of input values and produces a sequence of output values, and it becomes a finite-state transducer.

Theorem 1 (DynaDex and finite automata)**.**

1. *Every DynaDex model with a finite schedule and input sequences is a finite Mealy machine with at most $|X_A| \cdot |Q|$ states (times the state space of the delay cells, if any). It is a Moore machine if no arc path of total delay 0 leads from an input to an output; in particular, every model under the synchronous schedule is a Moore machine.*

2. *Conversely, every finite Mealy (or Moore) machine with state set Z, transition function Δ and output function Λ is realised by a synchronous DynaDex model with three nodes: an input i, a state node z with scale Z and a self-loop, with table $D_z(i, z) = \Delta(z, i)$, and an output node y with $D_y = \Lambda$.*

Proof sketch. (1) The configuration $(s|_A, q)$ is the machine state; one sweep computes the next state and the outputs from the current state and the current inputs. The outputs depend on the current inputs only through arcs of total delay 0, which is the Mealy case; without such paths they are functions of the state alone. (2) is a direct construction. □

DynaDex models with input sequences are therefore exactly as expressive as finite Mealy and Moore machines. A synchronous DynaDex model is a *factored* Moore machine, in the same sense that a synchronous sequential circuit is: its advantage over a flat automaton is a compact and readable representation, not additional computational power. In the normal form of Section 3.7, the registers hold the state z, and one sweep computes $z' = \Delta(z, u)$ and $y = \Lambda(z, u)$ by a DEX evaluation of the zero-delay DAG. Two special cases are worth noting. A standard DEX model is purely combinational: it has no memory and its output depends only on the current input. An acyclic model with input sequences and delays has bounded memory (a definite machine); unbounded memory requires cycles. The "singleton with input" model of Section 5.1 is the three-node construction of Theorem 1(2) and thus the universal form of a finite automaton in DynaDex.

Turing equivalence is a different question, because every DynaDex model is finite. It requires an unbounded family of models with a uniform description, for example an unbounded chain of identical cells, each connected to its neighbours by the same table. Such a chain is a one-dimensional cellular automaton, a special case of a DynaDex model with a synchronous schedule, and suitable cellular automata are known to be computationally universal; the best-known example is the elementary cellular automaton Rule 110 [12]. A Turing machine with tape alphabet Γ and state set Q_M can also be embedded directly, with cell scale $\Gamma \cup (Q_M \times \Gamma)$ and tables over the cell and its two neighbours. Within the finite setting of this report, DynaDex models are finite automata.

4. Algorithms

This section presents the algorithms developed for DynaDex: evaluation of a single instance, identification of all attractors of a model for given inputs, a reduction that makes the analysis of models obtained from `.dxi` files tractable, and the computation of occupancy distributions. All algorithms are deterministic and exact: they never rely on a convergence tolerance, and an oscillation of value assignments to nodes is reported as a result, not as a failure. They were implemented twice, as a Python reference and in Oxygene, and the two implementations were checked against each other (Section 4.9). Table 4.1 gives an overview.

Table 4.1: Overview of the DynaDex algorithms.

Algorithm	Result	Applicability and cost
Evaluation (4.2)	transient, attractor and output values of one instance	any model and schedule; time $\tau + p$ micro-steps
Cycles of a functional graph (4.3)	all cycles of a map on a finite set, and the cycle reached from every element	core routine of the next two; linear time
Exhaustive attractor search (4.4)	all attractors, basin sizes, longest transient	any model and schedule; time and memory linear in $\lvert X^{\iota}\rvert \cdot \lvert Q\rvert$
SCC-compositional search (4.5)	all attractors (no basins)	periodic, state-independent schedules; exponential only in the largest SCC
Stock-map reduction (4.6)	all attractors and basins of models from `.dxi` files	sequential schedule; space of stock values only
Occupancy distributions (4.7)	time shares of node values	post-processing of any of the above
Structural graphs (4.8)	drawing of the model structure	any model; independent of the schedule

4.1 Setting

All algorithms work on configurations (s, q), where $s \in X^{\iota}$ is a state compatible with the clamped inputs ι and $q \in Q$ is the schedule state (Section 3.5). If the model contains delay cells, their values are part of s. We write

$$\Phi(s, q) = \left(F_{\kappa(q,s)}(s),\ \nu(q, s)\right)$$

for one micro-step, extended by the latch of the delay cells (Definition 6) whenever the schedule returns to q_0. The configuration space $C = X^{\iota} \times Q$ is finite and Φ is a function on it, so the graph with an arc from every configuration to its successor is a *functional graph*: every vertex has exactly one outgoing arc. Each connected part of a functional graph consists of one cycle with trees leading into it. The cycles are exactly the attractors, the trees contain the transients,

and the vertices that reach a given cycle form its basin. Everything below is a way of exploring this graph without, or with as little as possible, constructing it explicitly.

Configurations are encoded as integers by the mixed-radix representation of the aggregate values (the scale sizes m_v are the radices) followed by the schedule state, so that sets of configurations can be stored in plain arrays. The inputs need no encoding because they are clamped.

For a periodic block-sequential schedule with k blocks the schedule state is simply $q = t \bmod k$. Periods are reported in sweeps (p/k) and transients in sweeps rounded up, as in Section 3.6.

4.2 Evaluation

Evaluation runs a single instance (M, ι, s^0, Σ) forward and reports where it ends. For an acyclic model it coincides with the standard DEX evaluation.

What is needed before starting. The input assignment ι is always needed. A seed for the aggregates is needed only if the model contains a cycle or a delay; the default is the lowest value of every scale. Note that the seed matters not only on the cyclic nodes: under a schedule that is not SCC-stratified, the initial values of feed-forward nodes upstream of a cycle are read by the cycle before they settle (Remark 2). Finally, a schedule is needed, with the synchronous schedule as the default, and a step budget as a safeguard.

Procedure.

1. Compute the SCCs of the model. If none of them is cyclic and the model has no delay cells, evaluate the aggregates once in topological order and return the resulting fixed point with $\tau = 0$. The seed is not used.
2. Otherwise set $(s, q) = (s^0, q_0)$ and $t = 0$, and apply Φ repeatedly. Store every visited configuration in a hash table together with the time t of its first visit.
3. When the current configuration is already in the table, with time t', stop: the transient is $\tau = t'$, the period is $p = t - t'$, and the configurations stored at times $\tau, \ldots, t-1$ form the attractor.
4. Classify the attractor as a *fixed point* if the state s is the same in all its configurations and as a *limit cycle* otherwise. If the step budget is exhausted before step 3 applies, return *unresolved* together with the last configuration.

Output. The kind of result (fixed point, limit cycle or unresolved), the transient and the period, the attractor configurations in order, and the full trace of the evaluation. The trace is useful for explaining the result to a decision maker, because it shows how the values of the attributes develop from the seed. For a limit cycle there is no single final state; as a canonical representative the algorithm returns the lexicographically smallest state of the cycle, so that the results of different runs can be compared. The projection of the attractor onto the output nodes R, and the occupancy distributions of Section 4.7, are derived from the attractor.

Correctness and cost. Since C is finite and Φ is deterministic, the trajectory repeats a configuration after at most $|C|$ micro-steps, so the procedure always terminates with the exact transient and period (Section 3.6). In practice it needs $\tau + p$ micro-steps, each costing $\sum_{v \in \kappa(q,s)} |S(v)|$ table-index operations, and memory for $\tau + p$ configurations. For very long trajectories the hash table can be replaced by Brent's cycle-detection algorithm [9], which finds τ and p with a constant amount of additional memory, at the cost of re-running part of the trajectory and of not keeping the trace. Both variants gave identical results in all tests.

4.3 Cycles of a functional graph

The two attractor-search algorithms below share one elementary routine: finding all cycles of a function g on a finite set $\{0, \ldots, N-1\}$, given as an array of successors. The routine walks from every element that has not been visited yet, marking the elements of the current walk with the walk's identifier, until it meets an already marked element. If that element was marked in an earlier walk, the current walk ends in an attractor that is already known, and all elements of the walk are assigned to it. If it was marked in the current walk, the walk has closed a new cycle, which is recorded as a new attractor; the elements of the walk before the cycle are assigned to it as well. Every element is visited once, so the time is $O(N)$ and the memory two integer arrays of length N.

A second pass computes, for every element, its distance to the cycle it reaches (the length of its transient), by walking backwards along each walk from the point where it met an element of known distance. The maximum of these distances is the longest transient of the model.

4.4 Exhaustive attractor search

The exhaustive search applies the routine of Section 4.3 to the whole configuration space: it encodes every configuration, computes its successor under Φ, and finds all cycles of the resulting functional graph. It yields all attractors, the attractor reached from every configuration, the basin sizes and the longest transient.

Basins are counted over the configurations with $q = q_0$ only, i.e. over the seeds $s^0 \in X^\iota$, in accordance with the definition of a basin in Section 3.6. Nothing is lost by this: in a periodic schedule the schedule state advances independently of s, so every attractor passes through q_0. Counting seeds rather than configurations also makes the basin sizes of different schedules directly comparable, since they always sum to $|X^\iota|$.

The method works for every model and every schedule, including state-dependent ones, but its time and memory are linear in $|C| = |X^\iota| \cdot |Q|$, which grows exponentially with the number of aggregates. With 32-bit configuration indices and the memory available in this study, the practical limit was set to about $4 \cdot 10^6$ configurations (Section 4.10).

4.5 SCC-compositional attractor search

The compositional search exploits the structure of the condensation (Section 3.3). Nodes upstream of a component never depend on it, so they evolve on their own, and the component only "sees" them as a periodic driving signal once they have reached an attractor. The attractors can therefore be built up one component at a time, from the sources of the condensation downstream.

Procedure.

1. Compute the SCCs of the dependency graph restricted to the aggregates (inputs are constants) and order them topologically, upstream first [62].

2. Treat the schedule state as an additional source component with a single attractor: the cycle $q_0, q_1, \ldots, q_{k-1}$ of length k. Keep a list of *partial attractors*, i.e. cycles of configurations restricted to the components processed so far. Initially the list contains only the schedule cycle.

3. For the next component K and every partial attractor P of period p_P, consider the *driven* state space $X_K \times \mathbb{Z}_{p_P}$. A pair (x_K, φ) consists of the values of the nodes of K and a phase φ of P. Its successor updates the nodes of K that belong to the block active at phase φ,

reading x_K for the nodes of K and the values stored in P at phase φ for the upstream nodes, and advances the phase to $(\varphi + 1) \bmod p_P$.

4. Find all cycles of this functional graph with the routine of Section 4.3. Each cycle, combined with P, is a partial attractor for the processed components extended by K; its period is a multiple of p_P. These cycles replace P in the list.

5. After the last component, the partial attractors are exactly the attractors of the whole model.

Proposition 5 (Correctness of the compositional search)**.** *For a periodic, state-independent schedule, the attractors found by the compositional search are exactly the attractors of* Φ*.*

Proof sketch. By induction over the components in topological order. The restriction of the global dynamics to the components processed so far, together with the schedule state, is itself a deterministic system, because those components do not depend on the rest. On any global attractor, this restriction runs through one of its partial attractors P, and the phase within P determines the values of all upstream nodes at every micro-step. The driven graph of step 3 is therefore exactly the restriction of the global dynamics to the processed components extended by K, on the set of configurations whose upstream part lies on P. Its cycles are the projections of the global attractors, and no cycle is lost or added. □

Trivial components need no special treatment: their value follows the upstream values with a lag, and the period does not change. Delay cells are treated as ordinary nodes of the dependency graph (a cell depends on its predecessor in the chain), which places them in the component of their source when the delayed arc closes a cycle and in a trivial component otherwise. At the end of every sweep the cells of K are latched: a cell copies the old value of its predecessor cell, the value of its source after the last block if the source is in K, or the value stored in P at the next phase if the source is upstream.

Cost. Processing a component K costs $\sum_P |X_K| \cdot p_P$, summed over the partial attractors P that reach it. The search is thus exponential only in the size of the largest SCC, not in the size of the whole model, which makes it the method of choice for models with many small cycles. What can still grow is the number of attractors itself: independent subsystems combine into all products of their attractors. This is a property of the model, not of the algorithm; for example, the Game of Life on a 4×5 torus has 461 attractors (Section 5).

Limitations and relation to existing work. The method needs the schedule to be periodic and independent of the state, because the schedule state is treated as an autonomous source; for state-dependent schedules the exhaustive search remains the fallback. Unlike the exhaustive search, it does not compute basin sizes, since it never visits the transient configurations.

The idea is not new. For synchronous Boolean networks, Mizera et al. [41] and Yuan et al. [71] decompose the network into blocks formed by its SCCs, compute the attractors of each block driven by the attractors of its parent blocks, and combine them, with a proof of correctness. Our version follows the same principle and extends it in three directions that are needed for DynaDex: multi-valued scales and arbitrary FSDTs, block-sequential schedules (through the schedule state as an additional source component), and delay cells. Symbolic methods based on binary decision diagrams or SAT solvers, which are standard for large Boolean networks [16], were not needed for the model sizes in this study.

4.6 Stock-map reduction

Models imported from `.dxi` files (Section 3.8) have a special structure: all feedback passes through the stocks `_X`, and the rest of the model is a DEX tree. Under the sequential schedule

with node order inputs, aggregates (children before parents), stocks, every aggregate is computed in a sweep from the inputs, from aggregates already computed in the same sweep, and from the stock values of the previous sweep. At the end of the sweep each stock copies its aggregate `*X`. Let Z be the set of stocks. A sweep therefore defines a map

$$G : X_Z \to X_Z, \qquad G(z) = \big(\texttt{*X}(\iota, z) \big)_{\texttt{_X} \in Z},$$

where $\texttt{*X}(\iota, z)$ is the value of `*X` obtained by a single DEX evaluation of the tree with inputs ι and stock values z.

Proposition 6 (Stock-map reduction)**.** *Under the sequential schedule, the cycles of G correspond one-to-one to the attractors of the full model, with the same period in sweeps. The fraction of seeds that end in an attractor equals the fraction of stock vectors that end in the corresponding cycle of G.*

Proof sketch. Within a sweep every aggregate is updated before it is read, so the seed values of the aggregates are never used, and the state at the end of each sweep is determined by the stock vector at its beginning. Configurations at the end of sweeps thus evolve by G, and every attractor of the full model corresponds to the cycle of G formed by its stock vectors at the sweep boundaries. Each stock vector stands for the same number $\prod_{v \in A \setminus Z} m_v$ of seeds, which gives the statement on basins. □

The reduction replaces the full configuration space by the much smaller space X_Z of stock values. For the Employee2 model of Section 5, for example, the full state space has about $5.8 \cdot 10^{11}$ states, while there are only 30 375 stock vectors. G is evaluated once for every stock vector, and the routine of Section 4.3 gives all attractors, basins and the longest transient. The reduction was validated against the full exhaustive search on two small models, where it reproduced the periods and basin fractions for every input combination. It does not cover the synchronous schedule, whose sweep updates `*X` and `_X` from the same old state and therefore needs the aggregates as part of the state.

