A Strategic Graph Rewriting Model of Rational Negligence in Financial Markets

Nneka Ene, Maribel Fernández, Bruno Pinaud

A Strategic Graph Rewriting Model of Rational Negligence in Financial Markets

Nneka Ene[1], Maribel Fernández[1] and Bruno Pinaud[2]

Abstract We propose to use strategic port graph rewriting as a visual modelling tool to analyse financial market processes. We illustrate the approach by specifying a basic "rational negligence" model in which investors may choose to trade securities without performing independent evaluations of the underlying assets. We show that our model is correct with respect to the equational model and can be used to simulate simple market behaviours. The model has been implemented within PORGY, a graph-based specification and simulation environment.

1 Introduction

Rational negligence [1] has been identified as a behavioural pattern in financial tradings, where transactions are performed without proper checks in order to maximise benefits and reduce operational costs. For example, in 2008 ratings from credit agencies (later found to be inaccurate) were used to replace costly checks, leading to a financial crisis that the DSGE (Dynamic Stochastic General Equilibrium) models [21] were unable to anticipate. This motivated a quest for more effective and transparent tools in the modelling of capital markets [26].

As an alternative to traditional top-down macro equilibrium models, Agent-Based Models (ABM) have been proposed, which examine behaviour at a micro-level [13]. In this paper we explore an alternative approach: we seek to formalise the rational negligence theory using *graph rewriting*. We provide an example to illustrate the ideas, as a step towards the development of alternative tools for the analysis of markets to complement the current agent-based implementations.

Nneka Ene and Maribel Fernández
King's College London, UK. e-mail: maribel.fernandez@kcl.ac.uk

Bruno Pinaud
University of Bordeaux, France

Rewrite rules are an intuitive and natural way of expressing dynamic, structural changes which are generally more difficult to model in traditional simulation approaches where the structure of the model is usually fixed [8]. Graph rewriting languages are well-suited to the study of the dynamic behaviour of complex systems: their declarative nature and visual aspects facilitate the analysis of the processes of interest producing a shorter distance between mental picture and implementation; they can be used for rapid prototyping, to run system simulations, and, thanks to their formal semantics, also to reason about system properties.

We use *attributed port graphs*, that is, graphs where edges are connected to nodes at specific points called ports, and where attributes are attached to ports, nodes and edges. Attributed port graphs are useful in the development of graph models, due to their support of both topology (via ports and edges) and data (via attributes). To control the rewriting process, we use *strategies* that permit to select which rules to apply and where, including probabilistic rule applications. We present first a basic model of asset trading following a discretised equational model presented in [1], where the probability of asset toxicity, due diligence analysis cost and asset cost are fixed. We then briefly discuss a more general version of the model where stochasticity is introduced by using a probabilistic choice model of logit type [13].

Summary of Contributions.

We provide port graph rewrite rules and strategies that specify basic asset-trading transactions, starting with an auction to select a potential buyer. These rules and strategies model the rational negligence phenomenon [1, 20], whereby investors may choose to trade securities without performing independent evaluations of the underlying assets. The model has been implemented in PORGY[1], an interactive, visual port graph rewriting tool. The graph rewriting approach we advocate produces flexible models that are easy to validate, experiment with and reason about. We illustrate it by showing the correctness of our graph rewrite rules and strategies with respect to the equations defining the rational negligence phenomenon, and using the implemented model to analyse simple market behaviours.

Related Work.

Graph Transformation Systems (GTSs) have been used as a modelling framework in many areas: for example, RuleBENDER[2] is a simulation tool that supports rule-based modelling of biochemical systems [30], Kappa [23] is a rule-based language for modelling protein interaction networks, graph transformation has also been used to outline the semantics of domain specific modelling languages [8].

[1] `http://porgy.labri.fr`

[2] `http://www.rulebender.org`

A basic set of port graph rewrite rules to model rational negligence was presented by Ene [11], focusing on implementation aspects. Here we extend the rules to include an abstract representation of an auction process and we analyse the properties of the model: we prove that the rewrite rules and the strategies we provide correctly simulate the equational model of rational negligence [1].

Previous rational negligence models followed an agent-based approach (see, for example [1, 26]). Test results for our model line up with results form traditional agent-based models (see Section 4 and [11] for a discussion of experimental results). General purpose agent-based simulation tools (see [22] for a survey) support an imperative object-oriented approach to model development. The graph rewriting approach used in this paper is declarative: the program consists of graph transformation rules and a strategy. Languages like Stratego [6, 36], Maude [10, 27] and ELAN [5] support a term rewriting approach with user-defined strategies to control the application of rules. Rascal [32] (and its predecessor ASF+DSF [33]) are closely related, using algebraic specifications as a basis to define programs, with traversal functions to control the application of rules. Tom [3] is an extension of Java with algebraic terms, rule definitions and a strategy language, thus allowing programmers to combine imperative object-oriented programming and strategic term rewriting. The symbolic transformation language symbtrans designed in the context of MEMSALab [4] (where models are defined using partial differential equations) extends Maple^{TM} with conditional rewriting, strategies and pattern-matching modulo associativity and commutativity.

An alternative rule-based approach uses rules to define predicates, as in the logic programming language Prolog and its variants, including in some cases domain-specific constraint solvers or special-purpose languages to handle constraints [17]. The multi-paradigm language Claire [7] combines the imperative, functional and object-oriented styles with rule processing capabilities, including constructs to create new branches in the search-tree and to backtrack if the current branch fails. The language Prholog [9] extends logic programming with strategic conditional transformation rules, combining Prolog with the ρLog calculus [25] to enable strategic programming.

