Algebraic Laws for Weak Consistency

Andrea Cerone[1], Alexey Gotsman[2], and Hongseok Yang[3]

1 **Imperial College London, UK**
`a.cerone@imperial.ac.uk`

2 **IMDEA Software Institute, Madrid, Spain**
`alexey.gotsman@imdea.org`

3 **University of Oxford, UK**
`hongseok.yang@cs.ox.ac.uk`

Abstract

Modern distributed systems often rely on so called weakly consistent databases, which achieve scalability by weakening consistency guarantees of distributed transaction processing. The semantics of such databases have been formalised in two different styles, one based on abstract executions and the other based on dependency graphs. The choice between these styles has been made according to intended applications. The former has been used for specifying and verifying the implementation of the databases, while the latter for proving properties of client programs of the databases. In this paper, we present a set of novel algebraic laws (inequalities) that connect these two styles of specifications. The laws relate binary relations used in a specification based on abstract executions to those used in a specification based on dependency graphs. We then show that this algebraic connection gives rise to so called robustness criteria: conditions which ensure that a client program of a weakly consistent database does not exhibit anomalous behaviours due to weak consistency. These criteria make it easy to reason about these client programs, and may become a basis for dynamic or static program analyses. For a certain class of consistency models specifications, we prove a full abstraction result that connects the two styles of specifications.

1998 ACM Subject Classification C.2.4 Distributed Databases

Keywords and phrases Weak Consistency Models, Distributed Databases, Dependency Graphs

Digital Object Identifier 10.4230/LIPIcs.CONCUR.2017.26

1 Introduction

Modern distributed systems often rely on databases that achieve scalability by weakening consistency guarantees of distributed transaction processing. These databases are said to implement weak consistency models. Such weakly consistent databases allow for faster transaction processing, but exhibit anomalous behaviours, which do not arise under a database with a strong consistency guarantee, such as serialisability. Two important problems for the weakly consistent databases are: (i) to find elegant formal specifications of their consistency models and to prove that these specifications are correctly implemented by protocols used in the databases; (ii) to develop effective reasoning techniques for applications running on top of such databases. These problems have been tackled by using two different formalisms, which model the run-time behaviours of weakly consistent databases differently.

When the goal is to verify the correctness of a protocol implementing a weak consistency model, the run-time behaviour of a distributed database is often described in terms of *abstract executions* [13], which abstract away low-level implementation details of the database (Section 2). An example of abstract execution is depicted in Figure 1; ignore the bold edges for the moment. It comprises four transactions, T_0, T_1, T_2, and S; transaction T_0 initializes

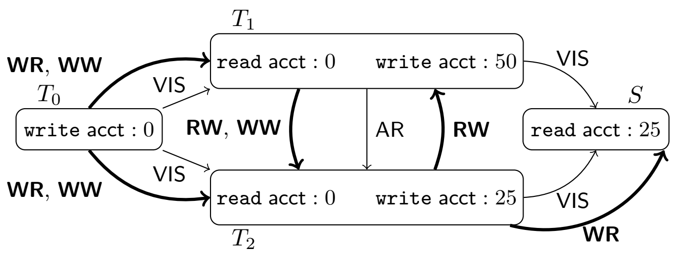

Figure 1 An example of abstract execution and of dependency graph.

the value of an object acct to 0; transactions T_1 and T_2 increment the value of acct by 50 and 25, respectively, after reading its initial value; transaction S reads the value of acct. In this abstract execution, both the updates of T_1 and T_2 are **VIS**ible to transaction S, as witnessed by the two VIS-labelled edges: $T_1 \xrightarrow{\mathsf{VIS}} S$ and $T_2 \xrightarrow{\mathsf{VIS}} S$.

On the other hand, the update of T_1 is not visible to T_2, and vice versa, as indicated by the absence of an edge labelled with VIS between these transactions. Intuitively, the absence of such an edge means that T_1 and T_2 are executed concurrently. Because S sees T_1 and T_2, as indicated by VIS-labelled edges from T_1 and T_2 to S, the result of reading the value of acct in S must be one of the values written by T_1 and T_2. However, because these transactions are concurrent, there is a race, or *conflict*, between them. The AR-labelled edge connecting T_1 to T_2, is used to **AR**bitrate the conflict: it states that the update of T_1 is older than the one of T_2, hence the query of acct in S returns the value written by the latter.

The style of specifications of consistency models in terms of abstract executions can be given by imposing constraints over the relations $\mathsf{VIS}, \mathsf{AR}$ (Section 2.1). A set of transactions $\mathcal{T} = \{T_1, T_2, \cdots\}$, called a *history*, is allowed by a consistency model specification if it is possible to exhibit two witness relations $\mathsf{VIS}, \mathsf{AR}$ over $\mathcal{T}$ such that the resulting abstract execution satisfies the constraints imposed by the specification. For example, *serialisability* can be specified by requiring that the relation VIS should be a strict total order. The set of transactions $\{T_0, T_1, T_2, S\}$ from Figure 1 is not serialisable: it is not possible to choose a relation VIS such that the resulting abstract execution relates the transactions T_1, T_2 and the results of read operations are consistent with visible updates.

Specifications of consistency models using abstract executions have been used in the work on proving the correctness of protocols implementing weak consistency models, as well as on justifying operational, implementation-dependent descriptions of these models [11, 12, 13, 15].

The second formalism used to define weak consistency models is based on the notion of *dependency graphs* [1], and it has been used for proving properties of client programs running on top of a weakly consistent database. Dependency graphs capture the data dependencies of transactions at run-time (Section 3); the transactions $\{T_0, T_1, T_2, S\}$ depicted above, together with the bold edges but without normal edges, constitute an example of dependency graph. The edge $T_2 \xrightarrow{\mathsf{WR(acct)}} S$[1] denotes a *write-read dependency.* It means that the read of acct in transaction S returns the value written by transaction T_2, and the edges $T_0 \xrightarrow{\mathsf{WR(acct)}} T_1$ and $T_0 \xrightarrow{\mathsf{WR(acct)}} T_2$ mean something similar. The edge $T_1 \xrightarrow{\mathsf{WW(acct)}} T_2$ denotes a *write-write dependency*, and says that the write to acct in T_2 supersedes the write to the same object in

[1] For simplicity, references to the object acct have been removed from the dependencies of Figure 1.

T_1. The remaining edges $T_1 \xrightarrow{\mathsf{RW(acct)}} T_2$ and $T_2 \xrightarrow{\mathsf{RW(acct)}} T_1$ express *anti-dependencies*. The former means that T_1 reads a value for object acct which is older than the value written by T_2.

When using dependency graphs, consistency models are specified as sets of transactions for which there exist $\mathsf{WR}, \mathsf{WW}, \mathsf{RW}$ relations that satisfy certain properties, usually stated as particular relations being acyclic [7, 16]; for example, serialisability can be specified by requiring that dependency graphs are acyclic. Because dependencies of transactions can be over-approximated at the compilation time, specifications of consistency models in terms of dependency graphs have been widely used for manually or automatically reasoning about properties of client programs of weakly consistent databases [19, 27]. They have also been used in the complexity and undecidability results for verifying implementations of consistency models [9].

Our ultimate aim is to reveal a deep connection between these two styles of specifying weak consistency models, which was hinted at for specific consistent models in the literature. Such a connection would, for instance, give us a systematic way to derive a specification of a weak consistency model based on dependency graphs from the specification based on abstract executions, while ensuring that the original and the derived specifications are equivalent in a sense. In doing so, it would enable us to prove properties about client programs of a weakly consistent database using techniques based on dependency graphs [9, 16, 18] even when the consistency model of the database is specified in terms of abstract executions.

In this paper, we present our first step towards this ultimate aim. First, we observe that each abstract execution determines an underlying dependency graph. Then we study the connection between these two structures at an algebraic level. We propose a set of algebraic laws, parametric in the specification of a consistency model to which the original abstract execution belongs (Section 4). These laws can be used to derive properties of the form $R_{\mathsf{G}} \subseteq R_{\mathsf{A}}$: here R_{G} is an expression from the Kleene Algebra with Tests [22] whose ground terms are run-time dependencies of transactions, and tests are properties over transactions. The relation R_{A} is one of the fundamental relations of abstract executions: VIS, AR, or a novel relation $\overline{\mathsf{VIS}^{-1}}$ that we call *anti-visibility*, defined as $\overline{\mathsf{VIS}^{-1}} = \{(T, S) \mid \neg(S \xrightarrow{\mathsf{VIS}} T)\}$. Some of the algebraic laws that we propose show that there is a direct connection between each kind of dependencies and the relations of abstract executions: $\mathsf{WR} \subseteq \mathsf{VIS}, \mathsf{WW} \subseteq \mathsf{AR}$, and $\mathsf{RW} \subseteq \overline{\mathsf{VIS}^{-1}}$. The other laws capture the connection between the relations of abstract executions $\mathsf{VIS}, \mathsf{AR}$, and $\overline{\mathsf{VIS}^{-1}}$. The exact nature of this connection depends on the specification of the consistency model of the considered abstract execution.

We are particularly interested in deriving properties of the form $R_{\mathsf{G}} \subseteq \mathsf{AR}$. Properties of this form give rise to so called robustness criteria for client programs, conditions ensuring that a program only exhibits serialisable behaviours even when it runs under a weak consistency model [7, 10, 19]. Because AR is a total order, this implies that R_{G} must be acyclic, hence all cycles must be in the complement of R_{G}. We can then check for the absence of such *critical* cycles at compile time: because dependency graphs of serialisable databases are always acyclic, this ensures that said application only exhibits serialisable behaviours.

As another contribution we show that, for a relevant class of consistency models, our algebraic laws can be used to derive properties which are not only necessary, but also sufficient, for dependency graphs in such models (Section 5).

2 Abstract Executions

We consider a database storing objects in $\mathsf{Obj} = \{x, y, \cdots\}$, which for simplicity we assume to be integer-valued. Client programs can interact with the database by executing operations from a set Op, grouped inside *transactions*. We leave the set Op unspecified, apart from requiring that it contains read and write operations over objects: $\{\mathtt{write}(x, n), \mathtt{read}(x, n) \mid x \in \mathsf{Obj}, n \in \mathbb{N}\} \subseteq \mathsf{Op}$.

Histories. To specify a consistency model, we first define the set of all client-database interactions allowed by the model. We start by introducing (run-time) *transactions* and *histories*, which record such interactions in a single computation. Transactions are elements from a set $\mathbb{T} = \{T, S, \cdots\}$; the operations executed by transactions are given by a function $\mathsf{behav} : \mathbb{T} \to 2^{\mathsf{Op}}$, which maps a transaction T to a set of operations that are performed by the transaction and can be observed by other transactions. We often abuse notations and just write $o \in T$ (or $T \ni o$) instead of $o \in \mathsf{behav}(T)$. We adopt similar conventions for $\mathcal{O} \subseteq \mathsf{behav}(T)$ and $\mathcal{O} = \mathsf{behav}(T)$ where $\mathcal{O}$ is a subset of operations.