Delay cells and settled starts. With lags, the reduced state consists of the stocks and the delay cells attached to them, and one sweep ends with the latch of the cells. The reduced space then becomes large again: for Employee2Lags it has about $9.3 \cdot 10^{12}$ elements. In such cases only *settled starts* are analysed: one trajectory for every stock vector, with every delay cell initialised to the value of its stock, as if the stocks had held their values for a long time. Each trajectory is evaluated as in Section 4.2, and identical attractors are merged. This finds all attractors that are reachable from settled starts, but not necessarily all attractors of the model, and the reported basins count settled starts only. Without delay cells, the analysis of settled starts coincides with the full stock-map analysis.

4.7 Occupancy distributions

The occupancy distribution μ_v of a node on an attractor (Section 3.6) is computed by counting the values of v along the attractor cycle. Three levels of aggregation are available:

1. *per attractor*: the distribution μ_v itself, which is the primary result and the only one that is exactly a time average;

2. *mixture over attractors*: the distributions of the individual attractors j weighted by their basin sizes,

$$\mu_v^{\text{mix}}(x) = \sum_j w_j \, \mu_v^{(j)}(x), \qquad w_j = \frac{|\text{basin}_j|}{\sum_i |\text{basin}_i|},$$

which corresponds to a uniform prior over the seeds; other weights can be supplied if some seeds are more plausible than others;

3. *over the trace*: the shares of the values among all states visited by one evaluation before the first repetition, which is a descriptive summary of a single run but not a time average.

The counting unit must be chosen explicitly: either every micro-step is counted, or only the configurations with $q = q_0$, i.e. one observation per sweep. The latter is preferable for sequential and block-sequential schedules, where counting micro-steps weights the values by how often a node happens to be updated. For the synchronous schedule both units coincide.

Three derived quantities summarise a distribution: the *mode* (the most frequent value), the *support* (the set of values visited), and the *entropy*

$$H_v = -\sum_{x \in X_v} \mu_v(x) \log_2 \mu_v(x),$$

which is zero for a node that has settled and grows with the amount of oscillation, so that it can serve as a per-node measure of instability. The support can always be derived from the distribution, but not vice versa; it is, however, less sensitive to the choice of schedule and counting unit. Distributions can also be accumulated on the fly during an evaluation, starting the count once the transient is over.

4.8 Drawing structural graphs

The structure of a model, i.e. its graph (V, E) with node types and arc kinds, is drawn automatically. The built-in drawing is a layered (hierarchical) layout, with the root-on-top orientation of DEX trees. The nodes are first numbered so that inputs come first and, as far as the cycles allow, every node comes after its direct inputs. An arc from a lower to a higher number is *forward*; all other arcs, including self-loops and arcs from delay cells, are *backward* and close the cycles of the model. For models with stocks (Section 3.8), the stocks are numbered before the aggregates, like the basic attributes of the DEX tree, so that the drawing shows the original DEX tree with the stock arcs $*\texttt{X} \to \texttt{_X}$ as the only backward arcs. Inputs are placed on the bottom row, sinks on the top row, and every other node above all of its forward predecessors; backward arcs do not affect the rows. Within a row, nodes are reordered to reduce the number of crossing arcs. Forward arcs are drawn as solid lines and backward arcs as dashed lines, so the feedback structure of the model is visible at a glance. Nodes are coloured by type: inputs, sinks, delay cells and all other nodes. The drawing does not depend on the schedule.

The same model can be exported in four forms:

- a *PNG image* with the layout described above, ready for inclusion in documents;
- a *Graphviz DOT* file [21], which Graphviz lays out with its own hierarchical algorithm and better arc routing, using the same colours and arc styles;
- a *GraphML* file in the dialect of the yEd graph editor, with the node coordinates of our layout, for manual refinement and for applying yEd's own layout algorithms;
- a *TikZ* picture for LaTeX documents, converted from the Graphviz layout of the DOT file (node positions and arc curves).

The structural figures in Section 5 were produced from the DOT files: the TikZ version is the Graphviz layout, and the PNG images are the TikZ pictures rendered at 300 dpi.

4.9 Validation

The Python reference implementation was checked by random testing. Random models with 2–6 nodes, scales of 2–3 values, up to three direct inputs per node (self-loops allowed) and at most one clamped input were generated, and each of 400 such models was analysed under three schedules (synchronous, sequential and a two-block schedule). In all 1200 cases the exhaustive and the compositional search found identical sets of attractors, and the hash and Brent variants of the evaluation gave identical results from a random seed. The two search methods share only the routine of Section 4.3, so the agreement is a meaningful check.

The Oxygene implementation was checked against the Python reference on all models of Section 5: for every model, schedule and input combination, the number of attractors, the periods, the longest transient and a signature of the largest basins were compared, and all rows agreed. The stock-map reduction was checked against the full exhaustive search as described in Section 4.6, and the compositional search with delay cells against the exhaustive search on all models with delays.

4.10 Controlling evaluation, and limits

Because every trajectory is eventually periodic, convergence never has to be "forced": evaluation always ends with an exact result. What the modeller can control is *which* result is obtained and how it is interpreted:

- the *schedule* determines the timing of all arcs and thereby the transients, the limit cycles and the basins, but not the fixed points (Proposition 2); an SCC-stratified schedule removes the influence of stale upstream values;
- *delays* fix the timing of selected arcs independently of the schedule, and introduce lags of several sweeps;
- the *seed* selects the attractor when there are several; for monotone models, the evaluations from the lowest and the highest seed bound all others (Proposition 4);
- the choice of *result representation* (attractor, occupancy distribution, support or summary value, Section 3.6) determines how an oscillation is reported to the decision maker;
- a *step budget* limits the run time of a single evaluation, at the price of a possibly unresolved result.

The limits of the approach are set by the size of the state space. Table 4.2 summarises the practical limits observed in this study. Evaluation of a single instance is cheap for all models considered, since trajectories were short: the longest transient found in any model of Section 5 was 30 sweeps (a synchronous double-loop model with loops of lengths 5 and 6), and 21 sweeps for the Game of Life on a 4×5 torus. Exhaustive analysis is limited to a few million configurations, i.e. to roughly 20 binary or 13 ternary aggregates under the synchronous schedule and fewer under sequential schedules, where $|Q|$ multiplies the space. The compositional search shifts this limit from the whole model to its largest SCC, and the stock-map reduction to the number of stocks. Beyond these limits, settled starts, sampled seeds or symbolic methods are the remaining options.

Table 4.2: Practical limits of the algorithms as used in this study.

Algorithm	Limit used	Largest case analysed
Exhaustive search	$\approx 4 \cdot 10^6$ configurations	Game of Life on a 4×5 torus, synchronous: $2^{20} \approx 10^6$ states, 461 attractors
Compositional search	largest SCC times attractor periods	Game of Life on a 4×4 torus: one SCC of 16 cells, 65 536 states (cross-checked with the exhaustive search)
Stock map, exhaustive	$1.6 \cdot 10^7$ reduced states	Employee2: 30 375 stock vectors (full space $\approx 5.8 \cdot 10^{11}$)
Stock map, settled starts	number of stock vectors	Employee2Lags: 30 375 starts (reduced space $\approx 9.3 \cdot 10^{12}$)

5. Models

This section presents the models analysed in the study, from small synthetic models with known answers (Section 5.1) to models reconstructed from the literature and DEX models extended with cycles. For each model we give its aim, its structure, the results of the analysis and the main findings.

All results were obtained with the exhaustive or compositional search of Section 4 and cross-checked between the two implementations (Section 4.9). Unless stated otherwise, all scales are binary, the seeds range over the whole state space X^ι, and periods and transients are given in sweeps (Section 3.6). The set of attractors of an instance family is summarised as a list of $p \times n$ entries, meaning n attractors of period p; for example, $1 \times 2,\ 2 \times 1$ denotes two fixed points and one limit cycle of period 2. Where basin sizes matter, they are given as $p{:}b$, an attractor of period p with a basin of b seeds.

In the structural graphs, inputs are green and delay cells yellow; solid arcs are forward arcs and dashed arcs the backward arcs that close cycles (Section 4.8).

5.1 Synthetic models

The synthetic models are small enough to be analysed by hand, so the software results can be compared with analytic predictions. They serve two purposes: to validate the algorithms, and to show in the simplest possible setting which phenomena cycles introduce into DEX models and how schedules and delays affect them.

5.1.1 Singletons

The smallest cyclic model is a single aggregate $\mathtt{A}$ that depends on itself, $\mathtt{A} := f(\mathtt{A})$ (Figure 5.1a). Its decision table is a map $f : X_{\mathtt{A}} \to X_{\mathtt{A}}$, and its dynamics is the functional graph of f. There is only one aggregate, so all schedules coincide. With two values there are four tables: the two constants and the identity give fixed points (the identity gives two of them, each with a basin of one seed), and the negation $\mathtt{A} := \neg\mathtt{A}$, the *toggle*, has no fixed point and oscillates with period 2 from every seed. By Proposition 3, the toggle oscillates under every schedule and with every delay: its oscillation is a property of the table, not of the timing.



(a) (b)

Figure 5.1: Singleton models: (a) a single node with a self-loop, $\mathtt{A} := f(\mathtt{A})$; (b) singleton with input, $\mathtt{A} := f(\mathtt{I}, \mathtt{A})$. The integrate-and-fire neuron of Section 5.1.3 has the structure of (b).

A complete enumeration of all single-node tables for small scales shows how common oscillation is when tables are unrestricted, and how it disappears under monotonicity (Table 5.1). Of the m^m tables on m values, $(m-1)^m$ have no fixed point and therefore oscillate from every seed, and more than half of the tables have at least one limit cycle for $m \geq 3$. Monotone (non-decreasing) tables have only fixed points, as expected from Proposition 4. Antitone (non-increasing) tables have fixed points and cycles of period 2 only, because $f \circ f$ is then monotone.

Table 5.1: All tables of a singleton $\mathtt{A} := f(\mathtt{A})$ on a scale of m values.

m	tables	no fixed point	with a limit cycle	monotone	antitone
2	4	1	1	3	3
3	27	8	11	10	10
4	256	81	131	35	35

Monotone tables: fixed points only. Antitone tables: fixed points and period-2 cycles only.

Adding a clamped input gives the *singleton with input*, $\mathtt{A} := f(\mathtt{I}, \mathtt{A})$ (Figure 5.1b). For every input value i the node follows the map $f(i, \cdot)$, so the input selects one of $|X_{\mathtt{I}}|$ dynamics on the same node. A simple monotone example on three values is $\mathtt{A} := \max(\mathtt{A} - 1, \mathtt{I})$: for every input value i the unique fixed point is $\mathtt{A} = i$, reached within two steps (from $\mathtt{A} = 2$ and $\mathtt{I} = 0$: $2 \to 1 \to 0$). With a time-varying input instead of a clamped one, the node is a finite-state machine whose state is the value of $\mathtt{A}$; adding a downstream output node gives the three-node construction of Theorem 1, which realises every finite Mealy or Moore machine. The singleton with input is thus the smallest DynaDex model with *memory*, and the building block of all larger ones.

5.1.2 Double loops

The double-loop family $\mathrm{DL}(a, b)$ was the first systematic experiment of the study. Its aim was to see how two interacting cycles of different lengths shape the dynamics, and to test the algorithms on a family with predictable results. A hub node $\mathtt{C}$ lies on two loops (Figure 5.2). Loop A is the chain $\mathtt{C} \to \mathtt{A1} \to \cdots \to \mathtt{A}(a{-}1) \to \mathtt{C}$ of copies, of length a counting $\mathtt{C}$ ($a = 1$ is a self-loop on $\mathtt{C}$). Loop B is the chain $\mathtt{C} \to \mathtt{B1} \to \cdots \to \mathtt{B}(b{-}1) \to \mathtt{C}$ of length $b \geq 2$, whose last node negates its input. An optional clamped input $\mathtt{I}$ overrides the hub: $\mathtt{I} = 1$ forces $\mathtt{C} = 1$. Two hub rules were compared, with x_A and x_B the values of the last nodes of the two loops:

- *AndNot*: $\mathtt{C} := \mathtt{I} \vee (x_A \wedge \neg x_B)$. The negation in the hub cancels the inverter at the end of loop B, so both loops are positive;
- *AndB*: $\mathtt{C} := \mathtt{I} \vee (x_A \wedge x_B)$. Loop A is positive and loop B negative.

All combinations $a = 1, \ldots, 6$ and $b = 2, \ldots, 6$ were analysed under the synchronous schedule, for both hub rules and for three input modes (no input, $\mathtt{I} = 0$, $\mathtt{I} = 1$): 180 instance families with up to 11 aggregates and 2048 states each. Under the synchronous schedule, the value of $\mathtt{C}$ obeys a delay recurrence; for AndNot with $\mathtt{I} = 0$,

$$\mathtt{C}(t+1) = \mathtt{C}(t+1-a) \ \wedge \ \mathtt{C}(t+1-b),$$

and for AndB the second term is negated. The results were:

- With $\mathtt{I} = 1$, every model has a single fixed point ($\mathtt{C} = 1$ and the loops filled accordingly), reached from every seed in at most $\max(a, b)$ sweeps. The input dominates the cycles. Without an input the results are identical to $\mathtt{I} = 0$.

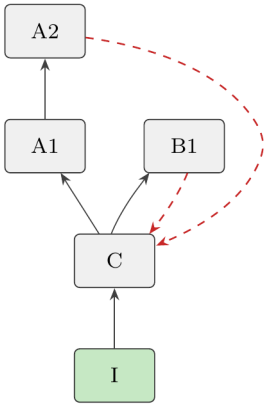


Figure 5.2: The double-loop model DL(3, 2) with input: loop A ($\mathtt{C} \to \mathtt{A1} \to \mathtt{A2} \to \mathtt{C}$) has length 3, loop B ($\mathtt{C} \to \mathtt{B1} \to \mathtt{C}$) has length 2.

- *AndNot*, $\mathtt{I} = 0$: the attractors correspond one-to-one to the binary necklaces of length $g = \gcd(a, b)$, so their number is 2, 3, 4, 6, 8 and 14 for $g = 1, \ldots, 6$, and all periods divide g. There are always exactly two fixed points, with $\mathtt{C} = 0$ and $\mathtt{C} = 1$ propagated along the loops. If a and b are coprime, the two fixed points are the only attractors; limit cycles occur only when the loop lengths share a factor (13 of the 30 pairs). These predictions were checked automatically for every row. The longest transient, 30 sweeps, occurs for $(a, b) = (5, 6)$ and $(6, 5)$.
- *AndB*, $\mathtt{I} = 0$: there is always exactly one fixed point, $\mathtt{C} = 0$. Limit cycles occur exactly when a does not divide b (17 of the 30 pairs), and all their periods divide a. For example, DL(6, 5) has $1 \times 1,\ 2 \times 1,\ 3 \times 1,\ 6 \times 2$, while DL(3, 6) converges to its fixed point from all 256 seeds.

The AndB observation follows from the recurrence $\mathtt{C}(t) = \mathtt{C}(t-a) \wedge \neg\mathtt{C}(t-b) \leq \mathtt{C}(t-a)$: along every residue class of t modulo a, $\mathtt{C}$ can only decrease, so it becomes constant on each class, and the period divides a. If a divides b, then $\mathtt{C}(t-b)$ lies in the same class, and a constant value c must satisfy $c = c \wedge \neg c$, i.e. $c = 0$. The AndNot result follows in the same way from monotone decrease along the classes modulo a and modulo b, hence modulo $\gcd(a, b)$.