We have chosen to develop our models using port graph rewriting in PORGY [14], since it provides a visual rule-based programming-style, including user-defined strategies. The visual, declarative nature of GTS tools such as PORGY is welcome in the cases where users seek to primarily focus on describing what the system should accomplish, and is especially useful for the analysis of complex systems in interactive environments.

A benchmark analysing the differences between several GTS tools has been developed by Varró et al. [35]. A variety of GTS tools are available: among others we can cite GROOVE [19], a graph-based model checker for object oriented systems; AGG (the Attributed Graph Grammar System) [31], a graph-based language for the transformation of attributed graphs that comes with a visual programming environment; PROGRES (Programmed Graph Rewriting Systems) [29] that offers backtracking and nondeterministic constructs; GrGen (Graph Rewrite Generator) [18] that uses attributed typed multigraphs and includes features such as Java/C code

generation, and GP [28], a graph programming language, where users can define rules and strategy expressions, with support for conditional rewriting. PORGY [14] has been used to model social networks [15] and database design [34, 16], as well as biochemical processes [2], where non-determinism, backtracking, positioning constructs, and probabilistic rule application are key features. A distinctive feature of PORGY is that rewriting derivations are directly available to users via the so-called derivation tree, which provides a visual representation of the dynamics of the system modelled and can be used to plot parameters and generate charts as illustrated in Section 4.

Overview.

We first recall key notions on securitisation and graph rewriting in Section 2. Section 3 describes the proposed approach to the modelling of securitisation, including a short description of rules and associated strategies. Section 4 examines key properties of the model. We finally conclude and briefly outline future plans in Section 5.

2 Background

In this section we recall the main notions of asset trading and port graph rewriting that are needed in the rest of the paper.

2.1 Asset-Backed Securities

Assets [20] represent loans to clients or obligors who make regular installment payments to the originator to clear their debts. In a securitisation, assets are selected, pooled and transferred to a special purpose vehicle (SPV), who funds them by issuing securities. In general, an ABS (*asset-backed security*), or simply asset if there is no ambiguity, is any securitisation issue backed by consumer loans, car loans, etc.

In the core rational negligence model [1], the profit $\mathscr{U}_w$ expected by an agent (e.g., a bank) w from trading an asset depends on whether or not w follows the *negligence rule*, i.e., the rule of not performing independent risk assessment. Let z be a binary variable indicating whether or not the agent is following the negligence rule, then $\mathscr{U}_w$ is a function of z. According to [1], $\mathscr{U}_w(z)$ can be characterised by the following equations, where p is the probability of asset toxicity, Z is the average of all z's in the domain, c is the cost of purchasing an asset (note that the payoff from successfully reselling the asset is normalised to unity), x_w is the cost of performing a complete risk analysis, k is the number of trading partners of the seller bank and $\mathscr{N}_i$ is the set of agents.

- Expected profit for w when following the negligence rule, i.e., when $z(w) = 1$, if w buys an asset and then tries to sell it to w':

$$\mathscr{U}_w(1) =^{def} -p(1-z(w'))c + [1-p(1-z(w'))](1-c) \approx 1-p(1-Z)-c$$

  This is because if the asset is toxic then w will loose c if w' checks, and will have a profit of $1-c$ if w' does not check. Of course w does not know a priori whether w' will or not follow the rule, but it can estimate $z(w')$ as the average of all the values of z in the system, Z. Note that when $p = 0$ the profit is $1-c$ as expected.
- Similarly, the expected profit for w when the rule is not followed, i.e., $z(w) = 0$, is defined by:

$$\mathscr{U}_w(0) =^{def} (1-p)(1-c) - x_w$$

  This is because if the asset is toxic, then w will not buy it (losing only x_w), but if it is not toxic then it will resell it with a profit of $1-c-x_w$. Note that when $p = 1$ the loss is x_w as expected.

So the best response of agent w to a buying request is determined by the value of $\mathscr{U}(1) - \mathscr{U}(0)$. If it is positive, then negligence is better, otherwise diligence is better. Note that

$$\mathscr{U}(1) - \mathscr{U}(0) = p(Z-c) + x_w = p\left(\frac{1}{k}\sum_{j \in \mathscr{N}i} z_j - c\right) + x_w$$

Following [1], in this paper we study the behaviour produced by the trading of one asset since this is sufficient to perform validations against equivalent DSGE analyses. The goal is to study the evolution of the system till *fixed point* (that is, a *stable state*) is reached i.e., in this case, a state such that all potential buyers in the universe of discourse no longer alternate between diligent and negligent behaviour in their handling of the purchase of a particular asset.

2.2 Port Graph Rewriting

A port graph is a graph where nodes have explicit connection points, called ports, and edges are attached to ports. Nodes, ports and edges are labelled by a set of attributes, including a mandatory attribute *Name* that characterises the type of the node, port or edge. Attributes describe properties such as colour, size, etc. In PORGY [14] labels are records, i.e., lists of attribute-value pairs. The values can be concrete (numbers, Booleans, etc.) or abstract (expressions in a term algebra, which may contain variables). For example, the port graph in Figure 1 depicts a toy ABS market universe represented by a community of banks (*B* nodes), one of which owns a tradeable asset (*A*), together with a global environment represented by the nodes *Z*, Change and Auction. The edge between *A* and *B* represents ownership.