We assume that transactions enjoy *atomic visibility* : for each object x, (i) a transaction S never observes two different writes to x from a single transaction T and (ii) it never reads two different values of x. Formally, the requirements are that if $T \ni (\mathtt{write}\ x : n)$ and $T \ni (\mathtt{write}\ x : m)$, or $T \ni (\mathtt{read}\ x : n)$ and $T \ni (\mathtt{read}\ x : m)$, then $n = m$. Our treatment of atomic visibility is taken from our previous work on transactional consistency models [15]. Atomic visibility is guaranteed by many consistency models [5, 19, 28]. We point out that although we focus on transactions in distributed systems in the paper, our results apply to weak shared-memory models [4]; there a transaction T is the singleton set of a read operation ($T = \{\mathtt{read}\ x : n\}$), that of a write operation ($T = \{\mathtt{write}\ x : n\}$), or the set of read and write representing a *compare and set* operation ($T = \{\mathtt{read}\ x : n,\ \mathtt{write}\ x : m\}$).

For each object x, we let $\mathsf{Writes}_x := \{T \mid \exists n.\ (\mathtt{write}\ x : n) \in T\}$ and $\mathsf{Reads}_x := \{T \mid \exists n,\ (\mathtt{read}\ x : n) \in \mathcal{T}\}$ be the sets of transactions that write to and read from x, respectively.

▶ **Definition 1.** A history $\mathcal{T}$ is a finite set of transactions $\{T_1, T_2, \cdots, T_n\}$.

Consistency Models. A consistency model Γ is a set of histories that may arise when client programs interact with the database. To define Γ formally, we augment histories with two relations, called *visibility* and *arbitration*.

▶ **Definition 2.** An *abstract execution* $\mathcal{X}$ is a tuple $(\mathcal{T}, \mathsf{VIS}, \mathsf{AR})$ where $\mathcal{T}$ is a history and $\mathsf{VIS}, \mathsf{AR} \subseteq (\mathcal{T} \times \mathcal{T})$ are relations on transactions such that $\mathsf{VIS} \subseteq \mathsf{AR}$ and AR is a strict total order[2].

We often write $T \xrightarrow{\mathsf{VIS}} S$ for $(T, S) \in \mathsf{VIS}$, and similarly for other relations. For each abstract execution $\mathcal{X} = (\mathcal{T}, \mathsf{VIS}, \mathsf{AR})$, we let $\mathcal{T}_{\mathcal{X}} := \mathcal{T}$, $\mathsf{VIS}_{\mathcal{X}} := \mathsf{VIS}$, and $\mathsf{AR}_{\mathcal{X}} := \mathsf{AR}$.

In an abstract execution $\mathcal{X}$, $T \xrightarrow{\mathsf{VIS}_{\mathcal{X}}} S$ means that the read operations in S may depend on the updates of T, while $T \xrightarrow{\mathsf{AR}_{\mathcal{X}}} S$ means that the update operations of S supersede those performed by T. Naturally, one would expect that the value fetched by read operations in a transaction T is the most up-to-date one among all the values written by transactions visible to T. For simplicity, we assume that such a transaction always exists.

[2] A relation $R \subseteq \mathcal{T} \times \mathcal{T}$ is a strict (partial) order if it is transitive and irreflexive; it is total if for any $T, S \in \mathcal{T}$, either $T = S$, $(T, S) \in R$ or $(S, T) \in R$.

▶ **Definition 3.** An abstract execution $\mathcal{X} = (\mathcal{T}, \mathsf{VIS}, \mathsf{AR})$ respects the *Last Write Win* (**LWW**) policy, if for all $T \in \mathcal{T}$ such that $T \ni (\texttt{read}\ x : n)$, the set $\mathcal{T}' := \left(\mathsf{VIS}^{-1}(T) \cap \mathsf{Writes}_x\right)$ is not empty, and $\max_{\mathsf{AR}}(\mathcal{T}') \ni (\texttt{write}\ x : n)$, where $\max_{\mathsf{AR}}(\mathcal{T}')$ is the AR-supremum of $\mathcal{T}'$.

▶ **Definition 4.** An abstract execution $\mathcal{X} = (\mathcal{T}, \mathsf{VIS}, \mathsf{AR})$ *respects causality* if VIS is transitive. Any abstract execution that respects both causality and the LWW policy is said to be *valid*.

We always assume an abstract execution to be valid, unless otherwise stated. Causality is respected by all abstract executions allowed by several interesting consistency models. They also simplify the mathematical development of our results. In [17, Appendix B], we explain how our results can be generalised for consistency models that do not respect causality. We also discuss how the model can be generalised to account for sessions and session guarantees [29].

We can specify a consistency model using abstract executions in two steps. First, we identify properties on abstract executions, or *axioms*, that formally express an informal consistency guarantee, and form a set with the abstract executions satisfying the properties. Next, we project abstract executions in this set to underlying histories, and define a consistency model Γ to be the set of resulting histories.

Abstract executions hide low-level operational details of the interaction between client programs and weakly consistent databases. This benefit has been exploited for proving that such databases implement intended consistency models [11, 12, 13, 15, 20].

2.1 Specification of Weak Consistency Models

In this section we introduce a simple framework for specifying consistency models using the style of specification discussed above. In our framework, axioms of consistency models relate the visibility and arbitration relations via inequalities of the form $R_1 \ ; \mathsf{AR}_{\mathcal{X}} \ ; R_2 \subseteq \mathsf{VIS}_{\mathcal{X}}$, where R_1 and R_2 are particular relations over transactions, and $\mathcal{X}$ is an abstract execution. As we will explain later, axioms of this form establish a necessary condition for two transactions in an abstract execution $\mathcal{X}$ to be related by $\mathsf{VIS}_{\mathcal{X}}$, i.e. they cannot be executed concurrently. Despite its simplicity, the framework is expressive enough to capture several consistency models for distributed databases [15, 23]; as we will show in Section 4, one of the benefits of this simplicity is that we can infer robustness criteria of consistency models in a systematic way.

As we will see, the relations R_1, R_2 in axioms of the form above, may depend on the visibility relation of the abstract execution $\mathcal{X}$. To define such relations, we introduce the notion of *specification function*.

▶ **Definition 5.** A function $\rho : 2^{(\mathbb{T} \times \mathbb{T})} \to 2^{(\mathbb{T} \times \mathbb{T})}$ is a *specification function* if for every history $\mathcal{T}$ and relation $R \subseteq \mathcal{T} \times \mathcal{T}$, then $\rho(R) = \rho(\mathcal{T} \times \mathcal{T}) \cap R?$. Here $R?$ is the reflexive closure of R. A *consistency guarantee*, or simply *guarantee*, is a pair of specification functions (ρ, π).

Definition 5 ensures that specification functions are defined locally: for any $R_1, R_2 \subseteq \mathcal{T} \times \mathcal{T}$, $\rho(R_1 \cup R_2) = \rho(R_1) \cup \rho(R_2)$, and in particular for any $R \subseteq \mathcal{T} \times \mathcal{T}$, $\rho(R) = \left(\bigcup_{T,S \in \mathcal{T}} \rho(\{(T,S)\})\right) \cap R?$. The reflexive closure in Definition 5 is needed because we will always apply specification functions to irreflexive relations (namely, the visibility relation of abstract executions), although the result of this application need not be irreflexive. For example, $\rho_{\mathsf{Id}}(R) := \mathsf{Id}$, where Id is the identity function, is a valid specification function.

Each consistency guarantee (ρ, π) defines, for each abstract execution $\mathcal{X}$, an axiom of the form $\rho(\mathsf{VIS}_{\mathcal{X}}) \ ; \mathsf{AR}_{\mathcal{X}} \ ; \pi(\mathsf{VIS}_{\mathcal{X}}) \subseteq \mathsf{VIS}_{\mathcal{X}}$: if this axiom is satisfied by $\mathcal{X}$, we say that $\mathcal{X}$

Function		Definition	Guarantee	Associated Axiom
$\rho_{\mathsf{Id}}(R)$	$=$	Id	$(\rho_{\mathsf{Id}}, \rho_{\mathsf{Id}})$	$\mathsf{AR} \subseteq \mathsf{VIS}$
$\rho_{\mathsf{SI}}(R)$	$=$	$R \backslash \mathsf{Id}$	$(\rho_{\mathsf{Id}}, \rho_{\mathsf{SI}})$	$\mathsf{AR} \mathbin{;} \mathsf{VIS} \subseteq \mathsf{VIS}$
$\rho_x(R)$	$=$	$[\mathsf{Writes}_x]$	(ρ_x, ρ_x)	$[\mathsf{Writes}_x] \mathbin{;} \mathsf{AR} \mathbin{;} [\mathsf{Writes}_x] \subseteq \mathsf{VIS}$
$\rho_S(R)$	$=$	$[\mathtt{SerTx}]$	(ρ_S, ρ_S)	$[\mathtt{SerTx}] \mathbin{;} \mathsf{AR} \mathbin{;} [\mathtt{SerTx}] \subseteq \mathsf{VIS}$

Figure 2 Some Specification Functions and Consistency Guarantees.

satisfies the consistency guarantee (ρ, π). Consistency guarantees impose a condition on when two transactions T, S in an abstract execution $\mathcal{X}$ are not allowed to execute concurrently, i.e. they must be related by a $\mathsf{VIS}_{\mathcal{X}}$ edge. By definition, in abstract executions visibility edges cannot contradict arbitration edges, hence it is only natural that the order in which the transactions T, S above are executed is determined by the arbitration order: in fact, the definition of specification function ensures that $\rho(\mathsf{VIS}_{\mathcal{X}}) \subseteq \mathsf{VIS}_{\mathcal{X}}?$ and $\pi(\mathsf{VIS}_{\mathcal{X}}) \subseteq \mathsf{VIS}_{\mathcal{X}}?$, so that $(\rho(\mathsf{VIS}_{\mathcal{X}}) \mathbin{;} \mathsf{AR}_{\mathcal{X}} \mathbin{;} \pi(\mathsf{VIS}_{\mathcal{X}})) \subseteq \mathsf{AR}_{\mathcal{X}}$ for all abstract executions $\mathcal{X}$.

▶ **Definition 6.** A *consistency model specification* Σ or *x-specification* is a set of consistency guarantees $\{(\rho_i, \pi_i)\}_{i \in I}$ for some index set I.