The model DL(3, 2) with AndNot is also used in Section 5.1.5 as *DoubleLoop32*. With $\mathtt{I} = 0$ it has the same two fixed points under both schedules, as required by Proposition 2, but their basins differ: 1:1, 1:15 under the synchronous schedule and 1:12, 1:4 under the sequential one, with longest transients of 6 and 2 sweeps.

5.1.3 Classical small networks

The next group re-creates well-known small networks from systems biology and neuroscience as DynaDex models (Figure 5.3). Their qualitative behaviour is known, which makes them good test cases, and they show the role of the schedule more clearly than random models. All were analysed under the synchronous schedule and under the sequential schedule in node order; the results are in Table 5.2.

- *Toggle switch* (Figure 5.3a): two mutually inhibiting nodes, $\mathtt{A} := \neg\mathtt{B}$, $\mathtt{B} := \neg\mathtt{A}$, a discrete version of the bistable genetic switch of Gardner et al. [23].
- *Repressilator*(n) (Figure 5.3b): a ring of n inverters, $\mathtt{R}_i := \neg\mathtt{R}_{i-1}$ (indices modulo n); for $n = 3$ this is the synthetic genetic oscillator of Elowitz and Leibler [17].

- *Greenberg–Hastings ring*(n) (Figure 5.3c): a ring of excitable cells with the scale *rest*, *excited*, *refractory*. A resting cell becomes excited if its left neighbour is excited, an excited cell becomes refractory, and a refractory cell returns to rest. This is the one-dimensional version of the excitable-medium automaton of Greenberg and Hastings [27], and a minimal model of a ring of neurons. Each cell reads its left neighbour and itself.

- *Integrate-and-fire neuron*: a singleton with input (Figure 5.1b) whose state is a membrane potential $\mathtt{V} \in \{0, 1, 2, 3\}$, with 3 standing for a spike, and whose input is a clamped drive $\mathtt{I} \in \{0, \ldots, 3\}$: $\mathtt{V} := 0$ after a spike, otherwise $\mathtt{V} := \min(3, \max(0, \mathtt{V} + \mathtt{I} - 1))$ (integration with a leak of 1).

- *Half-centre oscillator* (Figure 5.3d): two neurons `N1`, `N2` with the scale *silent*, *firing*, *fatigued*, and a clamped binary input `Drive`. A silent neuron fires if driven and not inhibited by a firing partner, a firing neuron becomes fatigued, and a fatigued one becomes silent. This is a minimal central pattern generator of the kind reviewed by Marder and Bucher [37].

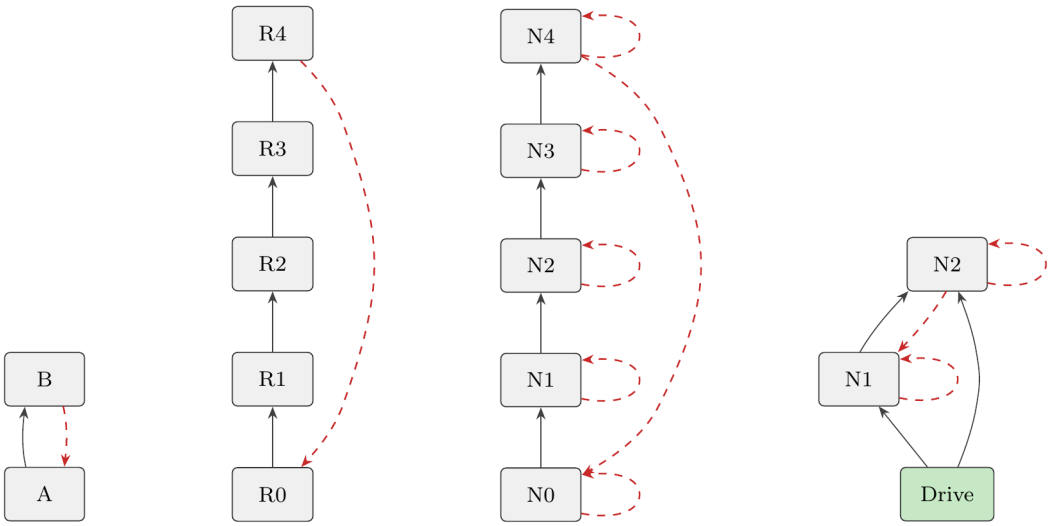


Figure 5.3: Classical small networks: (a) toggle switch, (b) repressilator with $n = 5$, (c) Greenberg–Hastings ring with $n = 5$, (d) half-centre oscillator. The node order is $\mathtt{R0}, \ldots, \mathtt{R4}$ and $\mathtt{N0}, \ldots, \mathtt{N4}$, so in the rings only the arc that closes the ring is backward.

The main observations are:

- *Spurious synchronous oscillations.* The toggle switch has the two expected stable states $(\mathtt{A}, \mathtt{B}) = (1, 0)$ and $(0, 1)$ under both schedules, but the synchronous schedule adds a limit cycle $00 \leftrightarrow 11$, in which both nodes switch at the same time. Under the sequential schedule this cycle disappears and every seed reaches one of the two stable states (basins 2 and 2). Synchronous updating is known to produce such artefacts in Boolean network models [58]; in DynaDex they appear whenever two mutually dependent nodes are updated at the same time.

- *Rings: the delay around the cycle decides.* The synchronous repressilator is a permutation of the state space (every state lies on a cycle, $\tau = 0$). An even ring is a positive loop and has the two alternating fixed points; an odd ring is a negative loop and has none, so it oscillates from every seed (Proposition 3). Under the synchronous schedule all n arcs of the ring carry a delay of one sweep, and the ring behaves like $x(t) = \pm x(t - n)$: the attractors are the binary patterns of length n (even n) or the patterns that are inverted

Table 5.2: Attractors of the classical small networks (periods in sweeps; τ = longest transient in sweeps).

Model	input	synchronous attractors	τ	sequential attractors	τ
Toggle switch	–	$1 \times 2,\ 2 \times 1$	0	1×2	1
Repressilator(3)	–	$2 \times 1,\ 6 \times 1$	0	2×1	1
Repressilator(4)	–	$1 \times 2,\ 2 \times 1,\ 4 \times 3$	0	1×2	1
Repressilator(5)	–	$2 \times 1,\ 10 \times 3$	0	2×1	1
Repressilator(6)	–	$1 \times 2,\ 2 \times 1,\ 3 \times 2,\ 6 \times 9$	0	1×2	1
GH ring(3)	–	$1 \times 1,\ 3 \times 1$	3	1×1	3
GH ring(5)	–	$1 \times 1,\ 5 \times 1$	3	1×1	3
GH ring(6)	–	$1 \times 1,\ 3 \times 1,\ 6 \times 1$	3	1×1	3
GH ring(7)	–	$1 \times 1,\ 7 \times 2$	3	1×1	3
Integrate-and-fire	$\mathtt{I} = 0$	1×1	2	(as synchronous)	
	$\mathtt{I} = 1$	1×3	1		
	$\mathtt{I} = 2$	4×1	0		
	$\mathtt{I} = 3$	3×1	1		
Half-centre	$\mathtt{Drive} = 0$	1×1	2	1×1	2
	$\mathtt{Drive} = 1$	$3 \times 1,\ 4 \times 1$	1	3×1	3

after n steps (odd n), with periods dividing n or $2n$, respectively. Under the sequential schedule only the closing arc $\mathtt{R}_{n-1} \to \mathtt{R}_0$ is delayed (Figure 5.3b), the total delay around the ring is one sweep, and the ring behaves like a single node: the toggle for odd n (one attractor of period 2, reached from all seeds) and the identity for even n (the two fixed points only). The Greenberg–Hastings ring behaves analogously: under the synchronous schedule, pulses of excitation travel around the ring with periods dividing n, whereas the sequential order, which follows the direction of propagation, lets every pulse run through the whole ring within one sweep and die out, so only the rest state remains.

- *Rhythm and coding by an input.* The integrate-and-fire neuron shows how a clamped input selects the dynamics of a singleton with input: no activity for $\mathtt{I} = 0$, three fixed points (sub-threshold memory of the potential) for $\mathtt{I} = 1$, and regular spiking with period 4 and 3 for $\mathtt{I} = 2$ and 3. The period as a function of the input is the discrete analogue of the frequency–current curve of a neuron. The reset after a spike makes the table non-monotone, which is what permits oscillation in a single node (Table 5.1).

- *Schedules can select rhythms.* With $\mathtt{Drive} = 1$ the synchronous half-centre oscillator has two rhythms, an in-phase one of period 3 (basin 3) and an anti-phase one of period 4 (basin 6). The sequential order $\mathtt{N1}, \mathtt{N2}$ breaks the symmetry, and only the rhythm of period 3 survives, reached from all 9 seeds.

5.1.4 Counters and accumulators

A register of n bits that adds a constant k modulo 2^n in every sweep is the standard example of a sequential circuit, and it was used to check how a known finite-state machine is represented in DynaDex. It was modelled in two ways (Figure 5.4):

- *direct form*, under the synchronous schedule: one node per bit $\mathtt{R}_i$, whose table computes bit i of $R + k$ from bits $0, \ldots, i$ of the old register value R; for $k = 1$ this is a binary counter;

- *circuit form*, under the sequential schedule: carry nodes $\mathtt{C}_i$, sum nodes $\mathtt{S}_i$ and register nodes $\mathtt{R}_i$ (copies of the sums), with only one or two inputs per table, as in a hardware adder (Table 5.3). The node order is carries, sums, registers, so one sequential sweep is one clock cycle: the sums are computed from the old register values, which are read through backward arcs, and then latched.

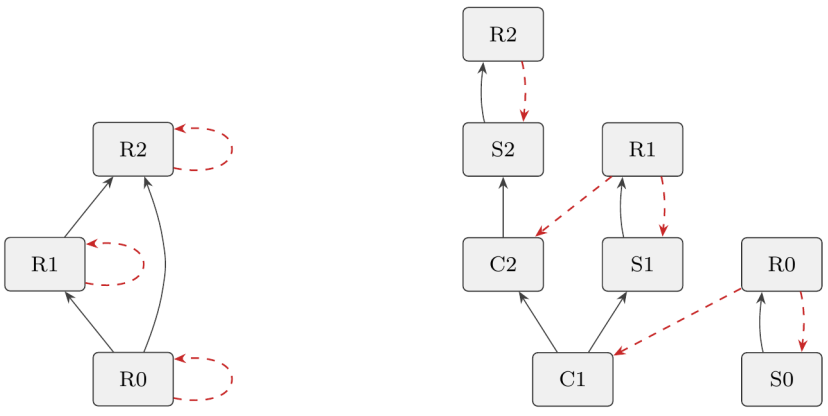
The tables of the circuit form are those of a ripple-carry adder whose second operand is the constant k. Bit i of the register is handled by a *half-adder* or *full-adder* cell that reads the old register bit $\mathtt{R}_i$ (through a backward arc) and the carry $\mathtt{C}_i$ into that bit, and produces the sum bit $\mathtt{S}_i$ and the carry $\mathtt{C}_{i+1}$ into the next bit. Writing k_i for bit i of k:

$$\mathtt{S}_i := \mathtt{R}_i \oplus \mathtt{C}_i \oplus k_i, \qquad \mathtt{C}_{i+1} := \begin{cases} \mathtt{R}_i \wedge \mathtt{C}_i, & k_i = 0, \\ \mathtt{R}_i \vee \mathtt{C}_i, & k_i = 1, \end{cases} \qquad \mathtt{R}_i := \mathtt{S}_i,$$

where $\mathtt{C}_0 = 0$, so the cell of bit 0 has a single input: $\mathtt{S}_0 := \mathtt{R}_0 \oplus k_0$ and $\mathtt{C}_1 := \mathtt{R}_0 \wedge k_0$. For the 3-bit counter ($k = 1$, Figure 5.4b) this gives $\mathtt{S}_0 := \neg\mathtt{R}_0$, $\mathtt{C}_1 := \mathtt{R}_0$, and for bits 1 and 2 the half-adder tables of Table 5.3. For comparison, the direct form needs, for bit 2 alone, a table with 8 rows over $\mathtt{R}_0, \mathtt{R}_1, \mathtt{R}_2$, in which $\mathtt{R}_2$ flips exactly when $\mathtt{R}_0 = \mathtt{R}_1 = 1$.

Table 5.3: Decision tables of the circuit form of the 3-bit counter ($k = 1$) for bit 1; the tables for bit 2 are the same with $\mathtt{R}_2, \mathtt{C}_2$ as inputs and $\mathtt{S}_2$ as output (bit 2 has no carry out). With $k_1 = 1$ (e.g. $k = 3$), $\mathtt{C}_2$ becomes $\mathtt{R}_1 \vee \mathtt{C}_1$ and $\mathtt{S}_1$ is negated.

$\mathtt{R}_1$	$\mathtt{C}_1$	$\mathtt{S}_1$	$\mathtt{C}_2$
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1



(a) (b)

Figure 5.4: A 3-bit accumulator: (a) direct form, one table per bit; (b) circuit form with carry ($\mathtt{C}$), sum ($\mathtt{S}$) and register ($\mathtt{R}$) nodes.

Both forms gave exactly the attractors predicted by elementary number theory, for all 59 combinations analysed ($n = 2, \ldots, 8$ and $n = 16$ with several values of k for the direct form, $n = 2, 3, 4$ for the circuit form): the register runs through $\gcd(k, 2^n)$ disjoint cycles of period $2^n / \gcd(k, 2^n)$. The direct form has no transient, since the machine is a permutation. The circuit form has $3n - 1$ aggregates instead of n, and hence a state space larger by a factor 2^{2n-1}, but the extra nodes are determined by the registers after at most two sweeps, so the attractors,

periods and the distribution of basins over them are the same. A 16-bit counter (65 536 states on a single cycle) served as a scale test of the exhaustive search. Finally, an acyclic ripple-carry adder evaluated for all 2^{2n} input combinations reproduced integer addition, which confirms that the acyclic case is plain DEX evaluation (Proposition 1).

The two forms illustrate the trade-off between table size and number of nodes discussed in Section 3.10. The direct form needs a table with 2^{i+1} rows for bit i, while the circuit form keeps all tables small at the price of more nodes, a larger state space and a schedule that must respect the order of the computation.

5.1.5 Delays

Three small models were used to study explicit delays (Definition 6); see Figure 5.5 and Table 5.4.

- *DelayNot*(d): the toggle with its self-loop read through a delay of d sweeps, $\mathtt{A}(t) = \neg\mathtt{A}(t-d)$;
- *DelayCopy*(d): the same with the identity table, $\mathtt{A}(t) = \mathtt{A}(t-d)$;
- *DoubleLoop32+Dd*: the model DL$(3, 2)$ of Figure 5.2 with an additional delay of $d = 2$ or 3 sweeps on the arc $\mathtt{A2} \to \mathtt{C}$.