Transactions between banks are specified by means of rewrite rules. A *port graph rewrite rule* $L \Rightarrow_C R$ is itself a port graph consisting of two port graphs L and R together with an "arrow" node. Intuitively, the pattern, L, is used to identify subgraphs (redexes) in a given graph which should be replaced by an instance of the right-hand side, R, provided the condition C holds. The arrow node may have ports and edges that connect it to L and R; these edges specify a partial morphism between the ports in L and R, following the single push-out approach [24] to graph rewriting (see [14] for more details). Operationally, the arrow-node edges are used during rewriting to redirect edges that arrive to ports in the redex from outside, ensuring that no edges are left dangling. Table 3 shows the rules used in our model (these will be discussed in the next sections). The arrow-node edges can be optionally displayed in PORGY; when displayed, they are shown in red. In PORGY attribute values can be updated in the right-hand side of a rule by means of an "algorithm tab" (see Table 3).

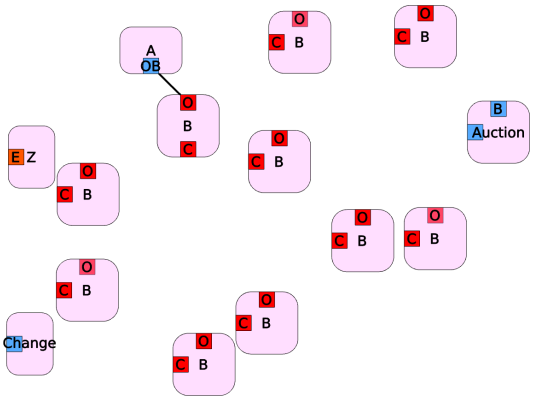


Fig. 1 *Sample Port-graph: Model's Starting Graph*

For a given graph, several different rewriting steps may be possible (due to the intrinsic non-determinism of rewriting). Strategies in rewriting systems are a means of controlling the creation of rewriting steps. A sequence of rewriting steps is called a *derivation*. A *derivation tree* is a collection of derivations with a common root. Intuitively, the derivation tree is a representation of the possible evolutions of the system starting from a given initial state (each derivation provides a trace, which can be used to analyse and reason about the behaviour of system).

PORGY's strategy language allows us to specify not only the rule to be used in a rewriting step, but also the position where the rule should (or should not) be applied. Formally, the rewriting relation is defined on *located graphs*, which are port-graphs with two distinguished subgraphs P (Position subgraph, the focus of rewriting) and Q (Banned subgraph, where rewriting steps are forbidden). The keywords `crtGraph`, `crtPos`, `crtBan` in the strategy language denote, respectively the current graph being rewritten and its Position and Banned subgraphs. For example, the strategy expression `setPos(crtGraph)` sets the position graph as the

full current graph. If T is a rule, then the strategy $one(T)$ randomly selects one possible occurrence of a match of rule T in the current graph G, which *should superpose the position subgraph P but not superpose the banned subgraph Q*. This strategy fails if the rule cannot be applied. Id and $Fail$ denote success and failure, respectively. The strategy expression $match(T)$ is used to check if the rule T can be applied but does not apply the rule. $(S)orelse(S')$ tries strategy S and if it fails then tries to apply S'. If both strategies fail then the whole statement fails. The strategy $ppick(T_1,\ldots,T_n,\Pi)$ selects one of the transformations $T_1, \ldots T_n$ according to the given probability distribution Π. The strategy $while(S)[(n)]do(S')$ executes strategy S' (not exceeding n iterations if the optional parameter n is specified) while S succeeds. $repeat(S)[max\ n]$ repeatedly executes a strategy S, not exceeding n times; it can never fail (when S fails, it returns Id). We refer the reader to [14] for the full definition of PORGY's strategy language.

PORGY [14] offers an in-built strategy editor, a navigable derivation tree widget, and widgets for the creation of rules and graphs. By navigating on the derivation tree and zooming on different nodes, we can see the various stages in the simulation (see Figure 2); if we click on the black arrows in the derivation tree we can see which rule has been applied and identify the cause of the change in the model state.

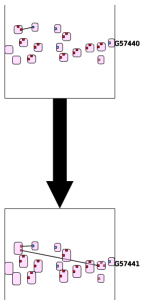


Fig. 2 A portion of the derivation tree in PORGY. The square boxes are nodes in the derivation tree: they contain graphs, and the black arrow represents the application of a rewrite rule.

3 The ABS-GTS Model

In this section we provide a graph-based model of the ABS process as specified by the equations given in Section 2.1. The ABS trading process is modelled hierarchically. The asset trading model sits at the top level of the model hierarchy. It is non-deterministic in nature. Below this system lie several subsystems that model origination, structuring of the deal, SPV transfers and profitability of the sale. In the rest of the paper we focus on the top tier level, which is where the 'rational negligence' phenomenon can be observed.

Asset-transfer transactions are modelled using a combination of global and local data: the global state includes Z (an indicator of market behaviour obtained as the average value of each individual bank's approach, represented by the bank's attribute z) and a *Change* indicator, to detect whether the market has reached a stable state. See Tables 1 and 2 for a description of the nodes used. Similar nodes were used in the model implemented in [11]; here we have additional nodes to represent the Auction and Bidders.

Model execution begins with a parameterised initialisation phase that produces a sample universe with one asset, linked to the owner bank (see Figure 1). Colour attributes in nodes and ports are used to distinguish between classes of objects and to aid in the identification of states of interest (such as negligent behaviour, as explained below).