We define $\mathsf{Executions}(\Sigma)$ to be the set of valid abstract executions that satisfy all the consistency guarantees of Σ. We let $\mathsf{modelOf}(\Sigma) := \{\mathcal{T}_{\mathcal{X}} \mid \mathcal{X} \in \mathsf{Executions}(\Sigma)\}$.

Examples of Consistency Model Specifications

Figure 2 shows several examples of specification functions and consistency guarantees. In the figure we use the relations $[\mathcal{T}] := \{(T,T) \mid T \in \mathcal{T}\}$ and $[o] := \{(T,T) \mid T \ni o\}$ for $\mathcal{T} \subseteq \mathbb{T}$ and $o \in \mathsf{Op}$. The guarantees in the figure can be composed together to specify, among others, several of the consistency models considered in [15]: we give some examples of them below. Each of these consistency models allows different kinds of anomalies: due to lack of space, these are illustrated in [17, Appendix A].

Causal Consistency [24]. This is the weakest consistency model we consider. It is specified by $\Sigma_{\mathsf{CC}} = \emptyset$. In this case, all abstract executions in $\mathsf{Executions}(\Sigma_{\mathsf{CC}})$ respect causality. The execution in Figure 1 is an example in $\mathsf{Executions}(\Sigma_{\mathsf{CC}})$.

Red-Blue Consistency [23]. This model extends causal consistency by marking a subset of transactions as serialisable, and ensuring that no two such transactions appear to execute concurrently. We model red-blue consistency via the x-specification $\Sigma_{\mathsf{RB}} = \{(\rho_S, \rho_S)\}$. In the definition of ρ_S, an element $\mathtt{SerTx} \in \mathsf{Op}$ is used to mark transactions as serialisable, and the specification requires that in every execution $\mathcal{X} \in \mathsf{Executions}(\Sigma_{\mathsf{RB}})$, any two transactions $T, S \ni \mathtt{SerTx}$ in $\mathcal{X}$ be compared by $\mathsf{VIS}_{\mathcal{X}}$. The abstract execution from Figure 1 is included in $\mathsf{Executions}(\Sigma_{\mathsf{RB}})$, but if it were modified so that transactions T_1, T_2 were marked as serialisable, then the result would not belong to $\mathsf{Executions}(\Sigma_{\mathsf{RB}})$.

Parallel Snapshot Isolation (PSI) [26, 28]. This model strengthens causal consistency by enforcing the *Write Conflict Detection* property: transactions writing to one same object do not execute concurrently. We let $\Sigma_{\mathsf{PSI}} = \{(\rho_x, \rho_x)\}_{x \in \mathsf{Obj}}$: every execution $\mathcal{X} \in \mathsf{Executions}(\Sigma_{\mathsf{PSI}})$ satisfies the inequality $([\mathsf{Writes}_x] \mathbin{;} \mathsf{AR}_{\mathcal{X}} \mathbin{;} [\mathsf{Writes}_x]) \subseteq \mathsf{VIS}_{\mathcal{X}}$, for all $x \in \mathsf{Obj}$.

Snapshot Isolation (SI) [6]. This consistency model strengthens PSI by requiring that, in executions, the set of transactions visible to any transaction T is a prefix of the arbitration relation. Formally, we let $\Sigma_{\mathsf{SI}} = \Sigma_{\mathsf{PSI}} \cup \{(\rho_{\mathsf{Id}}, \rho_{\mathsf{SI}})\}$. The consistency guarantee $(\rho_{\mathsf{Id}}, \rho_{\mathsf{SI}})$ ensures that any abstract execution $\mathcal{X} \in \mathsf{Executions}(\mathsf{SI})$ satisfies the property $(\mathsf{AR}_{\mathcal{X}} \,;\, \mathsf{VIS}_{\mathcal{X}}) \subseteq \mathsf{VIS}_{\mathcal{X}}$[3].

Similarly to what we did to specify Red-Blue consistency, we can strengthen SI by allowing the possibility to mark transactions as serialisable. The resulting x-specification is $\Sigma_{\mathsf{SI+SER}} = \Sigma_{\mathsf{SI}} \cup \{(\rho_S, \rho_S)\}$. This x-specification captures a fragment of Microsoft SQL server, which allows the user to select the consistency model at which a transaction should run [25].

Serialisability. Executions in this consistency model require the visibility relation to be total. This can be formalised via the x-specification $\Sigma_{\mathsf{SER}} := \{(\rho_{\mathsf{Id}}, \rho_{\mathsf{Id}})\}$. Any $\mathcal{X} \in \mathsf{Executions}(\Sigma_{\mathsf{SER}})$ is such that $\mathsf{AR}_{\mathcal{X}} \subseteq \mathsf{VIS}_{\mathcal{X}}$, thus enforcing $\mathsf{VIS}_{\mathcal{X}}$ to be a strict total order.

3 Dependency Graphs

We present another style of specification for consistency models based on dependency graphs, introduced in [1]. These are structures that capture the data-dependencies between transactions accessing one same object. Such dependencies can be over approximated at compilation time. For this reason, they have found use in static analysis [7, 16, 18, 19] for programs running under a weak consistency model.

▶ **Definition 7.** A ***dependency graph*** is a tuple $\mathcal{G} = (\mathcal{T}, \mathsf{WR}, \mathsf{WW}, \mathsf{RW})$, where $\mathcal{T}$ is a history and

1. $\mathsf{WR} : \mathsf{Obj} \to 2^{\mathcal{T} \times \mathcal{T}}$ is such that:
   (a) $\forall T, S \in \mathcal{T}. \forall x. T \xrightarrow{\mathsf{WR}(x)} S \implies T \neq S \wedge \exists n. (T \ni \texttt{write}\ x : n) \wedge (S \ni \texttt{read}\ x : n)$,
   (b) $\forall S \in \mathcal{T}. \forall x. (S \ni \texttt{read}\ x : n) \implies \exists T. T \xrightarrow{\mathsf{WR}(x)} S$,
   (c) $\forall T, T', S \in \mathcal{T}. \forall x. (T \xrightarrow{\mathsf{WR}(x)} S \wedge T' \xrightarrow{\mathsf{WR}(x)} S) \implies T = T'$;
2. $\mathsf{WW} : \mathsf{Obj} \to 2^{\mathcal{T} \times \mathcal{T}}$ is such that for every $x \in \mathsf{Obj}$, $\mathsf{WW}(x)$ is a strict, total order over Writes_x;
3. $\mathsf{RW} : \mathsf{Obj} \to 2^{\mathcal{T} \times \mathcal{T}}$ is such that $S \xrightarrow{\mathsf{RW}(x)} T$ iff $S \neq T$ and $\exists T'. T' \xrightarrow{\mathsf{WR}(x)} S \wedge T' \xrightarrow{\mathsf{WW}(x)} T$.

Given a dependency graph $\mathcal{G} = (\mathcal{T}, \mathsf{WR}, \mathsf{WW}, \mathsf{RW})$, we let $\mathcal{T}_{\mathcal{G}} := \mathcal{T}$, $\mathsf{WR}_{\mathcal{G}} := \mathsf{WR}$, $\mathsf{WW}_{\mathcal{G}} := \mathsf{WW}$, $\mathsf{RW}_{\mathcal{G}} := \mathsf{RW}$. The set of all dependency graphs is denoted as Graphs. Sometimes, we commit an abuse of notation and use the symbol WR to denote the relation $\bigcup_{x \in \mathsf{Obj}} \mathsf{WR}(x)$, and similarly for WW and RW. The actual meaning of WR will always be clear from the context.

Let $\mathcal{G} \in \mathsf{Graphs}$. The *write-read dependency* $T \xrightarrow{\mathsf{WR}_{\mathcal{G}}(x)} S$ means that S reads the value of object x that has been written by T. By Definition 7, for any transaction $S \in \mathsf{Reads}_x$ there exists exactly one transaction T such that $T \xrightarrow{\mathsf{WR}_{\mathcal{G}}(x)} S$. The relation $\mathsf{WW}_{\mathcal{G}}(x)$ establishes a total order in which updates over object x are executed by transactions; its elements are called *write-write dependencies*. Edges in the relation $\mathsf{RW}_{\mathcal{G}}(x)$ take the name of *anti-dependencies*. $T \xrightarrow{\mathsf{RW}_{\mathcal{G}}(x)} S$ means that transaction T fetches some value for object x, but this is later updated by S. Given an abstract execution $\mathcal{X}$, we can extract a dependency graph $\mathsf{graph}(\mathcal{X})$ such that $\mathcal{T}_{\mathsf{graph}(\mathcal{X})} = \mathcal{T}_{\mathcal{X}}$.

[3] To be precise, the property induced by the guarantee $(\rho_{\mathsf{Id}}, \rho_{\mathsf{SI}})$ is $(\mathsf{AR}_{\mathcal{X}} \,;\, (\mathsf{VIS}_{\mathcal{X}} \setminus \mathsf{Id})) \subseteq \mathsf{AR}_{\mathcal{X}}$. However, since $\mathsf{VIS}_{\mathcal{X}}$ is an irreflexive relation, $\mathsf{VIS}_{\mathcal{X}} \setminus \mathsf{Id} = \mathsf{VIS}_{\mathcal{X}}$. Also, note that $\rho(R) = R$ is not a specification function, so we cannot replace the guarantee $(\rho_{\mathsf{Id}}, \rho_{\mathsf{SI}})$ with $(\rho_{\mathsf{Id}}, \rho)$.

▶ **Definition 8.** Let $\mathcal{X} = (\mathcal{T}, \mathsf{VIS}, \mathsf{AR})$ be an execution. For $x \in \mathsf{Obj}$, we define $\mathsf{graph}(\mathcal{X}) = (\mathcal{T}, \mathsf{WR}_{\mathcal{X}}, \mathsf{WW}_{\mathcal{X}}, \mathsf{RW}_{\mathcal{X}})$, where:

1. $T \xrightarrow{\mathsf{WR}_{\mathcal{X}}(x)} S \iff (S \ni \mathtt{read}\ x : \_) \wedge T = \max_{\mathsf{AR}}(\mathsf{VIS}^{-1}(S) \cap \mathsf{Writes}_x)$;
2. $T \xrightarrow{\mathsf{WW}_{\mathcal{X}}(x)} S \iff T \xrightarrow{\mathsf{AR}} S \wedge T, S \in \mathsf{Writes}_x$;
3. $T \xrightarrow{\mathsf{RW}_{\mathcal{X}}(x)} S \iff S \neq T \wedge (\exists T'. T' \xrightarrow{\mathsf{WR}_{\mathcal{X}}(x)} T \wedge T' \xrightarrow{\mathsf{WW}_{\mathcal{X}}(x)} S))$.