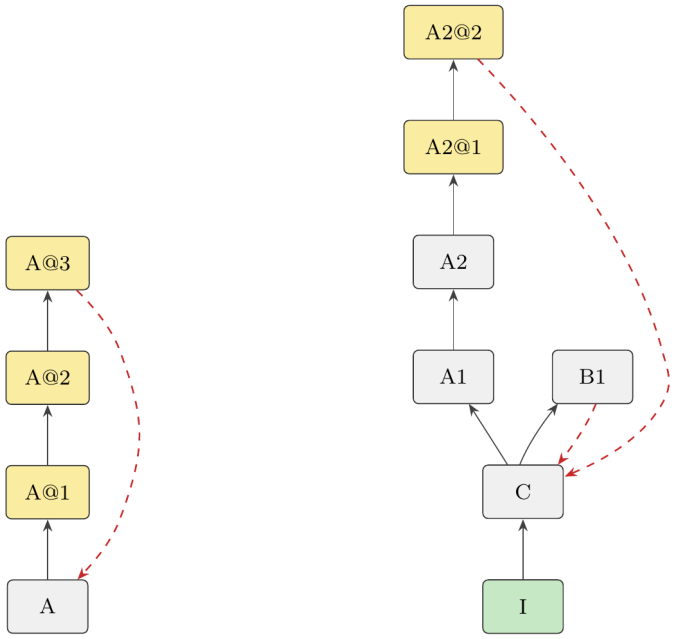


Figure 5.5: Models with delay cells (yellow): (a) a self-loop read through a delay chain of length 3 (DelayNot(3) and DelayCopy(3)); (b) DoubleLoop32 with a delay of 2 sweeps on the arc $\mathtt{A2} \to \mathtt{C}$.

The single-loop models confirm the behaviour of a cycle with total delay d. DelayCopy(d), a positive loop, has one attractor for every binary necklace of length d (3, 4 and 6 attractors for $d = 2, 3, 4$), with periods dividing d. DelayNot(d), a negative loop, has no fixed point, and its periods p satisfy $p \mid 2d$ but $p \nmid d$: period 2 for $d = 1$, 4 for $d = 2$, 2 and 6 for $d = 3$, and 8 for $d = 4$. These are the same patterns as for the synchronous rings of Section 5.1.3, in which the total delay around the ring is n. The models consist of a single scheduled node, so the schedule cannot matter here; the point of the comparison is that a delay chain on one node reproduces the dynamics of a ring of d copy nodes.

In DoubleLoop32 the delay is placed on one arc of a model with two interacting loops, and the effect of delays and schedules can be compared directly:

Table 5.4: Attractors of the delay models. Basins are counted over all seeds, including the initial values of the delay cells; τ is the longest transient in sweeps.

Model	synchronous attractors (basins)	τ	sequential attractors (basins)	τ
DelayNot(1)	2:4	1	(as synchronous)	
DelayNot(2)	4:8	1		
DelayNot(3)	2:4, 6:12	1		
DelayNot(4)	8:16, 8:16	1		
DelayCopy(2)	1:2, 1:2, 2:4	1		
DelayCopy(3)	1:2, 1:2, 3:6, 3:6	1		
DelayCopy(4)	1:2, 1:2, 2:4, 4:8, 4:8, 4:8	1		
DoubleLoop32, $\mathtt{I} = 0$	1:1, 1:15	6	1:12, 1:4	2
$+D2$	1:2, 1:42, 2:20	5	1:56, 1:8	4
$+D3$	1:126, 1:2	12	1:120, 1:8	6
DoubleLoop32, $\mathtt{I} = 1$	1:16	3	1:16	2
$+D2$	1:64	4	1:64	3
$+D3$	1:128	5	1:128	4

- The fixed points are the same in all six variants (two for $\mathtt{I} = 0$, one for $\mathtt{I} = 1$), as stated by Proposition 2.
- Delays can create new limit cycles: with $d = 2$ under the synchronous schedule, a cycle of period 2 appears and attracts 20 of the 64 seeds. With $d = 3$ it disappears again, so the effect of a delay on the cycles is not monotone in its length.
- Transients depend on the delay, but not monotonically either: under the synchronous schedule with $\mathtt{I} = 0$ the longest transient is 6, 5 and 12 sweeps for $d = 0, 2, 3$.
- The basins shift strongly with both the schedule and the delay. The share of the larger fixed-point basin is 15/16, 42/64 and 126/128 under the synchronous schedule and 12/16, 56/64 and 120/128 under the sequential one, for $d = 0, 2, 3$. Since the seeds include the initial values of the delay cells, these shares refer to different seed sets and should be compared with care.

5.1.6 Findings

The synthetic models provided a convenient basis for validating the formal DynaDex model and for testing the software that was progressively developed during the study. In addition, they illustrate the dramatic transformation brought about by introducing cycles into DEX models. With this change, DEX models evolved from a static calculation of a discrete input-to-output function into dynamic engines capable of readily emulating computational automata.

The analysis of synthetic models lead to the following conclusions, which are used in the analysis of the larger models below.

1. *Oscillation is the rule, not the exception, for unrestricted tables.* Already for a single node with three values, 11 of the 27 tables have a limit cycle. Monotone tables exclude oscillation in a single node, and in general they confine all synchronous attractors between the least and the greatest fixed point (Proposition 4).
2. *Fixed points are structural; everything else depends on timing.* In all models and all variants the fixed points were the same for every schedule and every delay, whereas limit

cycles, transients and basins changed, sometimes drastically (the toggle switch, the rings, DoubleLoop32).

3. *The delay around a cycle governs its periods.* For single loops of copy and inverter nodes, a loop with total delay D behaves like $x(t) = \pm x(t - D)$, whether the delay comes from the schedule (rings) or from delay cells (DelayCopy, DelayNot): positive loops give periods dividing D, negative loops periods dividing $2D$ but not D. For two positive loops through a common node, the gcd of the loop lengths takes this role (AndNot double loops); with a negative loop, the length of the positive loop does (AndB). Whether this extends to general tables and to several loops with delays is an open question (the retiming conjecture of Phase 5).

4. *The synchronous schedule can create artefacts.* Spurious cycles such as $00 \leftrightarrow 11$ in the toggle switch arise when mutually dependent nodes switch at the same time; sequential or SCC-stratified schedules avoid them. Conversely, models such as the Game of Life (Section 5.2.1) depend on synchronous timing by design.

5. *Inputs can dominate cycles.* A clamped input that forces the hub of a double loop removes all oscillation, and the drive of a neuron model selects between rest, memory and rhythm. For decision models this means that the dynamics must be analysed for each input assignment separately.

5.2 Reconstructed and decision-oriented models

The models of this section bring DynaDex closer to decision analysis. The first one, the Game of Life (Section 5.2.1), is still a borderline case: a well-known cellular automaton rather than a decision model, but larger than the synthetic models and a good benchmark for the role of the schedule. Four further models are small qualitative versions of classical feedback structures from economics, management and the psychology of decision making (Sections 5.2.2–5.2.5). The last model reconstructs a published WINGS model of a business decision (Section 5.2.6). In all of them the decision tables are our own: they encode the qualitative mechanism described in the source, not a calibrated quantitative model, and the aim is to see which kinds of behaviour a DynaDex model of the mechanism produces. Models derived from existing DEX models are presented in Section 5.3.

The four small models share one modelling convention. Their nodes are ordered so that one sequential sweep corresponds to one period of the story (a season, a planning period): nodes that are computed within the period come first, and *stock* nodes, which carry a state from one period to the next, read the values of the previous period through backward arcs. Under the synchronous schedule, by contrast, every arc is delayed by one period. Comparing the two schedules therefore shows which part of the behaviour is caused by the timing assumption. All scales have three values unless stated otherwise. The results are collected in Table 5.5.

5.2.1 Game of Life

Conway's Game of Life [22] is a cellular automaton on a grid of binary cells: a dead cell becomes alive if exactly three of its eight neighbours are alive, and a live cell survives if two or three are. As a DynaDex model, every cell is a binary node whose direct inputs are itself and its eight neighbours, with the Life rule as decision table; the grid is a torus, so all cells are equivalent. The model has no inputs. Its structural graph is not informative and is not shown. The Game of Life is defined with synchronous updating, and the question here is what happens when the same tables are evaluated under the sequential schedule, which updates the cells in place, row by row, so that each cell already sees the new values of the cells before it.

Table 5.5: Attractors of the cobweb, supply-line and trust models for every input combination, as $p{:}b$ (period in sweeps : basin), and the longest transient τ in sweeps.

Model	inputs	synchronous	τ	sequential	τ
Cobweb	weak, rigid	1:9	2	1:9	2
	weak, sluggish	4:9	2	2:9	2
	weak, immediate	4:9	2	2:9	2
	normal, rigid	1:9	2	1:9	2
	normal, sluggish	1:1, 4:8	2	1:9	2
	normal, immediate	1:1, 4:4, 4:4	0	1:3, 2:6	1
	strong, rigid	1:9	2	1:9	2
	strong, sluggish	4:9	2	2:9	2
	strong, immediate	4:9	2	2:9	2
Supply line	low, ignores	1:27	7	1:27	5
	low, accounts	4:27	4	2:27	4
	medium, ignores	1:1, 8:26	6	1:3, 6:24	2
	medium, accounts	1:2, 5:25	5	1:27	4
	high, ignores	1:27	7	1:27	5
	high, accounts	1:3, 6:24	2	1:27	3
Trust	none, none	1:1, 1:1, 1:1, 2:3, 2:3	1	1:3, 1:3, 1:3	1
	none, shock	1:9	4	1:9	3
	active, none	1:9	4	1:9	3
	active, shock	1:7, 1:2	2	1:6, 1:3	1

Inputs: Cobweb (Demand, Response); Supply line (Demand, Awareness of the pipeline); Trust (Regulation, Shock).

Table 5.6 gives the exhaustive results for tori of up to 20 cells ($2^{20} \approx 10^6$ states; the sequential 4×5 case was not run because its configuration space is 20 times larger). The fixed points are the same under both schedules, as Proposition 2 requires: 127, 39 and 53 still lifes on the three smaller tori. Everything else changes. The synchronous 4×4 torus has 292 limit cycles with periods up to 8, the sequential one only four cycles of period 2; the largest basin of the synchronous 4×4 torus is that of the empty grid (49 116 of 65 536 states). Single trajectories on larger tori show the same picture: under the synchronous schedule the blinker oscillates with period 2 and the glider travels around an $n \times n$ torus with period $4n$ (24 on 6×6, 32 on 8×8), whereas under the sequential schedule both patterns collapse within two sweeps into a fixed pattern of four live cells.

Table 5.6: Game of Life on small tori: attractors ($p \times n$), number of fixed points, longest transient τ (sweeps) and size of the largest basin, out of 2^{cells} states.

Torus	schedule	attractors	fixed	τ	largest basin
3×3	sync	1×127	127	2	386
	seq	1×127	127	1	46
3×4	sync	$1 \times 39,\ 2 \times 2$	39	4	3 754
	seq	1×39	39	14	1 920
4×4	sync	$1 \times 53,\ 2 \times 180,\ 4 \times 16,\ 8 \times 96$	53	9	49 116
	seq	$1 \times 53,\ 2 \times 4$	53	18	46 667
4×5	sync	$1 \times 313,\ 2 \times 32,\ 4 \times 20,\ 5 \times 8,\ 6 \times 60,\ 8 \times 20,\ 10 \times 8$	313	21	884 500

These observations agree with the study of synchronous and asynchronous updating in cellular automata by Schönfisch and de Roos [57], who found that the stationary states do not depend on the updating method while the basins can differ greatly. They also report that cyclic

dynamics occur only with synchronous updating. This does not hold for every deterministic order on small tori: under our sequential schedule, the 4×4 torus still has period-2 cycles (Table 5.6). Blok and Bergersen [4] studied a stochastic version in which only a random fraction of cells is updated in each step; such schedules are outside the deterministic scope of DynaDex. For DynaDex, the lesson is that the characteristic behaviour of Life, its oscillators and gliders, is a property of the synchronous schedule as much as of the tables. DynaDex makes this timing assumption an explicit and changeable part of the model.

5.2.2 Cobweb market

The cobweb model describes a market in which producers decide on supply before the price is known, so that supply reacts to the price of the previous period [18]. In the DynaDex version (Figure 5.6a) the inputs are `Demand` (weak, normal, strong) and the supply `Response` (rigid, sluggish, immediate), and the two aggregates are `Supply` and `Price` (low, medium, high). The price clears the market with the current supply, `Price` := clamp(`Demand` − `Supply` + 1), where clamp limits a value to the scale $\{0, 1, 2\}$, while `Supply` reads the price of the previous period: a rigid supply stays at medium, a sluggish one moves one step towards the last price, and an immediate one equals the last price (naive expectations).

With rigid supply the market settles at the price given by demand. With sluggish or immediate supply and normal demand there is a fixed point at medium supply and price, but under the sequential schedule an immediate response also has a two-period cycle (basin 6 of 9), the classical cobweb oscillation between high and low price. With weak or strong demand and a sluggish or immediate response, supply and price oscillate from every seed. This is not a dynamic instability but a consequence of the coarse scale: the equilibrium would require `Price` = `Supply` and `Price` = `Demand` − `Supply` + 1, which has no solution on the scale $\{0, 1, 2\}$ when `Demand` is 0 or 2. The model has no fixed point, and by Proposition 3 it must oscillate, between the two values that enclose the equilibrium. Qualitative scales can thus turn a stable equilibrium into a small persistent oscillation, which is better read as "between these values" than as a cycle in the real system. The synchronous schedule doubles all periods (4 instead of 2), because it delays also the arc `Supply` → `Price`, which in the story acts within the period.

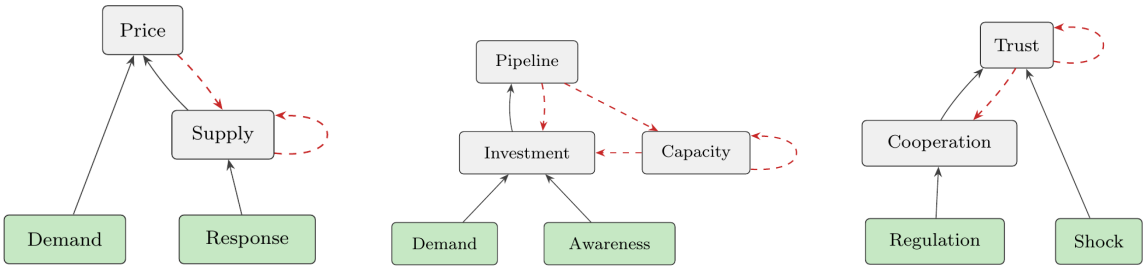


Figure 5.6: Small decision-oriented models: (a) cobweb market, (b) investment with a supply line, (c) trust and cooperation.

5.2.3 Investment with a supply line

This model (named *LimitsToGrowth* in the software) captures a typical management problem: capacity is built with a delay, and decision makers tend to ignore what has already been ordered but not yet delivered, the *supply line* [60]. The inputs are `Demand` (low, medium, high) and `Awareness` of the supply line (ignores, accounts for it); the aggregate `Investment` is decided from the gap between demand and capacity, and the stocks are `Capacity` and `Pipeline` (the

investment decided in the previous period), see Figure 5.6b. Capacity grows by the arriving pipeline and loses one level per period:

$$\begin{aligned}\texttt{Investment} &:= \text{clamp}(\texttt{Demand} - \texttt{Capacity} - a \cdot \texttt{Pipeline} + 1),\\ \texttt{Capacity} &:= \text{clamp}(\texttt{Capacity} + \texttt{Pipeline} - 1),\\ \texttt{Pipeline} &:= \texttt{Investment},\end{aligned}$$

with $a = 1$ if the supply line is accounted for and $a = 0$ otherwise.