Entity Name	**Attribute**	**Description**
Bank/Bidder/Potential Buyer (B/BD/PB)	Payoff (payoff)	Returns from re-selling an asset
	z	Indicates whether or not, as a rule, the bank performs independent risk analyses
	Bank ID (b_id)	Bank identifier
Asset (A)	Current Value (c_val)	Cost of purchasing an asset
	Probability of Toxicity (p_tox)	An asset is toxic if the borrowers of the underlying loans are likely to default or are in default
	Actualised Toxicity (a_tox)	Current toxicity level
	Perception (pe)	External rating of the asset by rating agencies
	Due Diligence Cost (ddcost)	Full cost of an independent risk assessment
Change	change	Change in bank approach
	Sum of change (sumofchange)	Sums all changes in a current cycle
Z	z	Represents the global average z
	Number of Iterations (numofiterations)	Counter that keeps track of AllTrade iterations
	Number of Agents (numofagents)	Variable that keeps track of number of banks
Theta	U1	Profitability of being negligent
	U0	Profitability of being diligent
	DeltaU1U0	Difference between U1 and U0
Auction	Kind	Abstract single-sided auction

Table 1 *Nodes and Attributes*

Tables 3 and 4 describe the rewrite rules handling asset transfer in our model. As in the foundational paper [1], our current implementation has been limited to the trading of one asset among k banks. The starting state of the model is the graph shown in Figure 1 and it is from this point that the derivation tree begins to undergo construction as the execution strategy calls on rules that create step-wise transforma-

Entity	Ports	Description
Bank, Bidder	O (Owns)	Edges attached to this port link to assets owned by the bank
	C (Contacts)	Communication channel with another bank
Asset	OB (Owned_by)	Connects the asset to its current owner
Z	E (Environment)	Global entity that tracks current average sentiment
PotentialBuyer	O (Owns)	Links to assets owned by the bank
	C (Contacts)	Communication channel with another bank
	GE (Generates)	Declares a relationship with an analysis node
Change	CH (change)	Keeps track of behaviour changes
Theta	PB (Produced_by)	Links to entity that produces this node
Auction	B (Buyers)	Links to bidders
	S (Seller)	Links to seller

Table 2 *Ports in each kind of node*

tions. Specifically, the asset transfer processes are governed by the strategies *Auction*, *AllTrade* and *FixedPointSearch* (see Strategies 1, 2 and 3 below).

Auction (Strategy 1) starts by specifying that rules will apply anywhere in the current graph (line 1). Line 2 applies rule *sellorder*, to represent a sell request from the asset owner. After a number of buy orders are received (specified in line 3 by repeated applications of the rule *buyorder*), an auction takes place and one of the bidders is selected (line 4, rule *matchorders*). The auction is then closed (line 5).

A basic description of the strategy *AllTrade* (Strategy 2) is as follows: Line 1 starts a trading cycle (the number of iterations is bound by the number of banks, k). Each iteration corresponds to one transaction: First an auction takes place (the Strategy Auction is called in line 1). After the auction, the potential buyer then begins the analysis in line 2 to decide whether or not to follow the negligence rule. It does this by computing the profitability of choices as described in Section 2.1 using rule *beginanalysis*. If diligence is more profitable the deviation rules will apply, otherwise the bank follows the negligence rule (see the `orelse` in lines 3 and 4). The rule *updatez* used in line 5 updates the global Z. We repeat k times in order to give all banks an opportunity to trade.

Strategy 3 controls the full execution: *AllTrade* is iterated until there are no changes in the agent behaviours (i.e., as long as the *change* rule can be applied).

A variant of strategy *AllTrade* replaces the `orelse` operator (lines 3 and 4) by a `ppick` operator, to model probabilistic choice of logit type between following or deviating from the negligence rule. The probability distribution used in this case implements the stochastic "trembles" described in [13] and can be written within our strategy environment as follows:

```
ppick((one(followresult);one(followdecision)),
      (one(deviationresult);one(deviationdecision)),
      udfLogitModel)
```

where `udfLogitModel` is a function that reads the profitability of being negligent or diligent (attributes U1 and U0 in the node Theta of the graph produced by the rule *beginanalysis*) and returns the following values as a list:

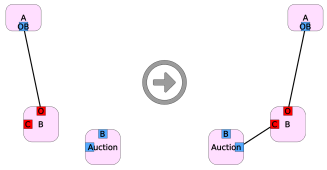
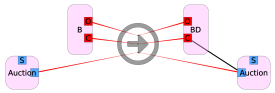
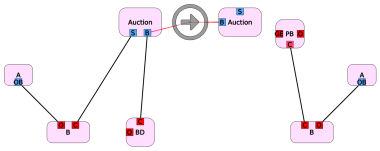
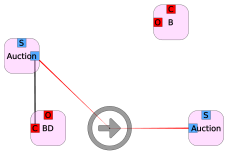
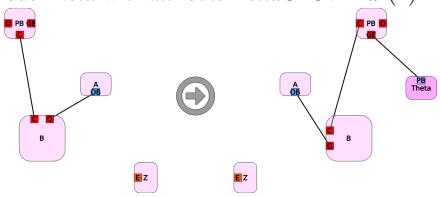
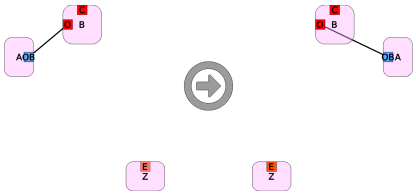
Name of Rule	Description
sellorder	Initiates sell-order communication with Auction 
buyorder	Initiates buy-order communication with Auction 
matchorders	Handles the match of sell-buy orders 
close	Closes the auction 
beginanalysis	Computes profitability $\mathscr{U}(1)$, $\mathscr{U}(0)$ of PB, generating a node $Theta$ with attribute $DeltaU1U0 = \mathscr{U}(1) - \mathscr{U}(0)$.  Algorithm tab: $Theta.U1 = 1 - A.p\_tox(1 - Z.z) - A.c\_val$ $Theta.U0 = (1 - A.p\_tox)(1 - A.c\_val) - A.ddcost$ $Theta.DeltaU1U0 = Theta.U1 - Theta.U0$
updatez	Updates the attribute z in node Z.  Algorithm tab: $Z.z = ((Z.z * (Z.numofagents - 1)) + B.z)/Z.numofagents$