▶ **Proposition 9.** *For any valid abstract execution $\mathcal{X}$, $\mathsf{graph}(\mathcal{X})$ is a dependency graph.*

Specification of Consistency Models using Dependency Graphs. We interpret a dependency graph $\mathcal{G}$ as a labelled graph whose vertices are transactions in $\mathcal{T}_x$, and whose edges are pairs of the form $T \xrightarrow{R} S$, where $R \in \{\mathsf{WR}_{\mathcal{G}}(x), \mathsf{WW}_{\mathcal{G}}(x)_{\mathcal{G}}, \mathsf{RW}_{\mathcal{G}}(x) \mid x \in \mathsf{Obj}\}$. To specify a consistency model, we employ a two-steps approach. We first identify one or more conditions to be satisfied by dependency graphs. Such conditions require cycles of a certain form not to appear in a dependency graph. Then we define a consistency model by projecting the set of dependency graphs satisfying the imposed conditions into the underlying histories. This style of specification is reminiscent of the one used in the CAT [4] language for formalising weak memory models. In the following we treat the relations $\mathsf{WR}_{\mathcal{G}}(x), \mathsf{WW}_{\mathcal{G}}(x), \mathsf{RW}_{\mathcal{G}}(x)$ both as set-theoretic relations, and as edges of a labelled graph.

▶ **Definition 10.** A *dependency graph based specification*, or simply g-specification, is a set $\Delta = \{\delta_1, \cdots, \delta_n\}$, where for each $i \in \{1, \cdots, n\}$, δ_i is a function of type $\mathsf{Graphs} \to 2^{(\mathbb{T} \times \mathbb{T})}$ and satisfies $\delta_i(\mathcal{G}) \subseteq (\mathsf{WR}_{\mathcal{G}} \cup \mathsf{WW}_{\mathcal{G}} \cup \mathsf{RW}_{\mathcal{G}})^*$ for every $\mathcal{G} \in \mathsf{Graphs}$.

Given a g-specification Δ, we define $\mathsf{Graphs}(\Delta) = \{\mathcal{G} \in \mathsf{Graphs} \mid \forall \delta \in \Delta. \delta(\mathcal{G}) \cap \mathsf{Id} = \emptyset\}$, and we let $\mathsf{modelOf}(\Delta) = \{\mathcal{T} \mid \exists \mathsf{WR}, \mathsf{WW}, \mathsf{RW}. (\mathcal{T}, \mathsf{WR}, \mathsf{WW}, \mathsf{RW}) \in \mathsf{Graphs}(\Delta)\}$.

The requirement imposed over the functions $\delta_1, \cdots, \delta_n$ ensures that, whenever $(T, S) \in \delta_i(\mathcal{G})$, for some dependency graph $\mathcal{G}$, then there exists a path in $\mathcal{G}$, that connects T to S. For $\Delta = \{\delta_i\}_{i=1}^n$ and $\mathcal{G} \in \mathsf{Graphs}$, the requirement that $\delta_i(\mathcal{G}) \cap \mathsf{Id} = \emptyset$ means that $\mathcal{G}$ does not contain any cycle $T_0 \xrightarrow{R_0} T_1 \xrightarrow{R_1} \cdots \xrightarrow{R_{n-1}} T_n$, such that $T_0 = T_n$, and $(R_0 ; \cdots ; R_{n-1}) \subseteq \delta_i(\mathcal{G})$.

Examples of g-specifications of consistency models. Below we give some examples of g-specifications for the consistency models presented in Section 2.

▶ **Theorem 11.**

1. *An execution $\mathcal{X}$ is serialisable iff* $\mathsf{graph}(\mathcal{X})$ *does not contain any cycle. That is,* $\mathsf{modelOf}(\Sigma_{\mathsf{SER}}) = \mathsf{modelOf}(\{\delta_{\mathsf{SER}}\})$*, where* $\delta_{\mathsf{SER}}(\mathcal{G}) = (\mathsf{WR}_{\mathcal{G}} \cup \mathsf{WW}_{\mathcal{G}} \cup \mathsf{RW}_{\mathcal{G}})^+$.
2. *An execution $\mathcal{X}$ is allowed by snapshot isolation iff* $\mathsf{graph}(\mathcal{X})$ *only admits cycles with at least two consecutive anti-dependency edge. That is,* $\mathsf{modelOf}(\Sigma_{\mathsf{SI}}) = \mathsf{modelOf}(\{\delta_{\mathsf{SI}}\})$*, where* $\delta_{\mathsf{SI}}(\mathcal{G}) = ((\mathsf{WR}_{\mathcal{G}} \cup \mathsf{WW}_{\mathcal{G}}) ; \mathsf{RW}_{\mathcal{G}}?)^+$.
3. *An execution $\mathcal{X}$ is allowed by parallel snapshot isolation iff* $\mathsf{graph}(\mathcal{X})$ *has no cycle where all anti-dependency edges are over the same object. Let* $\delta_{\mathsf{PSI}_0}(\mathcal{G}) = (\mathsf{WR}_{\mathcal{G}} \cup \mathsf{WW}_{\mathcal{G}})^+$, $\delta_{\mathsf{PSI}(x)}(\mathcal{G}) = (\bigcup_{x \in \mathsf{Obj}} (\mathsf{WR}_{\mathcal{G}} \cup \mathsf{WW}_{\mathcal{G}})^* ; \mathsf{RW}_{\mathcal{G}}(x))^+$*, and define* $\Delta_{\mathsf{PSI}} = \{\delta_{\mathsf{PSI}_0}\} \cup \{\delta_{\mathsf{PSI}(x)} \mid x \in \mathsf{Obj}\}$*. Then,* $\mathsf{modelOf}(\Sigma_{\mathsf{PSI}}) = \mathsf{modelOf}(\Delta_{\mathsf{PSI}})$.

Theorem 11(1) was proved in [1]. The only if condition of Theorem 11(2) was proved in [19]; we proved the if condition of Theorem 11(2) in [16]. Theorem 11(3) improves on the specification we gave for PSI in [16]; the latter does not have any constraints on the objects to which anti-dependencies refer to. We outline the proof of Theorem 11(3) in Section 5.

(a) Algebraic laws for sets of transactions		**(c)** Algebraic laws for abstract Executions	
(a.1)	$[\mathcal{T}'] \subseteq \mathsf{Id}$ **(a.2)** $[\mathcal{T}_1 \cap \mathcal{T}_2] = [\mathcal{T}_1] ; [\mathcal{T}_2]$	**(c.1)**	$\mathsf{WR}(x) \subseteq \mathsf{VIS}$ **(c.2)** $\mathsf{WW}(x) \subseteq \mathsf{AR}$
(a.3)	$(R_1 ; [\mathcal{T}']) \cap R_2 = (R_1 \cap R_2) ; [\mathcal{T}']$	**(c.3)**	$\mathsf{RW}(x) \subseteq \overline{\mathsf{VIS}^{-1}}$ **(c.4)** $\mathsf{VIS}^{+} \subseteq \mathsf{VIS}$
(a.4)	$([\mathcal{T}'] ; R_1) \cap R_2 = [\mathcal{T}'] ; (R \cap R_2)$	**(c.5)**	$\mathsf{AR}^{+} \subseteq \mathsf{AR}$ **(c.6)** $\mathsf{VIS} \subseteq \mathsf{AR}$
(b) Algebraic laws for (anti-)dependencies		**(c.7)**	$[\mathsf{Writes}_x] ; \mathsf{VIS} ; \mathsf{RW}(x) \subseteq \mathsf{AR}$
(b.1)	$\mathsf{WR}(x) \subseteq [\mathsf{Writes}_x] ; \mathsf{WR}(x) ; [\mathsf{Reads}_x]$	**(c.8)**	$\mathsf{VIS} ; \overline{\mathsf{VIS}^{-1}} \subseteq \overline{\mathsf{VIS}^{-1}}$
(b.2)	$\mathsf{WW}(x) \subseteq [\mathsf{Writes}_x] ; \mathsf{WW}(x) ; [\mathsf{Writes}_x]$	**(c.9)**	$\overline{\mathsf{VIS}^{-1}} ; \mathsf{VIS} \subseteq \overline{\mathsf{VIS}^{-1}}$
(b.3)	$\mathsf{RW}(x) \subseteq [\mathsf{Reads}_x] ; \mathsf{RW}(x) ; [\mathsf{Writes}_x]$	**(c.10)**	$(\overline{\mathsf{VIS}^{-1}} ; \mathsf{VIS}) \cap \mathsf{Id} \subseteq \emptyset$
(b.4)	$\mathsf{WR}(x) \subseteq \mathsf{WR}(x) \setminus \mathsf{Id}$	**(c.11)**	$(\mathsf{VIS} ; \overline{\mathsf{VIS}^{-1}}) \cap \mathsf{Id} \subseteq \emptyset$
(b.5)	$\mathsf{WW}(x) \subseteq \mathsf{WW}(x) \setminus \mathsf{Id}$	**(c.12)**	$\mathsf{AR} \cap \mathsf{Id} \subseteq \emptyset$
(b.6)	$\mathsf{RW}(x) \subseteq \mathsf{RW}(x) \setminus \mathsf{Id}$		

(d) Algebraic laws induced by the consistency guarantee (ρ, π)			
(d.1)	$\rho(\mathsf{VIS}) ; \mathsf{AR} ; \pi(\mathsf{VIS}) \subseteq \mathsf{VIS}$	**(d.2)**	$(\pi(\mathsf{VIS}) ; \overline{\mathsf{VIS}^{-1}} ; \rho(\mathsf{VIS})) \setminus \mathsf{Id} \subseteq \mathsf{AR}$
(d.3)	$(\mathsf{AR} ; \pi(\mathsf{VIS}) ; \overline{\mathsf{VIS}^{-1}}) \cap \rho(\mathcal{T} \times \mathcal{T})^{-1} \subseteq \overline{\mathsf{VIS}^{-1}}$		
(d.4)	$(\overline{\mathsf{VIS}^{-1}} ; \rho(\mathsf{VIS}) ; \mathsf{AR}) \cap \pi(\mathcal{T} \times \mathcal{T})^{-1} \subseteq \overline{\mathsf{VIS}^{-1}}$		

Figure 3 Algebraic laws satisfied by an abstract execution $\mathcal{X} = (\mathcal{T}, \mathsf{VIS}, \mathsf{AR})$. Here $\mathsf{graph}(\mathcal{X}) = (\mathcal{T}, \mathsf{WR}, \mathsf{WW}, \mathsf{RW})$. The inequalities in part **(d)** are valid under the assumption that $\mathcal{X} \in \mathsf{Executions}(\{(\rho, \pi)\})$.