The result reproduces the known effect. With medium demand, a decision maker who ignores the supply line over-invests, and the system enters a boom-and-bust cycle of six periods (sequential schedule) that attracts 24 of the 27 seeds; only three seeds reach the steady state. Accounting for the supply line removes the cycle: every seed reaches the steady state. The synchronous schedule, which adds a delay to every arc, produces cycles in four of the six input combinations, including cases where the sequential model is stable. As in the cobweb model, the extra delays of the synchronous schedule are not part of the story, and the sequential results are the relevant ones here. (With low demand and awareness of the supply line, investment alternates between 0 and 1 while capacity stays at its lowest level; this small oscillation is again a granularity effect.)

5.2.4 Trust and cooperation

Trust and cooperation reinforce each other: cooperation builds trust, and trust makes cooperation easier [45]. In the model (Figure 5.6c), the inputs are `Regulation` (none, active) and an external `Shock` (none, shock), both binary; $\texttt{Cooperation} := \text{clamp}(\texttt{Trust} + \texttt{Regulation})$, and `Trust` moves one step towards the current level of cooperation per period and is reduced by one level by a shock.

Without regulation and without shocks every level of trust is self-sustaining: there are three fixed points (low, medium and high cooperation and trust), and the outcome depends only on the initial level. This is path dependence in its simplest form. A shock without regulation drives the system to the low state, and regulation without shocks to the high state, from every seed. With both, there are two fixed points, *medium cooperation, low trust* and *high cooperation, medium trust*: regulation keeps cooperation up, but the shocks prevent full trust. The schedules disagree on which of the two is more likely: under the sequential schedule the better state has a basin of 6 of 9 seeds, under the synchronous one of only 2. The synchronous schedule also adds two spurious cycles of period 2 in the case without regulation and shocks.

5.2.5 Coherence-based choice

Coherence-based or parallel-constraint-satisfaction models of decision making describe a choice as the settling of a network in which the evaluation of the evidence and the emerging preference influence each other: ambiguous evidence is gradually re-interpreted in favour of the option that is ahead [24], [29]. This bidirectional influence is a cycle, which a DEX model cannot express. The DynaDex model (Figure 5.7) is a qualitative model inspired by this idea, not a replication of the quantitative activation rule of these models.

Two options, A and B, are judged by three criteria; criteria 1 and 3 support A and criterion 2 supports B. The inputs $\texttt{Imp1}, \ldots, \texttt{Imp3}$ give the stated importance of each criterion (low, medium, high), where *medium* is read as *ambiguous*. The aggregate `Lead` (B ahead, tie, A ahead) is computed from the current saliences $\texttt{Sal1}, \ldots, \texttt{Sal3}$ of the criteria; a lead requires an evidence difference of at least two levels. The options `OptionA` and `OptionB` (rejected, open, preferred) move up or down according to the lead. The saliences implement the coherence shift: a criterion of low or high importance keeps its salience, while an ambiguous one is re-read in favour of the option that currently leads (high salience if it supports the leader, low if it supports the rival).

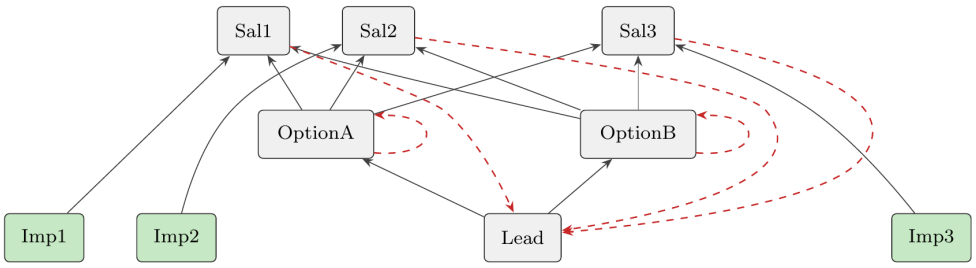


Figure 5.7: Coherence-based choice: importances (`Imp`), saliences (`Sal`), the lead and the two options. The saliences read the options, and the lead reads the saliences, which closes the cycle.

All $3^3 = 27$ importance profiles were analysed under both schedules. In all 54 cases every attractor is a fixed point: the model always reaches a decision state, never a cycle, but the number of such states ranges from 1 to 9. To turn this into a decision-support statement, every profile was evaluated from nine *prior leanings* (all combinations of the initial values of the two options, with saliences equal to the stated importances), and the final choice was recorded. The profiles fall into two clear groups, identical under both schedules:

- *decisive* profiles (11 of 27): the choice is the same from every prior leaning, A in ten profiles and B in one (only the B-criterion important). Examples: one high criterion for A and the rest low or ambiguous (e.g. $2, 1, 1$), where the ambiguous criteria are re-read in favour of A;
- *prior-preserving* profiles (16 of 27): the final choice equals the prior leaning (three leanings end in A, three in B and three undecided). These are the profiles in which the evidence is balanced (e.g. $2, 2, 0$: one high criterion for each option) or entirely ambiguous ($1, 1, 1$), but also some in which a single ambiguous opposing criterion is enough to block a decision (e.g. $0, 1, 2$).

The second group is the interesting one for decision support: here the model says that the evidence does not determine the choice, and that the decision maker's prior leaning will be confirmed by the coherence shift. A static DEX evaluation of the same profile would return a single value and hide this dependence.

5.2.6 A WINGS model of strategic offers

WINGS (Weighted Influence Non-linear Gauge System) is a method for analysing systems of interrelated factors, in which each factor has a strength and each arc an influence weight [40]. Banaś and Michnik [3] used an extended WINGS model to evaluate strategic offers of a telecommunication supplier by their financial and strategic effects. We reconstructed the structure of their model (their Figure 2) as a DynaDex model with 24 inputs and 8 aggregates (Figure 5.8). The model is acyclic with one exception: the factors *Financial risk* and *Internal risk* influence each other, which forms the only cycle.

WINGS is a continuous method, so the decision tables had to be constructed. We used a *weighted-influence compiler*: for a node with n inputs on scales of k values and the signed integer weights w_i read from the figure, each input value x_i is centred to $c_i = 2x_i - (k-1)$, the score $S = \sum_i w_i c_i$ is normalised by $D(k-1)$ with $D = \sum_i |w_i|$, and the interval $[-1, 1]$ of normalised scores is divided into k equal bins:

$$f(x_1, \ldots, x_n) = \min\Big(k-1,\ \Big\lfloor \frac{(S + D(k-1))\,k}{2D(k-1)} \Big\rfloor\Big).$$

The rule uses integer arithmetic only, so the bin boundaries are exact. The resulting tables are monotone in each input, increasing or decreasing according to the sign of its weight; this is a

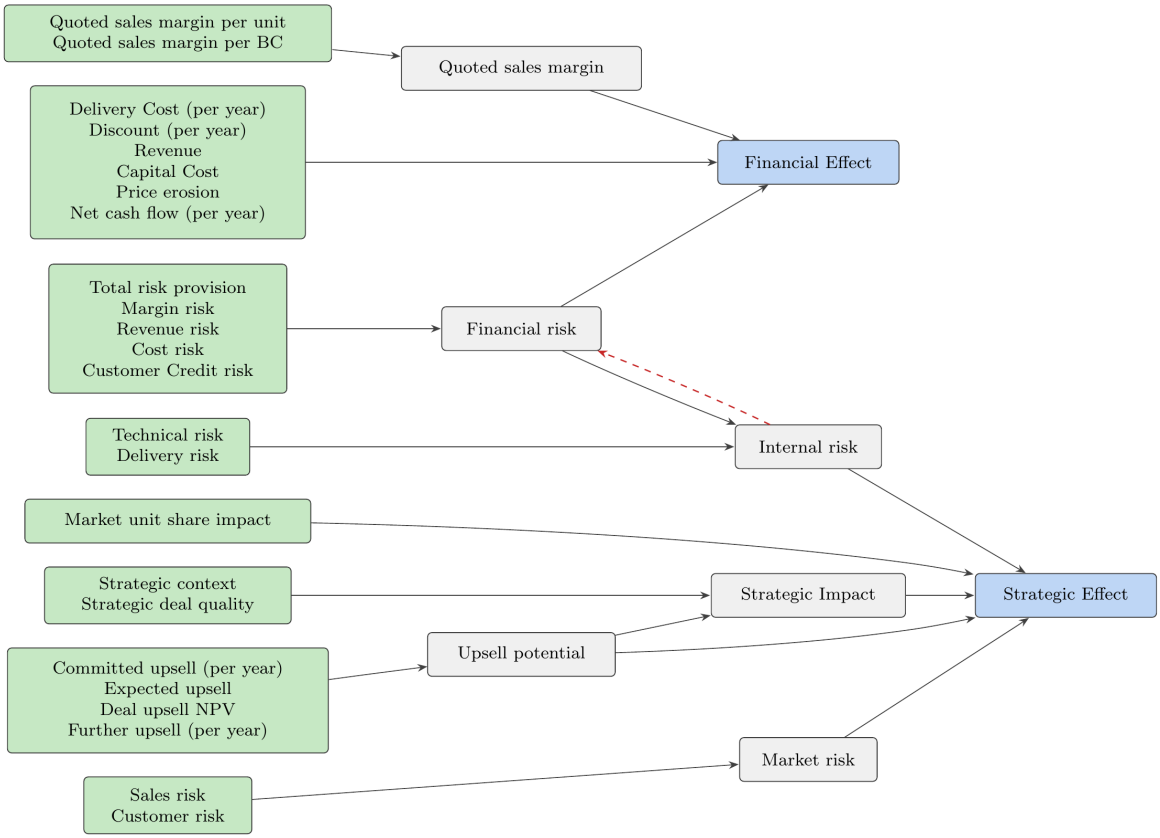


Figure 5.8: DynaDex reconstruction of the WINGS model of Banaś and Michnik [3]. For readability, the inputs of each aggregate are drawn as one box; the complete graph has 24 separate input nodes. The dashed arc closes the cycle between *Financial risk* and *Internal risk*.

consequence of the construction, not an assumption of DynaDex. Node values are *levels* (low, medium, high), so a negative weight means that a higher level of the source lowers the target, and the risk factors have negative weights on the two effects. The figure also gives importance weights of some nodes; they were ignored in the main variant and used, in a second variant, as multipliers of the arcs into the two effect nodes.

The analysis concentrated on the risk cycle. In its core (the two risk nodes, with the five financial-risk inputs merged into one and the two internal-risk inputs), the cycle is a positive loop, and in 8 of the 27 input combinations, in which the inputs roughly balance to a medium level, it is bistable: under the sequential schedule there are two fixed points with basins 3 and 6, and the synchronous schedule adds a cycle of period 2 in which the two risks alternate. In the full model the input space is too large to enumerate (3^{24} combinations), so named scenarios were analysed, each over all $3^8 = 6561$ seeds of the aggregates:

- With all inputs at medium, there are two stable states under the sequential schedule: all aggregates medium (basin 2/3), and financial and internal risk both high (basin 1/3). The synchronous schedule again adds a cycle of period 2 in which the two risks alternate between medium and high.
- With all inputs high, all inputs low, or with technical and delivery risk set low or high, the risk loop is driven to a single state, and every seed reaches the same fixed point.
- With only *Revenue* high (other inputs medium), the two stable risk states lead to different evaluations: *Financial Effect* is high if the risks are medium, but only medium if the risks are high. The evaluation of the offer thus depends on the history of the risk assessment, not only on the inputs. With high revenue and high net cash flow, the financial effect is high in both states.
- In the variant with importance multipliers, the weight of the risk on the financial effect is relatively smaller, and with only *Revenue* high the financial effect is medium in both risk states; the dependence on the history disappears.

WINGS computes the total influence of each factor in closed form, as the convergent series $T = S + S^2 + \cdots = S(I - S)^{-1}$ of the scaled matrix S of strengths and direct influences [3], [40]. This yields a single result for every input and cannot show the two stable risk states. In DynaDex, the positive risk loop appears as bistability, and its consequence for the evaluation of an alternative can be shown explicitly, together with its sensitivity to the modelling choices (here, the use of importance weights).

5.2.7 Findings

1. *Fixed points remain invariant at scale.* The Game of Life confirms Proposition 2 on state spaces of up to 2^{16} states under both schedules, while cycles, transients and basins change drastically.
2. *Classical feedback effects are reproduced qualitatively.* The cobweb oscillation, the boom-and-bust cycle caused by ignoring the supply line, path dependence of trust, and the coherence shift in choice all appear in small DynaDex models with tables of a few rows.
3. *The timing assumption must match the story.* In all four small models the sequential schedule, in which one sweep is one period, gives the intended behaviour. The synchronous schedule adds delays that the story does not contain, and with them additional or longer cycles (doubled cobweb periods, cycles in the stable cases of the supply line, spurious trust cycles), and it can even reverse which of two outcomes is more likely (trust under regulation and shocks).

4. *Coarse scales can create oscillations.* If an equilibrium lies between two scale values, the model has no fixed point and must oscillate around it. Such granularity oscillations should be reported as "between these values" rather than interpreted as cycles of the modelled system.

5. *Path dependence is a result, not a failure.* Several models have more than one stable state for the same inputs (trust, the coherence model, the WINGS risk loop). The value of an alternative then depends on its history or on the decision maker's prior, and DynaDex makes this visible: as basin sizes, as the dependence of an output on the stable state reached (WINGS), or as the split between decisive and prior-preserving profiles (coherence model).

6. *Reconstructing a published cyclic model is feasible but involves choices.* Continuous influence weights must be compiled into tables, and the outcome can depend on modelling details that the source leaves open (the importance weights in WINGS). Such choices should be stated and their effect analysed.

5.3 Extended DEX models

The models of this section start from DEX models: from an existing, published DEX model (Section 5.3.1), and from DEX models that were extended with cycles directly in DEX software using the stock convention of Section 3.8 (Sections 5.3.2–5.3.4). They are the closest to the intended use of DynaDex: a decision maker or analyst starts from a familiar hierarchical model and adds the feedback that the hierarchy cannot express. In all of them, one sequential sweep represents one period (a season, a year), and the sequential schedule is the one that matches the story; the synchronous results are reported for comparison where they were computed.

5.3.1 Models derived from the Grignon model

The Grignon model is a DEX model for the economic and ecological assessment of cropping systems with genetically modified maize [8]. It evaluates a cropping system at some time point within one season: an ECONOMY subtree (costs, product value, yields) and an ECOLOGY subtree are aggregated into an overall assessment. Many of its attributes describe quantities that carry over from one season to the next, such as the weed seed bank, the soil structure or the regional adoption of a crop, and the assessment of one season therefore influences the inputs of the next. These feedbacks are outside the scope of the DEX model.