Table 3 *Rewrite Rules*

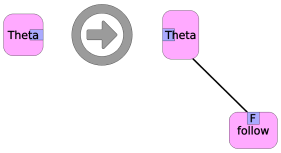
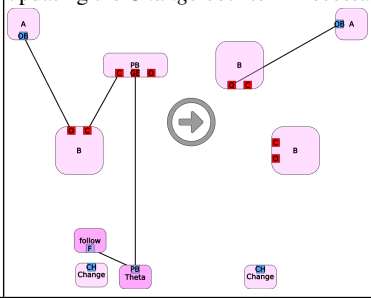
Name of Rule	Description
followresult	Applies if *DeltaU1U0* ≥ 0. It generates a *follow* node if more profitable to not do a full risk analysis.  Arrow-node Condition: If *Theta.DeltaU1U0* ≥ 0
deviationresult	Applies if *DeltaU1U0* < 0. It generates a *deviation* node if more profitable to do a full risk analysis (Similar to *followresult*).
followdecision	Transfers asset and prepares for a new transaction (i.e. cleans up after the decision negligence rule), updating bank's attribute z, updating the *Change* counter if necessary. 
deviationdecision	Transfers asset and prepares for a new transaction (i.e. cleans up after the decision to deviate from the negligence rule), updating bank's attribute z, updating the *Change* counter if necessary (Similar to *followdecision*).
change	Sets the *Change* counter back to 0 if greater than 0.  Algorithm tab: *Change.change* $= 0$ *Change.sumofchange* $= 0$

Table 4 *Rewrite Rules*

$$\frac{\exp^{\mathscr{B}U_i(z=1)}}{\exp^{\mathscr{B}U_i(z=1)}+\exp^{\mathscr{B}U_i(z=0)}} \quad \textit{and} \quad 1-\left(\frac{\exp^{\mathscr{B}U_i(z=1)}}{\exp^{\mathscr{B}U_i(z=1)}+\exp^{\mathscr{B}U_i(z=0)}}\right)$$

where i is the current agent number and $\mathscr{B}$ is the intensity of choice parameter that controls the ease at which fixed point is reached (as specified in [13]).

Levels of toxicity, asset value and due diligence cost are parameters of the simulation, which can be changed in our model by updating values of bank and asset node attributes.

```
setPos(crtGraph);
one(sellorder);
repeat(one(buyorder))(n);
one(matchorders);
repeat(one(close))
```

Strategy 1: *Auction*

```
repeat(#Auction#;
  one(beginanalysis);
  (one(deviationresult);one(deviationdecision)) orelse
  (one(followresult);one(followdecision));
  one(updatez))(k)
```

Strategy 2: *AllTrade*

```
#AllTrade#;
while(match(change))do(
  one(change);
  #AllTrade#)
```

Strategy 3: *FixedPointSearch*

4 Model Properties

First, we show that our model specification is correct with respect to the equational semantics (Section 2.1). This ensures that our model captures the ABS process of interest, and predictions from the ABS models under the same conditions coincide with the predictions produced by our system.

Lemma 1. *Starting from an initial graph that contains at least two bank nodes, one of which owns an asset, and an Auction node, Strategies 1 (Auction), 2 (AllTrade), and 3 (FixedPointSearch) never fail*[3].

Proof. Strategy 1 starts with a *setPos* command, which cannot fail, and then executes a sell order (which cannot fail if the graph has at least two banks, one of which owns an asset, and an auction node), followed by a repeat command, which according to PORGY's semantics [14] can never fail, then matches the buying and selling orders (this rule cannot fail since the previous repeat command generates a redex) and finally the strategy executes another repeat command which cannot fail. Strategy 2 (*AllTrade*) executes a command of the form $repeat(S)(k)$, which can never fail. Since *AllTrade* cannot fail, strategy *FixedPointSearch* can only fail if the rule *change* in the body of the while loop fails, which is impossible due to the condition in the "while" (there is at least one match for *change*).

Theorem 1 (Correctness). *The graph-based model defined by the initial state, rewrite rules and strategies defined above is correct with respect to the equationally defined ABS process (see Section 2.1). More precisely, the graphs generated by*

[3] A strategy fails if it attempts to apply a rule that is not applicable.

the application of the rewrite rules with the given strategy represent states reached by the system governed by the equational ABS model.

Proof. We show that one trading transaction in our system corresponds to one trading transaction in the equational model. Let w be the bank that owns the asset (i.e., the bank linked by an edge to the asset), and let w' be the potential buyer (selected by auction). Rule *beginanalysis* computes the value of the projected profitability made by w' following and not following the negligence rule using the attributes p-tox, c-val and dd-cost (i.e., probability of toxicity, current value and due diligence cost) in the asset, which correspond to the values of p, c and x in the equational model. It computes the difference between $\mathscr{U}_w(1)$ and $\mathscr{U}_w(0)$ using the equations given in section 2.1 and stores it in the attribute $DeltaU1U0$, as indicated in its algorithm tab. The result of this computation is the value specified by the equational model. The strategy ensures that the potential buyer selects the most profitable choice (lines 3-4 of Strategy 2), and the rule *updatez* recomputes the global Z value as outlined in Table 3, as follows:

$$Z_{i+1} = \frac{Z_i * (k-1) + z(w')}{k}$$

which gives the average value specified in the equational model.

Theorem 2 (Completeness). *The graph-based model defined by the initial state, rewrite rules and strategies defined above is complete with respect to stability as specified by the equational ABS model (see Section 2.1). More precisely, if the equational model reaches a stable state, so does our model.*

Proof. (Sketch) The transactions of the equational ABS model are mimicked by the iterations in our strategy. A stable state in the equational model is reached when banks do not change their approach to negligence, which corresponds to absence of "Change" in our model: the *Change* flag is updated as required when rules $followdecision$ and $deviationdecision$ are applied (see Table 4).