4 Algebraic Laws for Weak Consistency

Having two different styles for specifying consistency models gives rise to the following problems:

Weak Correspondence Problem. Given a x-specification Σ, determine a non-trivial g-specification Δ which over-approximates Σ, that is such that $\mathsf{modelOf}(\Sigma) \subseteq \mathsf{modelOf}(\Delta)$.

Strong Correspondence Problem. Given a x-specification Σ, determine an equivalent g-specification Δ, that is such that $\mathsf{modelOf}(\Sigma) = \mathsf{modelOf}(\Delta)$.

We first focus on the weak correspondence problem, and we discuss the strong correspondence problem in Section 5. This problem is not only of theoretical interest. Determining a g-specification Δ that over-approximates a x-specification Σ corresponds to establishing one or more conditions satisfied by all cycles of dependency graphs from the set $\{\mathsf{graph}(\mathcal{X}) \mid \mathcal{X} \in \mathsf{Executions}(\Sigma)\}$. Cycles in a dependency graph that respect such a condition are called Σ-*critical* (or simply critical), and graphs that admit a non-Σ-critical cycle cannot be obtained from abstract executions in $\mathsf{Executions}(\Sigma)$. One can ensure that an application running under the model Σ is *robust*, i.e. it only produces serialisable behaviours, by checking for the absence of Σ-critical cycles at static time [7, 19]. Robustness of an application can also be checked at run-time, by incrementally constructing the dependency graph of executions, and detecting the presence of Σ-critical cycles [31].

General Methodology. Let Σ be a given x-specification. We tackle the weak correspondence problem in two steps.

First, we identify a set of inequalities that hold for all the executions $\mathcal{X}$ satisfying consistency guarantees (ρ, π) in Σ. There are two kinds of such inequalities. The first are the inequalities in Figure 3, and the second the inequalities corresponding to the axioms of the Kleene Algebra $(2^{\mathbb{T} \times \mathbb{T}}, \emptyset, \mathsf{Id}, \cup, ;, \cdot^{*})$ and the Boolean algebra $(2^{\mathbb{T} \times \mathbb{T}}, \emptyset, \mathbb{T} \times \mathbb{T}, \cup, \cap, \bar{\cdot})$. The exact meaning of the inequalities in Figure 3 is discussed later in this section.

Second, we exploit our inequalities to derive other inequalities of the form $R_{\mathcal{X}} \subseteq \mathsf{AR}_{\mathcal{X}}$ for every $\mathcal{X} \in \mathsf{Executions}(\Sigma)$. Here $R_{\mathcal{X}}$ is a relation built from dependencies in $\mathsf{graph}(\mathcal{X})$, i.e. $R_{\mathcal{X}} \subseteq (\mathsf{WR}_{\mathcal{X}} \cup \mathsf{WW}_{\mathcal{X}} \cup \mathsf{RW}_{\mathcal{X}})^*$. Because $\mathsf{AR}_{\mathcal{X}}$ is acyclic (that is $\mathsf{AR}_{\mathcal{X}}^+ \cap \mathsf{Id} \subseteq \varnothing$), we may conclude that $R_{\mathcal{X}}$ is acyclic for any $\mathcal{X} \in \mathsf{Executions}(\Sigma)$. In particular, we have that $\mathsf{modelOf}(\Sigma) \subseteq \mathsf{modelOf}(\{\delta\})$, where δ is a function that maps, for every abstract execution $\mathcal{X}$, the dependency graph $\mathsf{graph}(\mathcal{X})$ into the relation $R_{\mathcal{X}}$.

Some of the inequalities we develop, namely those in Figure 3**(d)**, are parametric in the consistency guarantee (ρ, π). As a consequence, our approach can be specialised to any consistency model that is captured by our framework. To show its applicability, we derive critical cycles for several of the consistency models that we have presented.

Presentation of the Laws. Let $\mathcal{X} = (\mathcal{T}, \mathsf{VIS}, \mathsf{AR})$, and $\mathsf{graph}(\mathcal{X}) = (\mathcal{T}, \mathsf{WR}, \mathsf{WW}, \mathsf{RW})$. We now explain the inequalities in Figure 3. Among these, the inequalities in Figures 3**(a)** and **(b)** should be self-explanatory.

Let us discuss the inequalities of Figure 3**(c)**. The inequalities **(c.1)**, **(c.2)** and **(c.3)** relate dependencies to either basic or derived relations of abstract executions. Dependencies of the form WR, WW are included in the relations $\mathsf{VIS}, \mathsf{AR}$, respectively, as established by inequalities **(c.1)** and **(c.2)**. The inequality **(c.3)**, which we prove presently, is non-standard. It relates anti-dependencies to a novel *anti-visibility* relation $\overline{\mathsf{VIS}^{-1}}$, defined as $T \xrightarrow{\overline{\mathsf{VIS}^{-1}}} S$ iff $\neg(S \xrightarrow{\mathsf{VIS}} T)$. In words, S is *anti-visible* to T if T does not observe the effects of S. As we will explain later, anti-visibility plays a fundamental role in the development of our laws.

Proof of Inequality (c.3). Suppose $T \xrightarrow{\mathsf{RW}(x)} S$ for some object $x \in \mathsf{Obj}$. By definition, $T \neq S$, and there exists a transaction T' such that $T' \xrightarrow{\mathsf{WR}(x)} T$ and $T' \xrightarrow{\mathsf{WW}(x)} S$. In particular, $T' \xrightarrow{\mathsf{VIS}} T$ and $T' \xrightarrow{\mathsf{AR}} S$ by the inequalities **(c.1)** and **(c.2)**, respectively. Now, if it were $S \xrightarrow{\mathsf{VIS}} T$, then we would have that T' is not the AR-supremum of the set of transactions visible to T, and writing to object x. But this contradicts the definition of $\mathsf{graph}(\mathcal{X})$, and the edge $T' \xrightarrow{\mathsf{WR}(x)} T$. Therefore, $T \xrightarrow{\overline{\mathsf{VIS}^{-1}}} S$. ◀

Another non-trivial inequality is **(c.7)** in Figure 3**(c)**. It says that if a transaction T reads a value for an object x that is later updated by another transaction S ($T \xrightarrow{\mathsf{RW}} S$), then the update of S is more recent (i.e. it follows in arbitration) than all the updates to x seen by T. We prove it in [17, Appendix C]. The other inequalities in Figure 3**(c)** are self explanatory.

The inequalities in Figure 3**(d)** are specific to a consistency guarantee (ρ, π), and hold for an execution $\mathcal{X}$ when the execution satisfies (ρ, π). The inequality **(d.1)** is just the definition of consistency guarantee. The next inequality **(d.2)** is where the novel anti-visibility relation, introduced previously, comes into play. While the consistency guarantee (ρ, π) expresses when arbitration induces transactions related by visibility, the inequality **(d.2)** expresses when anti-visibility induces transactions related by arbitration. To emphasise this correspondence, we call the inequality **(d.2)** *co-axiom* induced by (ρ, π). Later in this section, we show how by exploiting the co-axiom induced by several consistency guarantees, we can derive critical cycles of several consistency models.

Proof of Inequality (d.2). Assume $\mathcal{X} \in \mathsf{Executions}(\{(\rho, \pi)\})$. Let $T, T', S', S \in \mathcal{T}$ be such that $T \neq S$, $T \xrightarrow{\pi(\mathsf{VIS})} T' \xrightarrow{\overline{\mathsf{VIS}^{-1}}} S' \xrightarrow{\rho(\mathsf{VIS})} S$. Because AR is total, either $S \xrightarrow{\mathsf{AR}} T$ or $T \xrightarrow{\mathsf{AR}} S$. However, the former case is not possible. If so, we would have $S' \xrightarrow{\rho(\mathsf{VIS})} S \xrightarrow{\mathsf{AR}} T \xrightarrow{\pi(\mathsf{VIS})} T'$.

because $\mathcal{X} \in \mathsf{Executions}(\{(\rho, \pi)\})$, by the inequality **(d.1)**, it would follow that $S' \xrightarrow{\mathsf{VIS}} T'$, contradicting the assumption that $T' \xrightarrow{\mathsf{VIS}^{-1}} S'$. Therefore, it has to be $T \xrightarrow{\mathsf{AR}} S$. ◀

The last inequalities **(d.3)** and **(d.4)** in Figure 3**(d)** show that anti-visibility edges of $\mathcal{X}$ are also induced by the consistency guarantee (ρ, π). We prove them formally in [17, Appendix C], where we also illustrate some of their applications.

Applications. We employ the algebraic laws of Figure 3 to derive Σ-critical cycles for arbitrary x-specifications, using the methodology explained previously: given a x-specification Σ and an abstract execution $\mathcal{X}$, we characterise a subset of $\mathsf{AR}_{\mathcal{X}}$ as a relation R_{G} built from the dependencies in $\mathsf{graph}(\mathcal{X})$ and relations of the form $[o]$, where $o \in \mathsf{Op}$. Because $R_{\mathsf{G}} \subseteq \mathsf{AR}_{\mathcal{X}}$, we conclude that R_{G} is acyclic.

The inequalities **(c.1)**, **(c.6)** and **(c.2)** ensure that we can always include write-read and write-write dependencies in the relation R_{G} above. Because of inequalities **(c.3)** and **(d.2)** (among others), we can include in R_{G} also relations that involve anti-dependencies. The following result shows how this methodology can be applied to serialisability. We use the notation $R_1 \overset{(\mathbf{eq})}{\subseteq} R_2$ to denote that the inequality $R_1 \subseteq R_2$ follows from $(\mathbf{eq})$.