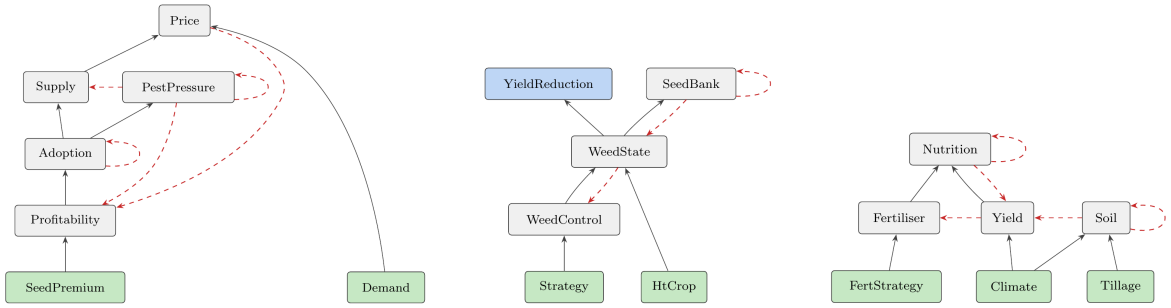
Three small dynamic models were derived from it, each keeping only the attributes that matter for one feedback mechanism (Figure 5.9). The decision tables are our own and qualitative: the aim is to show the kind of behaviour that the feedback produces, not to calibrate the model.

- *G1, regional adoption of Bt maize.* Inputs: the seed premium of Bt maize (low, high) and the demand (weak, normal, strong). The profitability of Bt maize depends on last season's price and pest pressure and on the premium; adoption moves one step per season towards profitability; adoption raises the supply, which lowers the price, and adoption lowers the pest pressure, which reduces the advantage of Bt maize. Both loops through adoption are negative.

- *G2, weeds.* Inputs: the weed control strategy (never treat, a standard single treatment, or a treatment that reacts to last season's weeds) and the crop (conventional or herbicide-tolerant). The weed state depends on the seed bank and on the control; the seed bank grows when weeds are abundant and shrinks when they are rare; the weed state determines the yield reduction.

- *G3, soil and nutrition.* Inputs: the fertilisation strategy (none, moderate, or yield-chasing, i.e. more fertiliser after a poor season), tillage (no-till, plough) and climate (dry, wet). The

yield depends on the nutrient status, the soil structure and the climate; the nutrient status is increased by fertiliser and decreased by the uptake of the crop; ploughing loosens the soil, and no-till in a wet season compacts it.



(a) (b) (c)

Figure 5.9: Dynamic models derived from the Grignon DEX model: (a) G1, regional adoption of Bt maize; (b) G2, weeds; (c) G3, soil and nutrition.

The results are given in Table 5.7. The three models show three different kinds of dynamics:

- *G1 oscillates.* Under the sequential schedule every input combination has a single attractor, reached from all 243 seeds, so the outcome does not depend on the initial state. But in five of the six combinations it is a cycle of two or three seasons: adoption rises, the price falls and the pests decline, Bt maize loses its advantage, adoption falls, and the cycle repeats. Only with a low seed premium and strong demand does the region settle at medium adoption. Under the synchronous schedule the cycles are mostly 7 or 8 seasons long, an effect of the additional delays.

- *G2 is path dependent.* Without treatment, a clean field stays clean and an infested field stays infested: two stable states, with the infested one attracting two thirds of the seeds. A standard treatment of a conventional crop keeps the seed bank at whatever level it had (three stable states), while a herbicide-tolerant crop empties it from most initial states. A treatment that reacts to last season's weeds produces cycles: strong treatment, few weeds, no treatment, many weeds, and so on (period 2 for the conventional and period 4 for the tolerant crop). This is the same delayed negative feedback as in the cobweb model (Section 5.2.2).

- *G3 remembers.* Up to six stable states exist for the same inputs: the soil structure and the nutrient status are carried over, so that, for example, a compacted soil under no-till in a dry climate stays compacted. Some stable states have an *excess* of nutrients, which the yield cannot remove. With ploughing in a wet climate, moderate and yield-chasing fertilisation produce yield cycles (periods 2 and 3 under the sequential schedule): a high yield depletes the nutrients, the next yield is poor, and the nutrients recover.

In all three models the fixed points are the same under both schedules, in accordance with Proposition 2, while the cycles and transients differ.

For decision support, these models add information that the static Grignon model cannot provide: whether a cropping strategy leads to a steady state at all (G1), whether its outcome depends on the history of the field (G2, G3), and whether it creates a self-sustaining cycle of treatment and infestation (G2). Connecting the stable states back to the ECONOMY and ECOLOGY assessments of the original model, and thereby assessing strategies rather than single seasons, was not done in this study.

Table 5.7: Attractors ($p \times n$, periods in sweeps = seasons) and longest transients τ of the Grignon-derived models for every input combination.

Model	inputs	sequential	τ	synchronous	τ
G1 Bt adoption	low, weak	2×1	3	$2 \times 1,\ 7 \times 1$	7
	low, normal	3×1	3	8×1	11
	low, strong	1×1	5	$1 \times 1,\ 7 \times 1$	13
	high, weak	3×1	3	8×1	7
	high, normal	2×1	3	$2 \times 1,\ 8 \times 1$	11
	high, strong	3×1	3	7×1	11
G2 weeds	never, conv.	1×2	3	1×2	7
	never, tolerant	1×2	3	1×2	7
	standard, conv.	1×3	2	1×3	5
	standard, tolerant	1×2	2	1×2	5
	reactive, conv.	$1 \times 2,\ 2 \times 1$	3	$1 \times 2,\ 4 \times 2$	7
	reactive, tolerant	$1 \times 1,\ 4 \times 1$	2	$1 \times 1,\ 3 \times 1$	15
G3 soil	none, no-till, dry	1×3	5	1×3	6
	none, no-till, wet	1×2	4	1×2	6
	none, plough, dry	1×1	5	1×1	6
	none, plough, wet	1×1	4	1×1	6
	moderate, no-till, dry	1×6	3	1×6	4
	moderate, no-till, wet	1×3	3	1×3	5
	moderate, plough, dry	1×3	3	1×3	4
	moderate, plough, wet	$1 \times 1,\ 2 \times 1$	3	$1 \times 1,\ 5 \times 1$	4
	chasing, no-till, dry	1×6	4	1×6	6
	chasing, no-till, wet	1×3	4	1×3	7
	chasing, plough, dry	1×3	3	1×3	5
	chasing, plough, wet	$1 \times 1,\ 3 \times 1$	4	1×1	10

Inputs: G1 (seed premium, demand); G2 (strategy, crop); G3 (fertilisation, tillage, climate).

5.3.2 Employee2: assessment of academic staff

Employee2 is a DEX model for the assessment of academic employees, with the three main criteria *Science*, *Education* and *Contribution to society*, which was extended with eight stocks (Figure 5.10). Its feedback expresses the idea that good results enable further results: for example, papers and citations support new scientific projects, projects and papers support courseware, and courseware supports good student evaluations. Each stock carries the value of an aggregate from one year to the next. The model has two inputs, *Supervision* and *Memberships*, and 12 aggregates; the stocks have three or five values, and the stock space has 30 375 states. The full state space, including the aggregates, has about $5.8 \cdot 10^{11}$ states and is out of reach, so the model was analysed with the stock-map reduction (Section 4.6) under the sequential schedule.

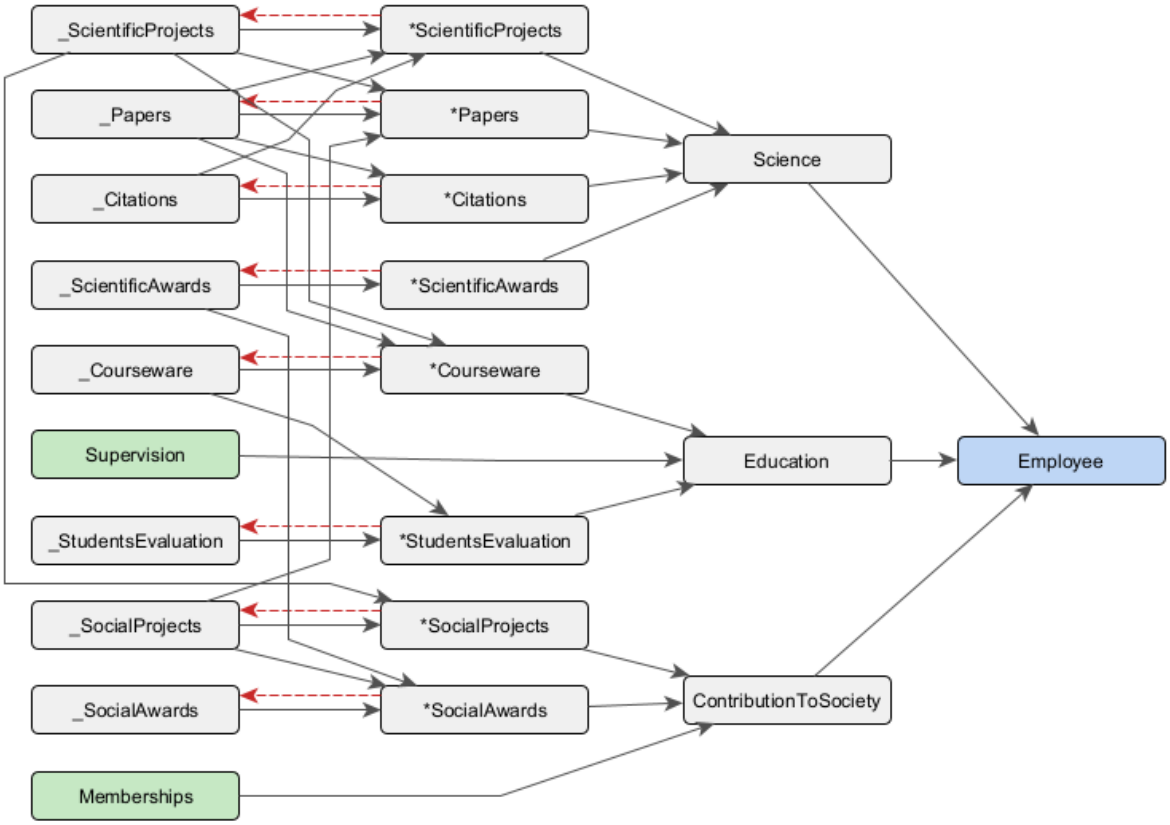


Figure 5.10: Employee2: a DEX model for the assessment of academic employees, extended with stocks. Each aggregate ∗`X` feeds its stock `_X`, which is read in the next sweep by ∗`X` itself and by other aggregates.

The results are the following:

- The model has 1286 attractors, all of them fixed points, reached within at most 10 years. There are no oscillations: all feedback in the model is reinforcing.
- The stock map does not depend on the inputs, because no stock reads *Supervision* or *Memberships*. The inputs affect only the final assessment of the stable states, not the states themselves.
- The largest basins belong to a *virtuous circle*: three stable states in which every stock is at its highest value, and *Employee* is very high. Together they attract 36.9% of all stock states. They differ only in *Scientific awards*, which reads only its own stock and therefore

keeps its initial value forever. The next largest basins (three times 810 states) belong to a *vicious circle*, in which science and education are at their lowest values while social projects and awards remain high, and *Employee* is very low.

- *Employee* never settles at the value *high*. Depending on the inputs, 30–38% of the stock states end with *very low*, 10–14% with *low*, 11–23% with *medium* and 36.9% with *very high*. Better inputs move weight from *very low* to *medium*, but the share of *very high* is fixed by the dynamics.

The bimodal outcome is a typical consequence of reinforcing feedback: the model polarises into excellent and poor employees, and the intermediate assessments of the static DEX model become rare. Whether this is a realistic picture or an artefact of tables that reinforce too strongly is a question for domain experts; the analysis shows clearly where the dynamics of the model lead. Adding negative feedback, for example limited time shared between science and education, was considered but not done.

5.3.3 Project2: assessment of projects

Project2 is a DEX model for the assessment of (hypothetical) projects by their *Benefits*, *Costs* and *Risks*. The costs are assessed from organisational costs (management, technical and academic support) and other resources (time, knowledge, infrastructure). In the extended model these six resources are stocks that support each other: for example, knowledge depends on management, technical and academic support and on time (Figure 5.11). Benefits and risks are inputs with four values; the stocks have three values, and the stock space has 729 states.

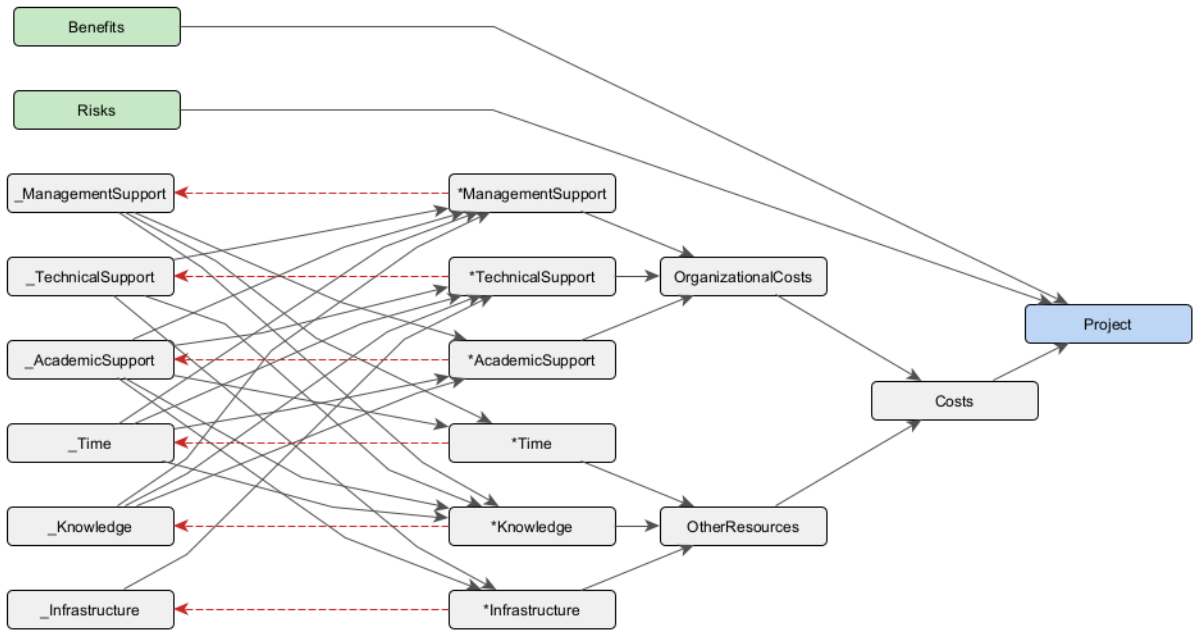


Figure 5.11: Project2: a DEX model for the assessment of projects, with six mutually supporting resources as stocks.

The dynamics are simple and strongly polarised: there are exactly three attractors, all fixed points, in which all six resources are *low* (basin 54.9%), all *medium* (38.0%) or all *high* (7.1%), reached within at most 7 sweeps. As in Employee2, the stocks do not read the inputs, so the three stable states are the same for every combination of benefits and risks; the inputs determine how they are rated. Table 5.8 gives the rating of the project in the three states. In all but one

input combination the rating depends on the state, usually by one level and in one case by two: for example, a project with excellent benefits and excellent risks is rated *good* if its resources settle low or medium, but *excellent* if they settle high. Since the low state has the largest basin, the most likely long-term rating is the lowest of the three.

Table 5.8: Project2: rating of the project in the three stable states (all resources low / medium / high) for every combination of benefits and risks. U = unacceptable, A = acceptable, G = good, E = excellent.