Theorems 1 and 2 ensure that our model reaches a stable state if and only if the ABS equational model (see Section 2.1 and [1]) reaches the same stable state.

Theorem 3 (Termination). *The graph program consisting of the initial graph, rewrite rules and strategy described above terminates.*

More generally, if the rule updatez *also changes the values of the asset attributes (reflecting external changes in risk analysis cost, toxicity and asset value) then the graph program terminates if and only if stable state is reached.*

Proof. A state is stable if no bank has changed its mind regarding its negligence choice when given an opportunity to trade. If stable state has been reached, there is no change after executing *AllTrade* hence the while loop found in line 2 of Strategy 3 stops. Conversely if our strategy terminates then the *change* rule does not apply since this is the condition to exit the while, hence no bank has changed its behaviour in *AllTrade* (stability has been reached). Thus, the graph program terminates if and only if the initial graph reaches a stable state.

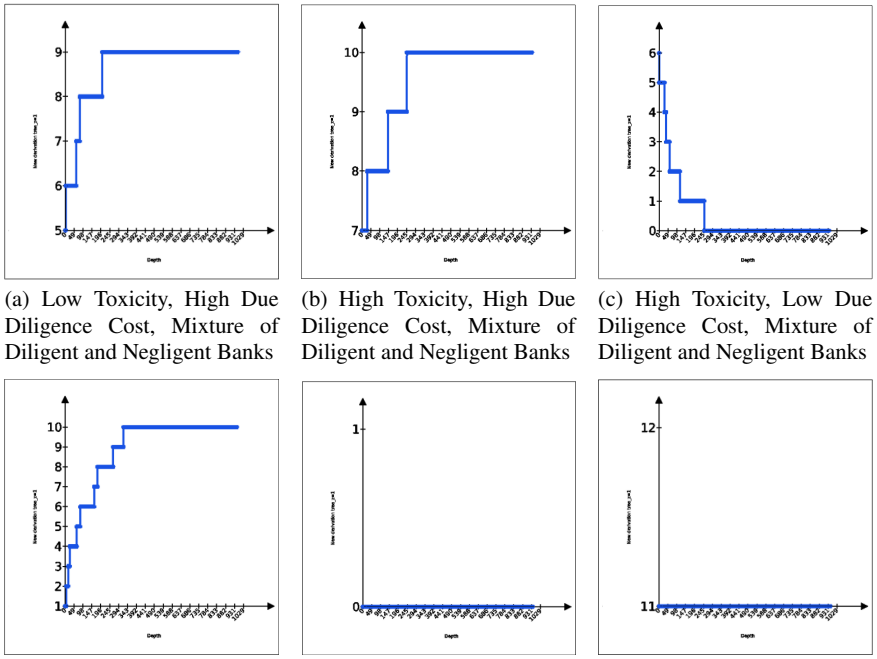
Moreover, if the parameters of the asset do not change during the simulation then the program is guaranteed to terminate, because in this case Z is monotonic (once a bank decides towards diligence or negligence, the rest follow the trend). Thus, in this simple case, the program terminates.

Experiments and Analysis.

A base case validation of the model is described in [11], in which test results line up with results from a traditional ABM simulation given in [1]. In Figure 3 we recall some experimental results, where average Z value is plotted versus depth of the simulation. A natural question arises: What events could have mitigated or further instigated a negligent behaviour? By increasing toxicity values for example we can take into account the increase in interest rates that led to increased default rates and the 2008 crisis. Our experiments show that when toxicity is increased (attribute p in node A) the system reaches a stable state where all banks perform independent risk analysis, as expected. In particular, for high values of p (that is, high probability of toxicity), we observe the expected result when the initial state contains a mixture of negligent and diligent agents: a sharp drop in Z, corresponding to a sharp switch towards diligence which in turn will generate stability. An illustration of this can be seen in Figure 3(c) and notice that given high due diligence costs Figures 3(b) and 3(d) highlight a negligent approach whereas Figures 3(c) and 3(e) reflect the favouring of a diligent approach. However, even for high toxicity, if the initial state is a set of negligent agents, the model reaches equilibrium without switching approach as seen in Figure 3(f).

We observe the following behaviours:

1. Negligent equilibrium: If in the initial graph $Z \approx 1$, then the system arrives at negligent equilibrium (i.e., a result that reflects a community decision to no longer perform due diligence on a particular asset) even when the asset has high probability of toxicity.
   Explanation: For $Z \approx 1$ the profitability equation outlined in section 2.1 reduces to: $\mathscr{U}(1) - \mathscr{U}(0) \approx 1 - c + x_w$ given that the difference between $\mathscr{U}(1)$ and expected profit when the rule is not followed (i.e. $\mathscr{U}(0)$) is $p(Z-c) + x_w$. This linear equation is computed at each iteration of the repeat loop. The result is positive given that c and x_w are both positive constants smaller than 1.
   Similarly, we observe that if p is high but in the initial graph the majority of banks are deviating from the negligence rule, then the system reaches a due diligence stable state.
2. Indefinite propagation: A continuous increase in the number of negligent bank agents means that a market crash can be postponed. This condition, although not feasible in a real market, is valid in equational models.
   Explanation: A continuous increase in the number of agents used in calculating the average current sentiment, Z, as outlined in section 2.1 and as computed by PORGY, means that the value of Z used in deciding whether or not to perform an independent risk analysis can remain unchanged.