▶ **Theorem 12.** *For all* $\mathcal{X} \in \mathsf{Executions}(\Sigma_{\mathsf{SER}})$*, the relation* $(\mathsf{WR}_{\mathcal{X}} \cup \mathsf{WW}_{\mathcal{X}} \cup \mathsf{RW}_{\mathcal{X}})$ *is acyclic.*

Proof. Recall that $\Sigma_{\mathsf{SER}} = \{(\rho_{\mathsf{Id}}, \rho_{\mathsf{Id}})\}$, where $\rho_{\mathsf{Id}}(\_) = \mathsf{Id}$. We have

$$\mathsf{RW}_{\mathcal{X}} \overset{(\mathbf{b.6})}{\subseteq} \mathsf{RW}_{\mathcal{X}} \backslash \mathsf{Id} \overset{(\mathbf{c.3})}{\subseteq} \overline{\mathsf{VIS}_{\mathcal{X}}^{-1}} \backslash \mathsf{Id} = (\rho_{\mathsf{Id}}(\mathsf{VIS}_{\mathcal{X}}) \,;\, \overline{\mathsf{VIS}_{\mathcal{X}}^{-1}} \,;\, \rho_{\mathsf{Id}}(\mathsf{VIS}_{\mathcal{X}})) \backslash \mathsf{Id} \overset{(\mathbf{d.2})}{\subseteq} \mathsf{AR}_{\mathcal{X}} \tag{1}$$

$$(\mathsf{WR}_{\mathcal{X}} \cup \mathsf{WW}_{\mathcal{X}} \cup \mathsf{RW}_{\mathcal{X}}) \overset{(\mathbf{c.1,c.6})}{\subseteq} (\mathsf{AR}_{\mathcal{X}} \cup \mathsf{WW}_{\mathcal{X}} \cup \mathsf{RW}_{\mathcal{X}}) \overset{(\mathbf{c.2})}{\subseteq} (\mathsf{AR}_{\mathcal{X}} \cup \mathsf{RW}_{\mathcal{X}}) \overset{(1)}{\subseteq} \mathsf{AR}_{\mathcal{X}} \tag{2}$$

$$(\mathsf{WR}_{\mathcal{X}} \cup \mathsf{WW}_{\mathcal{X}} \cup \mathsf{RW}_{\mathcal{X}})^{+} \cap \mathsf{Id} \overset{(2)}{\subseteq} \mathsf{AR}_{\mathcal{X}}^{+} \cap \mathsf{Id} \overset{(\mathbf{c.5})}{\subseteq} \mathsf{AR}_{\mathcal{X}} \cap \mathsf{Id} \overset{(\mathbf{c.12})}{\subseteq} \varnothing.$$ ◀

Along the lines of the proof of Theorem 12, we can characterise Σ-critical cycles for an arbitrary x-specification Σ. Below, we show how to apply our methodology to derive Σ_{RB}-critical cycles.

▶ **Theorem 13.** *Let* $\mathcal{X} \in \mathsf{Executions}(\Sigma_{\mathsf{RB}})$*. Say that a* $\mathsf{RW}_{\mathcal{X}}$ *edge in a cycle of* $\mathsf{graph}(\mathcal{X})$ *is* protected *if its endpoints are connected to serialisable transactions via a sequence of* $\mathsf{WR}_{\mathcal{X}}$ *edges. Then all cycles in* $\mathsf{graph}(\mathcal{X})$ *have at least one unprotected* $\mathsf{RW}_{\mathcal{X}}$ *edge. Formally, let* $\Vdash\!\mathsf{RW}_{\mathcal{X}}\!\dashv\!\mid$ *be* $([\mathtt{SerTx}] \,;\, (\mathsf{WR}_{\mathcal{X}})^{*} \,;\, \mathsf{RW}_{\mathcal{X}} \,;\, (\mathsf{WR}_{\mathcal{X}})^{*} \,;\, [\mathtt{SerTx}])$*. Then* $(\mathsf{WR}_{\mathcal{X}} \cup \mathsf{WW}_{\mathcal{X}} \cup \Vdash\!\mathsf{RW}_{\mathcal{X}}\!\dashv\!\mid)$ *is acyclic.*

$$
\left\{
\begin{array}{llllll}
\mathsf{WR} \subseteq X_V & \text{(V1)} & X_V \,;\, X_V \subseteq X_V & \text{(V2)} & \displaystyle\bigcup_{\{x \mid (\rho_x, \rho_x) \in \Sigma\}} \mathsf{WW}(x) \subseteq X_V & \text{(V3)} \\
 & & & & \rho(X_V) \,;\, X_A \,;\, \pi(X_V) \subseteq X_V & \text{(V4)} \\
\mathsf{WW} \subseteq X_A & \text{(A1)} & X_V \subseteq X_A & \text{(A2)} & \displaystyle\bigcup_{x \in \mathsf{Obj}} ([\mathsf{Writes}_x] \,;\, X_V \,;\, \mathsf{RW}(x)) \subseteq X_A & \text{(A3)} \\
 & & X_A \,;\, X_A \subseteq X_A & \text{(A4)} & (\pi(X_V) \,;\, X_N \,;\, \rho(X_V)) \setminus \mathsf{Id} \subseteq X_A & \text{(A5)} \\
\mathsf{RW} \subseteq X_N & \text{(N1)} & X_V \,;\, X_N \subseteq X_N & \text{(N2)} & X_N \,;\, X_V \subseteq X_N & \text{(N3)}
\end{array}
\right.
$$

Figure 4 The system of inequalities $\mathsf{System}_\Sigma(\mathcal{G})$ for the simple consistency model Σ and the dependency graph $\mathcal{G} = (\mathcal{T}, \mathsf{WR}, \mathsf{WW}, \mathsf{RW})$.

Proof. It suffices to prove that $\Vdash\!\mathsf{RW}_\mathcal{X}\!\dashv\!\mid \subseteq \mathsf{AR}_\mathcal{X}$. The rest of the proof is similar to the one of Theorem 12. We recall that $\Sigma_{\mathsf{RB}} = \{(\rho_S, \rho_S)\}$, where $\rho_S(\_) = [\mathtt{SerTx}]$.

$$
\begin{aligned}
&\mathsf{WR}^*_\mathcal{X} \,;\, \mathsf{RW}_\mathcal{X} \,;\, \mathsf{WR}^*_\mathcal{X} \overset{\textbf{(c.1,c.4)}}{\subseteq} \mathsf{VIS}_\mathcal{X}? \,;\, \mathsf{RW}_\mathcal{X} \,;\, \mathsf{VIS}_\mathcal{X}? \overset{\textbf{(b.6)}}{\subseteq} \mathsf{VIS}_\mathcal{X}? \,;\, (\mathsf{RW}_\mathcal{X} \setminus \mathsf{Id}) \,;\, \mathsf{VIS}_\mathcal{X}? \overset{\textbf{(c.3)}}{\subseteq} \\
&\mathsf{VIS}_\mathcal{X}? \,;\, (\overline{\mathsf{VIS}^{-1}_\mathcal{X}} \setminus \mathsf{Id}) \,;\, \mathsf{VIS}_\mathcal{X}? \subseteq ((\overline{\mathsf{VIS}^{-1}_\mathcal{X}} \setminus \mathsf{Id}) \cup (\mathsf{VIS}_\mathcal{X} \,;\, \overline{\mathsf{VIS}^{-1}_\mathcal{X}})) \,;\, \mathsf{VIS}_\mathcal{X}? \overset{\textbf{(c.11)}}{\subseteq} \\
&((\overline{\mathsf{VIS}^{-1}_\mathcal{X}} \setminus \mathsf{Id}) \cup (\mathsf{VIS}_\mathcal{X} \,;\, \overline{\mathsf{VIS}^{-1}_\mathcal{X}}) \setminus \mathsf{Id}) \,;\, \mathsf{VIS}_\mathcal{X}? \overset{\textbf{(c.8)}}{\subseteq} (\overline{\mathsf{VIS}^{-1}_\mathcal{X}} \setminus \mathsf{Id}) \,;\, \mathsf{VIS}_\mathcal{X}? \overset{\textbf{(c.10,c.9)}}{\subseteq} \overline{\mathsf{VIS}^{-1}_\mathcal{X}} \setminus \mathsf{Id} \qquad (3) \\
&[\mathtt{SerTx}] \,;\, (\overline{\mathsf{VIS}^{-1}_\mathcal{X}} \setminus \mathsf{Id}) \,;\, [\mathtt{SerTx}] \overset{\textbf{(a.3,a.4)}}{=} ([\mathtt{SerTx}] \,;\, \overline{\mathsf{VIS}^{-1}_\mathcal{X}} \,;\, [\mathtt{SerTx}]) \setminus \mathsf{Id} = \\
&(\rho_S(\mathsf{VIS}_\mathcal{X}) \,;\, \overline{\mathsf{VIS}^{-1}_\mathcal{X}} \,;\, \rho_S(\mathsf{VIS}_\mathcal{X})) \setminus \mathsf{Id} \overset{\textbf{(d.2)}}{\subseteq} \mathsf{AR}_\mathcal{X} \qquad (4) \\
&\Vdash\!\mathsf{RW}_\mathcal{X}\!\dashv\!\mid = [\mathtt{SerTx}] \,;\, \mathsf{WR}^*_\mathcal{X} \,;\, \mathsf{RW}_\mathcal{X} \,;\, \mathsf{WR}^*_\mathcal{X} \,;\, [\mathtt{SerTx}] \overset{(3,4)}{\subseteq} \mathsf{AR}_\mathcal{X}.
\end{aligned}
$$
◀

We remark that our characterisation of Σ_{RB}-critical cycle cannot be compared to the one given in [7]. In [17, Appendix C] we show how our methodology can be applied to give a characterisation of Σ_{RB}-critical cycles that is stronger than both the one presented in Theorem 13 and the one given in [7]. We also employ our proof technique to prove both known and new derivations of critical cycles for other x-specifications.

5 Characterisation of Simple Consistency Models

We now turn our attention to the *Strong Correspondence Problem* presented in Section 4. Given a x-specification $\Sigma = \{(\rho_1, \pi_1), \cdots, (\rho_n, \pi_n)\}$ and a dependency graph $\mathcal{G}$, we want to find a sufficient and necessary condition for determining whether $\mathcal{G} = \mathsf{graph}(\mathcal{X})$ for some $\mathcal{X} \in \mathsf{Executions}(\Sigma)$.

In this section we propose a proof technique for solving the strong correspondence problem. This technique applies to a particular class of x-specifications, which we call *simple* x-specifications. This class includes several of the consistency models we have presented.

Characterisation of Simple x-specifications. Recall that for each $x \in \mathsf{Obj}$, the function ρ_x of an abstract execution $\mathcal{X}$ is defined as $\rho_x(\_) = [\mathsf{Writes}_x]$, and the associated axiom is $[\mathsf{Writes}_x] \,;\, \mathsf{AR}_\mathcal{X} \,;\, [\mathsf{Writes}_x] \subseteq \mathsf{VIS}_\mathcal{X}$.

▶ **Definition 14.** A x-specification Σ is *simple* if there exists a consistency guarantee (ρ, π) such that $\Sigma \subseteq \{(\rho, \pi)\} \cup \{(\rho_x, \rho_x)\}_{x \in \mathsf{Obj}}$.

That is, a simple x-specification Σ contains at most one consistency guarantee, beside those of the form (ρ_x, ρ_x) which express the write-conflict detection for some object $x \in \mathsf{Obj}$. Among the x-specifications that we have presented in this paper, the only non-simple one is $\Sigma_{\mathsf{SI+SER}}$.