	Risks			
Benefits	U	A	G	E
U	U/U/U	U/U/A	U/U/A	U/U/A
A	U/U/A	U/A/A	U/A/A	A/A/G
G	U/A/A	A/A/G	A/A/G	A/A/G
E	U/A/G	A/G/G	A/G/G	G/G/E

5.3.4 Employee2 with lags

In Employee2 every stock delays its arc by exactly one year. In reality some effects take longer: citations follow papers with a delay, and awards follow achievements. Employee2Lags is the same DEX model in which six stock arcs were given longer lags, using the `lag=`n convention of Section 3.8: citations on citations (2 years), papers on courseware (3), scientific projects on social projects (3), social awards on social awards (2), social projects on social awards (3), and scientific awards on social awards (2). The importer adds 15 delay cells, so the model has 37 nodes, and the reduced state space (stocks and cells) has about $9.3 \cdot 10^{12}$ elements. It was therefore analysed from the 30 375 settled starts (Section 4.6), in which every delay cell starts with the value of its stock.

The results confirm the general properties of delays found with the synthetic models (Section 5.1.5):

- With the lags switched off, the analysis reproduces Employee2 exactly: 1286 fixed points and a longest transient of 10 years.
- With the lags, the model has 1600 attractors reachable from settled starts: the same 1286 fixed points and 314 new cycles of period 2. The longest transient grows to 12 years, and the six largest basins are unchanged. The result again does not depend on the inputs.

The lags thus leave the stable states of the model unchanged, as Proposition 2 requires, but they introduce oscillations into a model whose feedback is entirely reinforcing: a performance indicator that reacts to its own value of several years ago can overshoot and alternate between two levels. For the assessment of an employee this means that, with realistic delays, some combinations of results do not lead to a stable assessment at all, but to an assessment that alternates from year to year.

5.3.5 Findings

1. *Existing DEX models can be extended with cycles with little effort.* The stock convention allows cycles to be added in standard DEX software, and the resulting models remain valid DEX models. The stock-map reduction makes their analysis feasible even when the full state space is far too large.
2. *Reinforcing feedback polarises the assessment.* In Employee2 and Project2, the attractors concentrate in a few extreme states, and intermediate assessments become rare. In

Employee2, one of the five values of the overall assessment is never reached in the long run.

3. *Stable states and their basins are new decision-relevant information.* The static DEX assessment of an alternative is replaced by a set of possible long-term assessments, weighted by their basins. Inputs that are not read by the stocks change the rating of the stable states but not the states themselves.

4. *Delays and delayed reactions create cycles.* Reactive weed control (G2), yield-chasing fertilisation (G3), and lags in Employee2 produce oscillations, even where all feedback is reinforcing (Employee2Lags). Fixed points are not affected.

5. *Derived models need their own tables.* The Grignon-derived models required new decision tables for the dynamic part; the tables of the original model describe one season and do not specify how a state carries over. Making such transitions explicit is part of the modelling effort that DynaDex adds.

5.4 Overview of the models

Table 5.9 summarises all models analysed in this study. *Free* is the number of nodes in the analysed state (aggregates; for models analysed with the stock map, stocks and delay cells), and *states* the size of the analysed state space without the schedule state. The columns *attractors*, *max p* and *max* τ give the range of the number of attractors per instance family and the largest period and transient (in sweeps) over all input combinations and schedules. Every result was obtained twice, by the Python reference and the Oxygene implementation, and all rows agreed.

Taken together, the models cover the range from single nodes to models with millions of states, and from purely synthetic structures to extended real DEX models. Three observations hold across all of them. First, in every comparison of schedules or delays made in the 567 analysed instance families, the fixed points were the same. Second, limit cycles were found in almost every model class, including those with only reinforcing feedback once delays were added; oscillation is a normal outcome of a cyclic DEX model, not an anomaly. Third, in the decision-oriented models the most informative results were not single values but the structure of the attractors: how many stable states there are, how large their basins are, and how the assessment of an alternative differs between them.

Table 5.9: Overview of the analysed models. Schedules: S synchronous, Q sequential, M stock map (sequential), T settled starts (sequential). *Rows*: number of analysed instance families (model variant, schedule and input combination).

Section	Model	rows	free	states	sched.	attractors	max p	max τ
5.1.2	Double loops, AndNot	90	2–11	≤ 2048	S	1–14	6	30
	Double loops, AndB	90	2–11	≤ 2048	S	1–6	6	11
5.1.3	Toggle switch	2	2	4	S, Q	2–3	2	1
	Repressilator(2–6)	10	2–6	≤ 64	S, Q	1–14	10	1
	Greenberg–Hastings ring(3–7)	10	3–7	≤ 2187	S, Q	1–3	7	3
	Integrate-and-fire	4	1	4	S	1–3	4	2
	Half-centre oscillator	4	2	9	S, Q	1–2	4	3
5.1.4	Accumulator, direct	44	2–16	$\leq 65\,536$	S	1–128	65 536	0
	Accumulator, circuit	15	5–11	≤ 2048	Q	1–8	16	2
5.1.5	DelayNot, DelayCopy	14	2–5	≤ 32	S, Q	1–6	8	1
	DoubleLoop32 with delays	12	4–7	≤ 128	S, Q	1–3	2	12
5.2.1	Game of Life	7	9–20	$\leq 2^{20}$	S, Q	39–461	10	21
5.2.2	Cobweb	18	2	9	S, Q	1–3	4	2
5.2.3	Supply line	12	3	27	S, Q	1–2	8	7
5.2.4	Trust	8	2	9	S, Q	1–5	2	4
5.2.5	Coherence-based choice	54	6	729	S, Q	1–9	1	10
5.2.6	WINGS risk loop	54	2	9	S, Q	1–3	2	4
	WINGS full (two variants)	28	8	6561	S, Q	1–3	2	4
5.3.1	G1 Bt adoption	12	5	243	S, Q	1–2	8	13
	G2 weeds	12	4	108	S, Q	2–4	4	15
	G3 soil	24	4	72	S, Q	1–6	5	10
5.3.2	Employee2	9	8	30 375	M	1286	1	10
5.3.3	Project2	16	6	729	M	3	1	7
5.3.4	Employee2Lags	9	23	$9.3 \cdot 10^{12}$	T	1600	2	12

6. Conclusions

This report introduced DynaDex, an extension of the DEX method in which the structure of a model may contain cycles. The study defined DynaDex models formally, developed algorithms for their evaluation and analysis, implemented them in experimental open-source software, and applied them to synthetic models, to models reconstructed from the literature, and to DEX models extended with feedback. This section summarises what the approach combines, what was found, and what it contributes to decision modelling, decision analysis and decision support.

What DynaDex combines, and why. DynaDex combines two traditions. From DEX it takes qualitative attributes with small, meaningful value scales, decision rules in the form of decision tables, and the evaluation of decision alternatives as the main task. These properties make DEX models transparent and easy to discuss with decision makers, but they come with a restriction: a hierarchy cannot express feedback. Real decisions, however, may involve feedback: the result of an evaluation changes the conditions of the next one, an assessment of risk influences the measures that change the risk, and a good reputation attracts the resources that sustain it. From the theory of discrete dynamical systems, in particular multi-valued logical networks, discrete iterations and automata, DynaDex takes the concepts and results needed to give cyclic models a precise meaning: schedules, trajectories, attractors and basins. The combination keeps what makes DEX useful and adds what feedback requires. The multi-criteria methods that already allow feedback, such as ANP, DEMATEL and WINGS, take a different route: they are numeric and treat loops algebraically, in closed form. DynaDex is, to our knowledge, the first approach that combines qualitative, rule-based multi-criteria evaluation with the dynamics of cyclic models.

What was adopted from other disciplines. Most of the theory used in this report was developed elsewhere. A DynaDex model is a multi-valued logical network with a decision-analytic interpretation, so the dynamics of DynaDex models are those studied for decades in systems biology, the theory of automata networks and discrete iterations. In particular, DynaDex adopts Robert's theorem on acyclic networks, which identifies standard DEX as the acyclic special case; the invariance of fixed points under block-sequential schedules and the view of schedules as labelled update digraphs; Tarski's fixed-point theorem and Kleene iteration for monotone models; the feedback-circuit rules of Thomas and their proofs; the SCC-based decomposition for attractor detection; standard cycle-finding algorithms; the Mealy and Moore machines for characterising computational power; the well-formedness condition for cycles from synchronous languages; and the stocks and delays of system dynamics (Section 2). The contribution of this study lies in bringing these results together for decision models, adapting them where DEX requires it, and showing what they mean for decision analysis.

Main contributions.

1. *A formal framework that conservatively extends DEX* (Section 3). It defines DynaDex models, their evaluation and its result, and it makes explicit what an acyclic structure leaves implicit: the schedule, the initial state of the cyclic part, and the form of the

result. Node types are redefined through the condensation of the model graph, delays are introduced as an optional model element, and every DynaDex model has a normal form consisting of a DEX evaluation and a set of registers. For acyclic models, everything reduces to standard DEX.

2. *A new kind of evaluation result for decision analysis.* Instead of a single value per attribute, the evaluation of an alternative yields an attractor: a fixed point or a limit cycle, together with its basin, i.e. the share of initial states that lead to it. Oscillating attributes can be summarised by occupancy distributions, their supports or summary values, which form a hierarchy of increasingly coarse results.

3. *Exact and practical algorithms* (Section 4). Every evaluation terminates with an exactly identified attractor, without convergence tolerances. All attractors can be found exhaustively, compositionally along the strongly connected components (extended here to multi-valued scales, block-sequential schedules and delays), or, for models built in DEX software, by the stock-map reduction, which made models with state spaces of about 10^{11} to 10^{13} states tractable.

4. *A way to build cyclic models with existing DEX software.* The stock convention (Section 3.8) lets modellers add feedback, and lags, to ordinary DEX models in DEXi or DEXiWin, so that the familiar modelling environment can be used.

5. *A characterisation of the computational nature of DynaDex.* With clamped inputs, a DynaDex model is a finite deterministic dynamical system; with input sequences, it is exactly as expressive as finite Mealy and Moore machines, of which it is a factored, readable representation. Turing equivalence requires unbounded families of models.

6. *An experimental study and open-source software.* The study analysed 567 instance families of synthetic, reconstructed and extended DEX models, each computed independently by two implementations, and the software is publicly available.

Main findings. Five findings hold across the models of Section 5:

- *Fixed points are structural, everything else depends on timing*: the fixed points of a model never changed with the schedule or with delays, whereas limit cycles, transients and basins did, sometimes drastically. Statements about the long-term evaluation of an alternative are therefore robust if they concern fixed points, and must be accompanied by the timing assumption otherwise.

- *Timing is a modelling decision*: in the decision-oriented models, the schedule in which one sweep corresponds to one period of the modelled process gave the intended behaviour, whereas the synchronous schedule added delays that the process does not contain, together with spurious cycles, and could even reverse which outcome is more likely.

- *Oscillations are common and not always meaningful*: already small tables oscillate, and when an equilibrium falls between two values of a coarse scale, the model must oscillate around it; such granularity oscillations should be reported as an intermediate result rather than as a cycle of the modelled system.

- *Multiple stable states and path dependence are decision-relevant information*: when a model has several attractors for the same inputs, the long-term evaluation of an alternative depends on its history, which a static model cannot show, and reinforcing feedback tends to polarise the assessments into a few extreme states.

- *Delays create cycles but do not change fixed points*: delayed reactions, such as reactive control or lagged effects, produce oscillations even where all feedback is reinforcing. These findings matter because each of them would remain hidden in a hierarchical model, or in a method that computes the effect of feedback in closed form.

Contribution to decision modelling, analysis and support. For *decision modelling*, DynaDex allows feedback, accumulation and delays to be modelled explicitly, with the same qualitative attributes and decision tables as in DEX. It also makes a modelling task visible that hierarchical models hide: the transitions from one period to the next must be specified, and the timing of the model must be chosen to match the process being modelled. For *decision analysis*, the evaluation of an alternative becomes richer: instead of one value, the analyst obtains the possible long-term assessments, their likelihood in terms of basins, the time needed to reach them, and their dependence on history and timing. This supports new kinds of questions: which alternatives lead to a single stable outcome and which to several; which outcomes are robust with respect to timing; which inputs remove an undesired cycle or a vicious circle. For *decision support*, the trajectories, attractors and loop structure give an explanation of the dynamics that decision makers can follow, in the familiar terms of reinforcing and balancing loops, and the software makes the analysis practical for models of realistic size built in standard DEX tools.

Limitations. This study was exploratory. The decision tables of the reconstructed and derived models were designed by us to capture the qualitative mechanisms described in the sources, and the extended DEX models were built for this study; none of the models has yet been used with real decision makers. Evaluation was restricted to deterministic schedules and to full standard decision tables. Exhaustive analysis is limited by the size of the state space, which the compositional search and the stock-map reduction only partly overcome. Several properties were stated with proof sketches, and some observations, such as the dependence of periods on the total delay around cycles, remain conjectures.

Further research. Further research will proceed in several directions. On the theoretical side, the proof sketches should be completed, structural conditions under which every deterministic sequential schedule converges should be sought, and the conjecture on periods and total cycle delays should be settled. On the methodological side, DynaDex should be extended with the richer decision-table outcomes of DEX, i.e. value intervals, sets and distributions, possibly with three-valued or interval semantics for cycles that do not settle; with time-varying inputs, which would allow the evaluation of policies and scenario streams rather than static alternatives; and with guidelines for eliciting transition tables and for choosing the timing of a model. On the computational side, symbolic and SAT-based methods would extend the analysis to larger models, and the algorithms should be integrated into DEX software, so that the analysis of feedback becomes part of ordinary DEX modelling. Most importantly, DynaDex needs to be tested in real decision problems, with decision makers, to assess whether the additional information it provides improves the understanding and quality of decisions. These steps, together with the results of this report, will form the basis for subsequent conference and journal publications.

Acknowledgement

This work has been financially supported by the Slovenian Research and Innovation Agency for the programme Knowledge Technologies, research core funding No. P2-0103. The author acknowledges the use of Claude (Anthropic) as a research and writing assistant throughout this study.