(a) Low Toxicity, High Due Diligence Cost, Mixture of Diligent and Negligent Banks

(b) High Toxicity, High Due Diligence Cost, Mixture of Diligent and Negligent Banks

(c) High Toxicity, Low Due Diligence Cost, Mixture of Diligent and Negligent Banks

(d) High Toxicity, High Due Diligence Cost, Diligent Banks

(e) High Toxicity, Low Due Diligence Cost, Diligent Banks

(f) High Toxicity, Low Due Diligence Cost, Negligent Banks

Fig. 3 *Experiment Results.* y-axis: Count of the number of negligent banks. The intersection of x and y axes in the case of a starting universe of purely diligent banks corresponds to the co-ordinates (0,0) as opposed to (11,0) in the case where we begin with negligent banks. Curves tending upwards reflect a negligent equilibrium result

3. Dangerous Equilibrium: A negligent stable state can be reached despite high toxicity under certain circumstances (high due diligence costs).
   Explanation: A sensitivity analysis shows that for a certain range of high due diligence cost values, negligent equilibrium can be obtained despite high toxicity values, even if initially negligence is not the norm, see Figure 3 (b and d).

The results obtained with the basic experiments performed so far suggest that the graph rewriting approach, and in particular the derivation tree provided by PORGY, could be used to get insights beyond simulation runs. For example, the derivation tree could be used to search for states with specific properties, or to identify the occurrence of specific events (e.g., the first application of a specific rule). More meaningful analyses could be carried out, such as calculating propagation speeds (i.e., number of steps it takes for rule sentiment to be adopted by all agents relative to the size of network or the rate of change of average sentiment within different environments), taking into the account the pay-down factor of the loans supporting the asset and the expected contractual degradation of the asset itself, etc.

5 Conclusions

We have shown that strategic port-graph rewriting provides a basis for the design and implementation of graph models of the rational negligence phenomenon. Whilst ABMs rely on the internal processing of its agents, GTSs provide at each point in time a holistic view of the system state and a visual trace of the specific rules that trigger specific behaviours. In future, we will further develop the model using hierarchical graphs [12] to capture all tiers of the model, and also generalise the rules to permit dynamic changes in key attributes such as asset toxicity and costs.

References

1. Kartik Anand, Alan Kirman, and Matteo Marsili. Epidemics of rules, rational negligence and market crashes. *The European Journal of Finance*, 19(5):438–447, 2013.
2. Oana Andrei, Maribel Fernández, Hélène Kirchner, and Bruno Pinaud. Strategy-driven exploration for rule-based models of biochemical systems with Porgy. In William S. Hlavacek, editor, *Modeling Biomolecular Site Dynamics: Methods and Protocols*. Springer, 2018.
3. Emilie Balland, Paul Brauner, Radu Kopetz, Pierre-Etienne Moreau, and Antoine Reilles. Tom: Piggybacking rewriting on Java. In Franz Baader, editor, *Term Rewriting and Applications, 18th International Conference, RTA 2007, Paris, France, June 26-28, 2007, Proceedings*, volume 4533 of *Lecture Notes in Computer Science*, pages 36–47. Springer, 2007.
4. Walid Belkhir, Alain Giorgetti, and Michel Lenczner. A symbolic transformation language and its application to a multiscale method. *Journal of Symbolic Computation*, 65:49 – 78, 2014.
5. Peter Borovanský, Claude Kirchner, Hélène Kirchner, and Christophe Ringeissen. Rewriting with strategies in ELAN: A functional semantics. *Int. J. Found. Comput. Sci.*, 12(1):69–95, 2001.
6. Martin Bravenboer, Karl Trygve Kalleberg, Rob Vermaas, and Eelco Visser. Stratego/xt 0.17. A language and toolset for program transformation. *Sci. Comput. Program.*, 72(1-2):52–70, 2008.
7. Yves Caseau, Francois-Xavier Josset, and Francois Laburthe. Claire: Combining sets, search and rules to better express algorithms, 2004. arXiv.
8. Juan de Lara, Esther Guerra, Artur Boronat, Reiko Heckel, and Paolo Torrini. Domain-specific discrete event modelling and simulation using graph transformation. *Software and System Modeling*, 13(1):209–238, 2014.
9. Besik Dundua, Temur Kutsia, and Klaus Reisenberger-Hagmayer. An overview of pρlog. In Yuliya Lierler and Walid Taha, editors, *Practical Aspects of Declarative Languages - 19th International Symposium, PADL 2017, Paris, France, January 16-17, 2017, Proceedings*, volume 10137 of *Lecture Notes in Computer Science*, pages 34–49. Springer, 2017.
10. Francisco Durán, Steven Eker, Santiago Escobar, Narciso Martí-Oliet, José Meseguer, Rubén Rubio, and Carolyn L. Talcott. Programming and symbolic computation in Maude. *J. Log. Algebraic Methods Program.*, 110, 2020.
11. Nneka Ene. Implementation of a port-graph model for finance. In *Proceedings of TERMGRAPH 2018: Computing with terms and graphs*, pages 14–25, 2019.
12. Nneka Ene, Maribel Fernández, and Bruno Pinaud. Attributed hierarchical port graphs and applications. In *Proceedings Fourth International Workshop on Rewriting Techniques for Program Transformations and Evaluation, WPTE@FSCD 2017, Oxford, UK, 8th September 2017*, pages 2–19, 2017.