For simple x-specifications, it is possible to solve the strong correspondence problem. Fix a simple x-specification $\Sigma \subseteq \{(\rho, \pi)\} \cup \{(\rho_x, \rho_x) \mid x \in \mathsf{Obj}\}$ and a dependency graph $\mathcal{G}$. We define a system of inequalities $\mathsf{System}_\Sigma(\mathcal{G})$ in three unknowns X_V, X_A and X_N, and depicted in Figure 4 (the inequalities (V4) and (A5) are included in the system if and only if $(\rho, \pi) \in \Sigma$). These unknowns correspond to subsets of the visibility, arbitration and anti-visibility relations of the abstract execution $\mathcal{X} \in \mathsf{Executions}(\Sigma)$, with underlying dependency graph $\mathcal{G}$, that we wish to find. Note that each one of the inequalities of $\mathsf{System}_\Sigma(\mathcal{G})$, with the exception of (V3), follows the structure of one of the algebraic laws from Figure 3. We prove that, in order to ensure that the abstract execution $\mathcal{X}$ exists, it is sufficient to find a solution of $\mathsf{System}_\Sigma(\mathcal{G})$ whose X_A-component is acyclic. In particular, this is true if and only if the X_A-component of the smallest solution[4] of $\mathsf{System}_\Sigma(\mathcal{G})$ is acyclic.

▶ **Theorem 15.**

Soundness: *for any* $\mathcal{X} \in \mathsf{Executions}(\Sigma)$ *such that* $\mathsf{graph}(\mathcal{X}) = \mathcal{G}$*, the triple* $(X_V = \mathsf{VIS}_\mathcal{X}, X_A = \mathsf{AR}_\mathcal{X}, X_N = \overline{\mathsf{VIS}_\mathcal{X}^{-1}})$ *is a solution of* $\mathsf{System}_\Sigma(\mathcal{G})$*,*

Completeness: *Let* $(X_V = \mathsf{VIS}_0, X_A = \mathsf{AR}_0, X_N = \mathsf{AntiVIS}_0)$ *be the smallest solution of* $\mathsf{System}_\Sigma(\mathcal{G})$*. If* AR_0 *is acyclic, then there exists an abstract execution* $\mathcal{X}$ *such that* $\mathcal{X} \in \mathsf{Executions}(\Sigma)$ *and* $\mathsf{graph}(\mathcal{X}) = \mathcal{G}$*.* ◀

Note that the relation AR_0 need not to be total in the completeness direction of Theorem 15.

Before discussing the proof of Theorem 15, we show how it can be used to prove the equivalence of a x-specification and a g-specification. We give a proof of Theorem 11(3). Theorems 11(1) and 11(2) can be proved similarly, and their proof is given in [17, Appendix D].

Proof Sketch of Theorem 11(3). Recall that $\Delta_{\mathsf{PSI}} = \{\delta_{\mathsf{PSI}_0}\} \cup \{\delta_{\mathsf{PSI}(x)}(\mathcal{G}) \mid x \in \mathsf{Obj}\}$, where $\delta_{\mathsf{PSI}_0}(\mathcal{G}) = (\mathsf{WR}_\mathcal{G} \cup \mathsf{WW}_\mathcal{G})^+$, $\delta_{\mathsf{PSI}(x)}(\mathcal{G}) = ((\mathsf{WR}_\mathcal{G} \cup \mathsf{WW}_\mathcal{G})^* ; \mathsf{RW}_\mathcal{G}(x))^+$. In [17, Appendix D] we prove that $\mathsf{Graphs}(\Delta_{\mathsf{PSI}}) = \mathsf{Graphs}(\{\delta_{\mathsf{PSI}}\})$, where

$$\delta_{\mathsf{PSI}}(\mathcal{G}) = (\mathsf{WR}_\mathcal{G} \cup \mathsf{WW}_\mathcal{G})^+ \cup \bigcup_{x \in \mathsf{Obj}} ([\mathsf{Writes}_x] ; (\mathsf{WR}_\mathcal{G} \cup \mathsf{WW}_\mathcal{G})^* ; \mathsf{RW}_\mathcal{G}(x))^+ .$$

Therefore, it suffices to prove that $\mathsf{modelOf}(\Sigma_{\mathsf{PSI}}) = \mathsf{modelOf}(\{\delta_{\mathsf{PSI}}\})$:

modelOf$(\Sigma_{\mathsf{PSI}}) \subseteq$ **modelOf**$(\{\delta_{\mathsf{PSI}}\})$: given $\mathcal{X} \in \mathsf{Executions}(\Sigma_{\mathsf{PSI}})$, and let $\mathcal{G} := \mathsf{graph}(\mathcal{X})$, we need to show that $\delta_{\mathsf{PSI}}(\mathcal{G}) \cap \mathsf{Id} = \emptyset$. The proof follows the style of Theorems 12 and 13; details can be found in [17, Appendix C],

modelOf$(\{\delta_{\mathsf{PSI}}\}) \subseteq$ **modelOf**(Σ_{PSI}): given $\mathcal{G} \in \mathsf{Graphs}(\{\delta_{\mathsf{PSI}}\})$, let $\mathsf{VIS}_\mathcal{G} = (\mathsf{WR} \cup \mathsf{WW})^+$; It is immediate to prove that the triple $(X_V = \mathsf{VIS}_\mathcal{G}, X_A = \delta_{\mathsf{PSI}}(\mathcal{G}), X_N = \mathsf{VIS}_\mathcal{G}? ; \mathsf{RW} ; \mathsf{VIS}_\mathcal{G}?)$ is a solution of $\mathsf{System}_{\Sigma_{\mathsf{PSI}}}(\mathcal{G})$. Because $\delta_{\mathsf{PSI}}(\mathcal{G})$ is acyclic, if we take the smallest solution $(X_V = \_, X_A = \mathsf{AR}_\mathcal{G}, X_N = \_)$ of $\mathsf{System}_\Sigma(\mathcal{G})$, then $\mathsf{AR}_\mathcal{G} \subseteq \delta_{\mathsf{PSI}}(\mathcal{G})$, hence $\mathsf{AR}_\mathcal{G}$ is acyclic. By Theorem 15, there exists an abstract execution $\mathcal{X} \in \mathsf{Executions}(\mathsf{PSI})$ such that $\mathsf{graph}(\mathcal{X}) = \mathcal{G}$, and in particular $\mathcal{T}_\mathcal{X} = \mathcal{T}_\mathcal{G}$. ◀

[4] A solution $(X_V = \mathsf{VIS}, X_A = \mathsf{AR}, X_N = \mathsf{AntiVIS})$ is smaller than another one $(X_V = \mathsf{VIS}', X_A = \mathsf{AR}', X_N = \mathsf{AntiVIS}')$ iff $\mathsf{VIS} \subseteq \mathsf{VIS}', \mathsf{AR} \subseteq \mathsf{AR}'$ and $\mathsf{AntiVIS} \subseteq \mathsf{AntiVIS}'$.

We now turn our attention to the proof of Theorem 15. The proof of the soundness direction is straightforward.

Proof of Theorem 15 (Soundness). Let $\mathcal{X} \in \mathsf{Executions}(\Sigma)$, and define $\mathcal{G} := \mathsf{graph}(\mathcal{X})$. To show that the triple $(X_V = \mathsf{VIS}_{\mathcal{X}}, X_A = \mathsf{AR}_{\mathcal{X}}, X_N = \overline{\mathsf{VIS}_{\mathcal{X}}^{-1}})$ is a solution of $\mathsf{System}_{\Sigma}(\mathcal{G})$, we need to show that all the inequalities from said system are satisfied, when the unknowns X_A, X_V, X_N are replaced with $\mathsf{VIS}_{\mathcal{X}}, \mathsf{AR}_{\mathcal{X}}, \overline{\mathsf{VIS}_{\mathcal{X}}^{-1}}$, respectively. In practice, all the inequalities, with the exception of (V3), follow from the algebraic laws of Figure 3. Let us prove that (V3) is also valid: for any $(\rho_x, \rho_x) \in \Sigma$ we have that

$$\mathsf{WW}_{\mathcal{X}}(x) \overset{\textbf{(b.2)}}{=} [\mathsf{Writes}_x] \,;\, \mathsf{WW}_{\mathcal{X}}(x) \,;\, [\mathsf{Writes}_x] \overset{\textbf{(c.2)}}{\subseteq} [\mathsf{Writes}_x] \,;\, \mathsf{AR}_{\mathcal{X}} \,;\, [\mathsf{Writes}_x] \overset{\textbf{(d.1)}}{\subseteq} \mathsf{VIS}_{\mathcal{X}}.$$

◀

The proof of the completeness direction of Theorem 15 is much less straightforward. Let $(X_V = \mathsf{VIS}_0, X_A = \mathsf{AR}_0, X_N = \mathsf{AntiVIS}_0)$ be the smallest solution of $\mathsf{System}_{\Sigma}(\mathcal{G})$. Assume that AR_0 is acyclic. The challenge is that of constructing a valid abstract execution $\mathcal{X}$, i.e. whose arbitration order is total, from the dependencies in $\mathcal{G}$, that is included in $\mathsf{Executions}(\Sigma)$. We do this incrementally: at intermediate stages of the construction we get structures similar to abstract executions, but where the arbitration order can be partial.

► **Definition 16.** A *pre-execution* $\mathcal{P} = (\mathcal{T}_{\mathcal{G}}, \mathsf{VIS}, \mathsf{AR})$ is a tuple that satisfies all the constraints of abstract executions, except that AR is not necessarily total, although AR is still required to be total over the set Writes_x for every object x.

The notation adopted for abstract executions naturally extends to pre-executions; also, for any pre-execution $\mathcal{P}$, $\mathsf{graph}(\mathcal{P})$ is a well-defined dependency graph. Given a x-specification Σ, we let $\mathsf{PreExecutions}(\Sigma)$ be the set of all valid pre-executions that satisfy all the consistency guarantees in Σ.

$\mathsf{System}_{\Sigma}(\mathcal{G})$ is defined so that all of its solutions whose X_A-component is acyclic induce a valid pre-execution in $\mathsf{PreExecutions}(\Sigma)$ with underlying dependency graph $\mathcal{G}$.