References

[1] J. Aracena, E. Goles, A. Moreira, and L. Salinas, "On the robustness of update schedules in Boolean networks," *BioSystems*, vol. 97, no. 1, pp. 1–8, 2009, V (Phase 5). DOI: 10.1016/j.biosystems.2009.03.006

[2] C. A. Bana e Costa, J.-M. De Corte, and J.-C. Vansnick, "Macbeth," *International Journal of Information Technology & Decision Making*, vol. 11, no. 02, pp. 359–387, 2012. DOI: 10.1142/S021962201250017X

[3] D. Banaś and J. Michnik, "Evaluation of the impact of strategic offers on the financial and strategic health of the company—a soft system dynamics approach," *Mathematics*, vol. 7, no. 2, p. 208, 2019. DOI: 10.3390/math7020208

[4] H. J. Blok and B. Bergersen, "Synchronous versus asynchronous updating in the "game of Life"," *Physical Review E*, vol. 59, no. 4, pp. 3876–3879, 1999. DOI: 10.1103/PhysRevE.59.3876

[5] M. Bohanec, "DEXi: Program for Multi-Attribute Decision Making, User's Manual, Version 5.04," Jožef Stefan Institute, Ljubljana, IJS Report DP-13100, 2020. [Online]. Available: https://kt.ijs.si/MarkoBohanec/pub/DEXiManual504.pdf

[6] M. Bohanec, "DEX (Decision EXpert): A qualitative hierarchical multi-criteria method," in *Multiple Criteria Decision Making: Techniques, Analysis and Applications*, A. J. Kulkarni, Ed. Singapore: Springer Nature Singapore, 2022, pp. 39–78, ISBN: 978-981-16-7414-3. DOI: 10.1007/978-981-16-7414-3_3

[7] M. Bohanec, "DEXiWin: DEX decision modeling software, User's manual, Version 1.2," Jožef Stefan Institute, Ljubljana, IJS Report IJS DP-14747, 2024. [Online]. Available: https://kt.ijs.si/MarkoBohanec/pub/2024_DP14747_DEXiWin.pdf

[8] M. Bohanec et al., "A qualitative multi-attribute model for economic and ecological assessment of genetically modified crops," *Ecological Modelling*, vol. 215, pp. 247–261, 2008. DOI: 10.1016/j.ecolmodel.2008.02.016

[9] R. P. Brent, "An improved Monte Carlo factorization algorithm," *BIT*, vol. 20, pp. 176–184, 1980.

[10] O. Cappé, É. Moulines, and T. Rydén, *Inference in Hidden Markov Models*. Springer, 2005, ISBN: 978-0-387-40264-2. DOI: 10.1007/0-387-28982-8

[11] C.-Y. Chen and J.-J. Huang, "Integrating dynamic bayesian networks and analytic hierarchy process for time-dependent multi-criteria decision-making," *Mathematics*, vol. 11, no. 10, p. 2362, 2023. DOI: 10.3390/math11102362

[12] M. Cook, "Universality in elementary cellular automata," *Complex Systems*, vol. 15, no. 1, pp. 1–40, 2004. DOI: 10.25088/ComplexSystems.15.1.1

[13] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 4th. Cambridge, MA: MIT Press, 2022, ISBN: 978-0-262-04630-5.

[14] S. Corrente, S. Greco, and R. Słowiński, "Multiple criteria hierarchy process in robust ordinal regression," *Decision Support Systems*, vol. 53, no. 3, pp. 660–674, 2012, ISSN: 0167-9236. DOI: https://doi.org/10.1016/j.dss.2012.03.004 [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167923612000905

[15] T. Dean and K. Kanazawa, "A model for reasoning about persistence and causation," *Computational Intelligence*, vol. 5, no. 3, pp. 142–150, 1989.

[16] E. Dubrova and M. Teslenko, "A SAT-based algorithm for finding attractors in synchronous Boolean networks," *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, vol. 8, no. 5, pp. 1393–1399, 2011. DOI: 10.1109/TCBB.2010.20

[17] M. B. Elowitz and S. Leibler, "A synthetic oscillatory network of transcriptional regulators," *Nature*, vol. 403, no. 6767, pp. 335–338, 2000. DOI: 10.1038/35002125

[18] M. Ezekiel, "The cobweb theorem," *The Quarterly Journal of Economics*, vol. 52, no. 2, pp. 255–280, 1938. DOI: 10.2307/1881734

[19] M. Fitting, "A Kripke–Kleene semantics for logic programs," *Journal of Logic Programming*, vol. 2, no. 4, pp. 295–312, 1985. DOI: 10.1016/S0743-1066(85)80005-4

[20] M. Gadouleau, *Robert's theorem and graphs on complete lattices*, arXiv:2309.11363, 2023.

[21] E. R. Gansner and S. C. North, "An open graph visualization system and its applications to software engineering," *Software: Practice and Experience*, vol. 30, no. 11, pp. 1203–1233, 2000. DOI: 10.1002/1097-024X(200009)30:11<1203::AID-SPE338>3.0.CO;2-N

[22] M. Gardner, "Mathematical games: The fantastic combinations of John Conway's new solitaire game "life"," *Scientific American*, vol. 223, no. 4, pp. 120–123, 1970.

[23] T. S. Gardner, C. R. Cantor, and J. J. Collins, "Construction of a genetic toggle switch in Escherichia coli," *Nature*, vol. 403, no. 6767, pp. 339–342, 2000. DOI: 10.1038/35002131

[24] A. Glöckner and T. Betsch, "Modeling option and strategy choices with connectionist networks: Towards an integrative model of automatic and deliberate decision making," *Judgment and Decision Making*, vol. 3, no. 3, pp. 215–228, 2008.

[25] E. Goles and S. Martínez, *Neural and Automata Networks: Dynamical Behavior and Applications*. Dordrecht: Kluwer Academic Publishers, 1990, ISBN: 978-0-7923-0632-0.

[26] S. Greco, M. Ehrgott, and J. Figueira, Eds., *Multi Criteria Decision Analysis: State of the Art Surveys*. New York: Springer, 2016. DOI: 10.1007/978-1-4939-3094-4

[27] J. M. Greenberg and S. P. Hastings, "Spatial patterns for discrete models of diffusion in excitable media," *SIAM Journal on Applied Mathematics*, vol. 34, no. 3, pp. 515–523, 1978. DOI: 10.1137/0134040

[28] N. Halbwachs, P. Caspi, P. Raymond, and D. Pilaud, "The synchronous dataflow programming language LUSTRE," *Proceedings of the IEEE*, vol. 79, no. 9, pp. 1305–1320, 1991.

[29] K. J. Holyoak and D. Simon, "Bidirectional reasoning in decision making by constraint satisfaction," *Journal of Experimental Psychology: General*, vol. 128, no. 1, pp. 3–31, 1999. DOI: 10.1037/0096-3445.128.1.3

[30] C.-L. Hwang and K. Yoon, *Multiple Attribute Decision Making: Methods and Applications*. New York: Springer-Verlag, 1981, The seminal book introducing the TOPSIS method. DOI: 10.1007/978-3-642-48318-9

[31] S. A. Kauffman, "Metabolic stability and epigenesis in randomly constructed genetic nets," *Journal of Theoretical Biology*, vol. 22, no. 3, pp. 437–467, 1969. DOI: 10.1016/0022-5193(69)90015-0

[32] H. Klarner and H. Siebert, "Approximating attractors of Boolean networks by iterative CTL model checking," *Frontiers in Bioengineering and Biotechnology*, vol. 3, p. 130, 2015. DOI: 10.3389/fbioe.2015.00130

[33] B. Kosko, "Fuzzy cognitive maps," *International Journal of Man-Machine Studies*, vol. 24, no. 1, pp. 65–75, 1986. DOI: 10.1016/S0020-7373(86)80040-2

[34] B. Kuipers, "Qualitative simulation," *Artificial Intelligence*, vol. 29, no. 3, pp. 289–338, 1986.

[35] C. E. Leiserson and J. B. Saxe, "Retiming synchronous circuitry," *Algorithmica*, vol. 6, pp. 5–35, 1991.

[36] S. Malik, "Analysis of cyclic combinational circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 13, no. 7, pp. 950–956, 1994. DOI: 10.1109/43.293952

[37] E. Marder and D. Bucher, "Central pattern generators and the control of rhythmic movements," *Current Biology*, vol. 11, no. 23, R986–R996, 2001. DOI: 10.1016/S0960-9822(01)00581-4

[38] R. T. Marler and J. S. Arora, "The weighted sum method for multi-objective optimization: New insights," *Structural and Multidisciplinary Optimization*, vol. 41, no. 6, pp. 853–862, 2010, A highly cited primary journal review and analysis providing foundational insights into the mechanics, advantages, and drawbacks of WSM. DOI: 10.1007/s00158-009-0460-7

[39] G. H. Mealy, "A method for synthesizing sequential circuits," *Bell System Technical Journal*, vol. 34, no. 5, pp. 1045–1079, 1955.

[40] J. Michnik, "Weighted influence non-linear gauge system (WINGS) – an analysis method for the systems of interrelated components," *European Journal of Operational Research*, vol. 228, no. 3, pp. 536–544, 2013. DOI: 10.1016/j.ejor.2013.02.007

[41] A. Mizera, J. Pang, H. Qu, and Q. Yuan, "A decomposition method for attractor detection in large synchronous Boolean networks," in *Dependable Software Engineering: Theories, Tools, and Applications (SETTA 2017)*, ser. Lecture Notes in Computer Science, vol. 10606, Springer, 2017, pp. 232–249.

[42] E. F. Moore, "Gedanken-experiments on sequential machines," in *Automata Studies*, ser. Annals of Mathematics Studies 34, C. E. Shannon and J. McCarthy, Eds., Princeton, NJ: Princeton University Press, 1956, pp. 129–153.

[43] A. Motzek and R. Möller, "Indirect causes in dynamic Bayesian networks revisited," *Journal of Artificial Intelligence Research*, vol. 59, pp. 1–58, 2017. DOI: 10.1613/jair.5361

[44] T. Murata, "Petri nets: Properties, analysis and applications," *Proceedings of the IEEE*, vol. 77, no. 4, pp. 541–580, 1989.

[45] E. Ostrom, "A behavioral approach to the rational choice theory of collective action: Presidential address, American Political Science Association, 1997," *American Political Science Review*, vol. 92, no. 1, pp. 1–22, 1998. DOI: 10.2307/2585925

[46] E. I. Papageorgiou and J. L. Salmeron, "A review of fuzzy cognitive maps research during the last decade," *IEEE Transactions on Fuzzy Systems*, vol. 21, no. 1, pp. 66–79, 2013. DOI: 10.1109/TFUZZ.2012.2201727

[47] K. Perrot, S. Sené, and L. Tapin, *Creation of fixed points in block-parallel Boolean automata networks*, arXiv:2505.15499, 2025.

[48] W. Reisig, *Understanding Petri Nets: Modeling Techniques, Analysis Methods, Case Studies*. Springer, 2013, ISBN: 978-3-642-33277-7. DOI: 10.1007/978-3-642-33278-4

[49] É. Remy, P. Ruet, and D. Thieffry, "Graphic requirements for multistability and attractive cycles in a Boolean dynamical framework," *Advances in Applied Mathematics*, vol. 41, no. 3, pp. 335–350, 2008, V (Phase 1).

[50] A. Richard, "Negative circuits and sustained oscillations in asynchronous automata networks," *Advances in Applied Mathematics*, vol. 44, no. 4, pp. 378–392, 2010, ISSN: 0196-8858. DOI: 10.1016/j.aam.2009.11.011

[51] A. Richard and J.-P. Comet, "Necessary conditions for multistationarity in discrete dynamical systems," *Discrete Applied Mathematics*, vol. 155, no. 18, pp. 2403–2413, 2007.

[52] F. Robert, *Discrete Iterations: A Metric Study* (Springer Series in Computational Mathematics). Berlin: Springer, 1986, vol. 6, V (Phase 1); translated by J. Rokne.

[53] T. L. Saaty, "How to make a decision: The analytic hierarchy process," *European Journal of Operational Research*, vol. 48, no. 1, pp. 9–26, 1990.

[54] T. L. Saaty, *Decision Making with Dependence and Feedback: The Analytic Network Process*. Pittsburgh, PA: RWS Publications, 1996.

[55] T. L. Saaty, "Fundamentals of the analytic network process — multiple criteria decision making," *International Journal of Production Economics*, vol. 89, no. 3, pp. 251–257, 2004.

[56] T. L. Saaty and L. G. Vargas, *Decision Making with the Analytic Network Process* (International Series in Operations Research & Management Science). New York: Springer, 2006. DOI: 10.1007/0-387-33987-6

[57] B. Schönfisch and A. de Roos, "Synchronous and asynchronous updating in cellular automata," *BioSystems*, vol. 51, no. 3, pp. 123–143, 1999. DOI: 10.1016/S0303-2647(99)00025-8

[58] J. D. Schwab, S. D. Kühlwein, N. Ikonomi, M. Kühl, and H. A. Kestler, "Concepts in Boolean network modeling: What do they all mean?" *Computational and Structural Biotechnology Journal*, vol. 18, pp. 571–582, 2020. DOI: 10.1016/j.csbj.2020.03.001

[59] S.-L. Si, X.-Y. You, H.-C. Liu, and P. Zhang, "Dematel technique: A systematic review of the state-of-the-art literature on methodologies and applications," *Mathematical Problems in Engineering*, vol. 2018, no. 1, p. 3 696 457, 2018. DOI: https://doi.org/10.1155/2018/3696457 [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1155/2018/3696457

[60] J. D. Sterman, "Modeling managerial behavior: Misperceptions of feedback in a dynamic decision making experiment," *Management Science*, vol. 35, no. 3, pp. 321–339, 1989. DOI: 10.1287/mnsc.35.3.321

[61] J. D. Sterman, *Business Dynamics: Systems Thinking and Modeling for a Complex World*. Boston: Irwin/McGraw-Hill, 2000, ISBN: 978-0-07-231135-8.

[62] R. E. Tarjan, "Depth-first search and linear graph algorithms," *SIAM Journal on Computing*, vol. 1, no. 2, pp. 146–160, 1972. DOI: 10.1137/0201010

[63] A. Tarski, "A lattice-theoretical fixpoint theorem and its applications," *Pacific Journal of Mathematics*, vol. 5, no. 2, pp. 285–309, 1955, V (Phase 1). DOI: 10.2140/pjm.1955.5.285

[64] P. Thagard, "Explanatory coherence," *Behavioral and Brain Sciences*, vol. 12, no. 3, pp. 435–467, 1989. DOI: 10.1017/S0140525X00057046

[65] R. Thomas, "Boolean formalization of genetic control circuits," *Journal of Theoretical Biology*, vol. 42, no. 3, pp. 563–585, 1973, V (Phase 1); pages differ between sources (563–585 / 565–583).

[66] R. Thomas and M. Kaufman, "Multistationarity, the basis of cell differentiation and memory. II. Logical analysis of regulatory networks in terms of feedback circuits," *Chaos*, vol. 11, no. 1, pp. 180–195, 2001. DOI: 10.1063/1.1349893

[67] N. Trdin and M. Bohanec, "Extending the multi-criteria decision making method DEX with numeric attributes, value distributions and relational models," *Central European Journal of Operations Research*, vol. 26, no. 1, pp. 1–41, 2018. DOI: 10.1007/s10100-017-0468-9

[68] A. Van Gelder, K. A. Ross, and J. S. Schlipf, "The well-founded semantics for general logic programs," *Journal of the ACM*, vol. 38, no. 3, pp. 620–650, 1991.

[69] Z. Wang and G. P. Rangaiah, *Multi-Criteria Decision-Making: Principles, Methods and Programs*. CRC Press, 2026, ISBN: 9781032853635. [Online]. Available: https://www.routledge.com/Multi-Criteria-Decision-Making-Principles-Methods-and-Programs/Rangaiah-Wang/p/book/9781032853635

[70] S. Wolfram, "Statistical mechanics of cellular automata," *Reviews of Modern Physics*, vol. 55, no. 3, pp. 601–644, 1983. DOI: 10.1103/RevModPhys.55.601

[71] Q. Yuan, A. Mizera, J. Pang, and H. Qu, "A new decomposition-based method for detecting attractors in synchronous Boolean networks," *Science of Computer Programming*, vol. 180, pp. 18–35, 2019.