13. J.Doyne Farmer, M. Gallegati, C. Hommes, A. Kirman, P. Ormerod, S. Cincotti, A. Sanchez, and D. Helbing. A complex systems approach to constructing better models for managing financial markets and the economy. *The European Physical Journal Special Topics*, 214(1):295–324, 2012.
14. Maribel Fernández, Hélène Kirchner, and Bruno Pinaud. Strategic port graph rewriting: an interactive modelling framework. *Mathematical Structures in Computer Science*, 29(5):615–662, 2019.
15. Maribel Fernández, Hélène Kirchner, Bruno Pinaud, and Jason Vallet. Labelled graph strategic rewriting for social networks. *J. Log. Algebraic Methods Program.*, 96:12–40, 2018.
16. Maribel Fernández, Bruno Pinaud, and János Varga. A port graph rewriting approach to relational database modelling. In *Proceedings 29th International Conference on Logic-based Program Synthesis and Transformation, LOPSTR 2019, Porto, October 2019*, Lecture Notes in Computer Science. Springer, 2020.
17. Thom W. Frühwirth. Parallelism, concurrency and distribution in constraint handling rules: A survey. *Theory Pract. Log. Program.*, 18(5-6):759–805, 2018.
18. Rubino Geiß and Moritz Kroll. Grgen.net: A fast, expressive, and general purpose graph rewrite tool. In *Applications of Graph Transformations with Industrial Relevance, Third International Symposium, AGTIVE 2007, Kassel, Germany, October 10-12, 2007, Revised Selected and Invited Papers*, pages 568–569, 2007.
19. Amir Hossein Ghamarian, Maarten de Mol, Arend Rensink, Eduardo Zambon, and Maria Zimakova. Modelling and analysis using GROOVE. *STTT*, 14(1):15–40, 2012.
20. Gary Gorton and Andrew Metrick. Securitization. Working Paper 18611, National Bureau of Economic Research, December 2012.
21. Charles Ka Yui Leung and Thomas A. Lubik. Introduction: Dynamic stochastic general equilibrium modelling and the study of Asia-Pacific economies. *Pacific Economic Review*, 17(2):204–207, 2012.
22. Kalliopi Kravari and Nick Bassiliades. A survey of agent platforms. *Journal of Artificial Societies and Social Simulation*, 18(1):11, 2015.
23. Jean Krivine, Vincent Danos, and Arndt Benecke. Modelling epigenetic information maintenance: A Kappa tutorial. In *Computer Aided Verification, 21st International Conference, CAV 2009, Grenoble, France, June 26 - July 2, 2009. Proceedings*, pages 17–32, 2009.
24. Michael Löwe. Algebraic approach to single-pushout graph transformation. *Theor. Comput. Sci.*, 109(1&2):181–224, 1993.
25. Mircea Marin and Temur Kutsia. Foundations of the rule-based system ρLog. *J. Appl. Non Class. Logics*, 16(1-2):151–168, 2006.
26. Sheri Markose, Yang Dong, and Bewaji Oluwasegun. An multi-agent model of rmbs, credit risk transfer in banks and financial stability: Implications of the subprime crisis, 2008.
27. Narciso Martí-Oliet, José Meseguer, and Alberto Verdejo. Towards a strategy language for Maude. *Electronic Notes in Theoretical Computer Science*, 117:417 – 441, 2005. Proceedings of the Fifth International Workshop on Rewriting Logic and Its Applications (WRLA 2004).
28. Detlef Plump. The graph programming language GP. In Symeon Bozapalidis and George Rahonis, editors, *Algebraic Informatics: Third International Conference, CAI 2009, Thessaloniki, Greece, May 19-22, 2009, Proceedings*, pages 99–122. Springer, 2009.
29. Andy Schürr, Andreas J Winter, and Albert Zündorf. The PROGRES approach: Language and environment. In *Handbook of graph grammars and computing by graph transformation*, pages 487–550. World Scientific Publishing Co., Inc., 1999.
30. Adam M. Smith, Wen Xu, Yao Sun, James R. Faeder, and G.Elisabeta Marai. Rulebender: integrated modeling, simulation and visualization for rule-based intracellular biochemistry. *BMC Bioinformatics*, 13(8), 2012.
31. Gabriele Taentzer. AGG: A graph transformation environment for modeling and validation of software. In *Applications of Graph Transformations with Industrial Relevance*, volume 3062 of *Lecture Notes in Computer Science*, pages 446–453. Springer Berlin Heidelberg, 2004.
32. Jeroen van den Bos, Mark Hills, Paul Klint, Tijs van der Storm, and Jurgen J. Vinju. Rascal: From algebraic specification to meta-programming. *Electronic Proceedings in Theoretical Computer Science*, 56:15–32, Jun 2011.

33. M. G. J. van den Brand, A. van Deursen, J. Heering, H. A. de Jong, M. de Jonge, T. Kuipers, P. Klint, L. Moonen, P. A. Olivier, J. Scheerder, J. J. Vinju, E. Visser, and J. Visser. The ASF+SDF meta-environment: A component-based language development environment. In Reinhard Wilhelm, editor, *Compiler Construction*, pages 365–370, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.
34. János Varga. Finding the transitive closure of functional dependencies using strategic port graph rewriting. In *Proceedings Tenth International Workshop on Computing with Terms and Graphs, TERMGRAPH@FSCD 2018, Oxford, UK, 7th July 2018*, pages 50–62, 2018.
35. Gergely Varró, Andy Schürr, and Dániel Varró. Benchmarking for graph transformation. In *2005 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC 2005), 21-24 September 2005, Dallas, TX, USA*, pages 79–88. IEEE Computer Society, 2005.
36. Eelco Visser. Stratego: A language for program transformation based on rewriting strategies. In Aart Middeldorp, editor, *Rewriting Techniques and Applications, 12th International Conference, RTA 2001, Utrecht, The Netherlands, May 22-24, 2001, Proceedings*, volume 2051 of *Lecture Notes in Computer Science*, pages 357–362. Springer, 2001.