► **Proposition 17.** *Let* $(X_V = \mathsf{VIS}', X_A = \mathsf{AR}', X_N = \mathsf{AntiVIS}')$ *be a solution to* $\mathsf{System}_{\Sigma}(\mathcal{G})$*. If* $\mathsf{AR}' \cap \mathsf{Id} = \emptyset$*, then* $\mathcal{P} = (\mathcal{T}_{\mathcal{G}}, \mathsf{VIS}', \mathsf{AR}') \in \mathsf{PreExecutions}(\Sigma)$*; moreover,* $\mathsf{graph}(\mathcal{P}) = \mathcal{G}$*.*

Proof Sketch. The inequalities (A1), (A2) and (A4) together with the assumption that AR_0 is acyclic, ensure that $\mathcal{P}$ is a pre-execution. In particular, (A1) ensures that AR_0 is a total relation over the set Writes_x, for any $x \in \mathsf{Obj}$. As we explain in [17, Appendix D], the inequalities (V1), (A1) and (A3) enforce the Last Write Wins policy (Definition 3). The inequality (V2) mandates that $\mathcal{P}$ respects causality. Finally, the inequalities (V3) and (V4) ensure that all the consistency guarantees in Σ are satisfied by $\mathcal{P}$. ◀

In particular, the smallest solution $(X_V = \mathsf{VIS}_0, X_A = \mathsf{AR}_0, X_N = \mathsf{AntiVIS}_0)$ of $\mathsf{System}_{\Sigma}(\mathcal{G})$ induces the pre-execution $(\mathcal{T}_{\mathcal{G}}, \mathsf{VIS}_0, \mathsf{AR}_0) \in \mathsf{PreExecutions}(\Sigma)$.

To construct an abstract execution $\mathcal{X} \in \mathsf{Executions}(\Sigma)$, with $\mathsf{graph}(\mathcal{X}) = \mathcal{G}$, we define a finite chain of pre-executions $\{\mathcal{P}_i,\}_{i=0}^{n}$, $n \geqslant 0$, as follows: **(i)** let $\mathcal{P}_0 := (\mathcal{T}_{\mathcal{G}}, \mathsf{VIS}_0, \mathsf{AR}_0)$; **(ii)** given $\mathcal{P}_i$, $i \geqslant 0$, choose two different transactions $T_i, S_i \in \mathcal{T}_{\mathcal{G}}$ (if any) that are not related by AR_i, compute the smallest solution $(X_V = \mathsf{VIS}_{i+1}, X_A = \mathsf{AR}_{i+1}, X_N = \_)$ such that $\mathsf{AR}_{i+1} \supseteq \mathsf{AR}_i \cup \{(T_i, S_i)\}$, and let $\mathcal{P}_{i+1} := (\mathcal{T}_{\mathcal{G}}, \mathsf{VIS}_{i+1}, \mathsf{AR}_{i+1})$; **(iii)** if the transactions $T_i, S_i \in \mathcal{T}_{\mathcal{G}}$ from the previous step do not exist, then let $n := i$ and terminate the construction. Because we are assuming that $\mathcal{T}_{\mathcal{G}}$ is finite, the construction of $\{\mathcal{P}_0, \cdots, \mathcal{P}_n\}$ always terminates.

To prove the completeness direction of Theorem 15, we show that all of the pre-executions $\{\mathcal{P}_0, \cdots, \mathcal{P}_n\}$ in the construction outlined above are included in $\mathsf{PreExecutions}(\Sigma)$; then, because in $\mathcal{P}_n = (\mathcal{T}_\mathcal{G}, \mathsf{VIS}_n, \mathsf{AR}_n)$ all transactions are related by AR_n, we may conclude that AR_n is total, and $\mathcal{P}_n \in \mathsf{Executions}(\Sigma)$. According to Proposition 17, it suffices to show that each of the relations $\mathsf{AR}_i, i = 0, \cdots, n$ is acyclic. However, this is not completely trivial, because of how AR_{i+1} is defined: adding one edge (T_i, S_i) in AR_{i+1} may cause more edges to be included in VIS_{i+1}, due to the inequality (V4). This in turn leads to including more edges in AR_{i+1}, thus augmenting the risk of having a cycle in AR_{i+1}.

In practice, the definition of $\mathsf{System}_\Sigma(\mathcal{G})$ ensures that this scenario does not occur.

▶ **Proposition 18.** *For $i = 0, \cdots, n-1$, let $\Delta\mathsf{AR}_i := \mathsf{AR}_i? \; ; \; \{(T_i, S_i)\} \; ; \; \mathsf{AR}_i?$. Then $\mathsf{AR}_{i+1} = \mathsf{AR}_i \cup \Delta\mathsf{AR}_i$.*

▶ **Corollary 19.** *For $i = 0, \cdots, n-1$, if $\mathsf{AR}_i \cap \mathsf{Id} = \emptyset$, then $\mathsf{AR}_{i+1} \cap \mathsf{Id} = \emptyset$.*

Proof. Because $\mathsf{AR}_i \cap \mathsf{Id} = \emptyset$ by hypothesis, by Proposition 18 we only need to show that $\Delta\mathsf{AR}_i \cap \mathsf{Id} = \emptyset$. If $(T, T) \in \Delta\mathsf{AR}_i$ for some $T \in \mathcal{T}_\mathcal{G}$, then it must be $T \xrightarrow{\mathsf{AR}_i?} T_i$ and $S_i \xrightarrow{\mathsf{AR}_i?} T$. It follows that $S_i \xrightarrow{\mathsf{AR}_i?} T_i$. But this contradicts the hypothesis that AR_i does not relate transactions T_i and S_i. Therefore, $(T, T) \notin \Delta\mathsf{AR}_i$ for any $T \in \mathcal{T}_\mathcal{G}$, i.e. $\Delta\mathsf{AR}_i \cap \mathsf{Id} = \emptyset$. ◀

We have now everything in place to prove Theorem 15.

Proof of Theorem 15 (Completeness). Let $\mathcal{G}$ be a dependency graph, and define the chain of pre-executions $\mathcal{P}_0 = (\mathcal{T}_\mathcal{G}, \mathsf{VIS}_0, \mathsf{AR}_0), \cdots, \mathcal{P}_n = (\mathcal{T}_\mathcal{G}, \mathsf{VIS}_n, \mathsf{AR}_n)$ as described above. We show that for any $i = 0, \cdots, n$, $\mathcal{P}_i \in \mathsf{PreExecutions}(\Sigma)$, and $\mathsf{graph}(\mathcal{P}_i) = \mathcal{G}$. Because AR_n is a total order, this implies that $\mathcal{P}_n \in \mathsf{Executions}(\Sigma)$, and $\mathsf{graph}(\mathcal{P}_n) = \mathcal{G}$, as we wanted to prove. The proof is by induction on n.

Case $i = 0$: observe that the triple $(X_V = \mathsf{VIS}_0, X_A = \mathsf{AR}_0, X_N = \_)$ corresponds to the smallest solution of $\mathsf{System}_\Sigma(\mathcal{G})$, hence AR_0 is acyclic by hypothesis. It follows from Proposition 17 that $\mathcal{P}_0 \in \mathsf{PreExecutions}(\Sigma)$, and $\mathsf{graph}(\mathcal{P}_0) = \mathcal{G}$,

Case $i > 0$: assume that $i \leqslant n$; then $i - 1 < n$, and by induction hypothesis $\mathcal{P}_{i-1} \in \mathsf{PreExecutions}(\Sigma)$. In particular, the relation AR_{i-1} is acyclic; by Corollary 19 we obtain that AR_i is acyclic. Finally, recall that the triple $(X_V = \mathsf{VIS}_i, X_A = \mathsf{AR}_i, X_N = \_)$ is a solution of $\mathsf{System}_\Sigma(\mathcal{G})$ by construction. It follows from Proposition 17 that $\mathcal{P}_i \in \mathsf{PreExecutions}(\Sigma)$, and $\mathsf{graph}(\mathcal{P}_i) = \mathcal{G}$. ◀

6 Conclusion

We have explored the connection between two different styles of specifications for weak consistency models at an algebraic level. We have proposed several laws which we applied to devise several robustness criteria for consistency models. To the best of our knowledge, this is the first generic proof technique for proving robustness criteria of weak consistency models. We have shown that, for a particular class of consistency models, our algebraic approach leads to a precise characterisation of consistency models in terms of dependency graphs.

Related Work. Abstract executions have been introduced by Burckhardt in [12] to model the behaviour of eventually consistent data-stores; They have been used to capture the behaviour of replicated data types (Gotsman et al. [13]), geo-replicated databases (Cerone et al. [15]) and non-transactional distributed storage systems (Viotti et al. [30]).

Dependency graphs have been introduced by Adya [1]; they have been used since to reason about programs running under weak consistency models. Bernardi et al., used dependency graphs to derive robustness criteria of several consistency models [7], including PSI and red-blue; in contrast with our work, the proofs there contained do not rely on a general technique. Brutschy et al. generalised the notion of dependency graphs to replicated data types, and proposed a robustness criterion for eventual consistency [10].

Weak consistency also arises in the context of shared memory systems [4]. Alglave et al., proposed the CAT language for specifying weak memory models in [4], which also specifies weak memory models as a set of irreflexive relations over data-dependencies of executions. Castellan [14], and Jeffrey et al. [21], proposed different formalisations of weak memory models via event structures. The problem of checking the robustness of applications has also been addressed for weak memory models [2, 3, 8].

The strong correspondence problem (Section 5) is also highlighted by Bouajjani et al. in [9]: there the authors emphasize the need for general techniques to identify all the *bad patterns* that can arise in dependency-graphs like structures. We solved the strong correspondence problem for SI in [16].

References

1 Atul Adya. Weak consistency: A generalized theory and optimistic implementations for distributed transactions. PhD thesis, MIT, 1999.

2 Jade Alglave, Daniel Kroening, Vincent Nimal, and Daniel Poetzl. Don't sit on the fence: A static analysis approach to automatic fence insertion. *ACM Transactions on Programming Languages Systems*, 39(2):6:1–6:38, 2017.

3 Jade Alglave and Luc Maranget. Stability in weak memory models. In *International Confence on Computer Aided Verification (CAV)*, pages 50–66, 2011.

4 Jade Alglave, Luc Maranget, and Michael Tautschnig. Herding cats: Modelling, simulation, testing, and data mining for weak memory. *ACM Transactions on Programming Languages Systems*, 36(2):7:1–7:74, 2014.

5 Peter Bailis, Alan Fekete, Ali Ghodsi, Joseph M. Hellerstein, and Ion Stoica. Scalable atomic visibility with RAMP transactions. In *2014 ACM SIGMOD International Conference on Management of Data (SIGMOD)*, pages 27–38, 2014.

6 Hal Berenson, Phil Bernstein, Jim Gray, Jim Melton, Elizabeth O'Neil, and Patrick O'Neil. A critique of ANSI SQL isolation levels. In *1995 ACM SIGMOD international conference on Management of data (SIGMOD)*, pages 1–10, 1995.

7 Giovanni Bernardi and Alexey Gotsman. Robustness against consistency models with atomic visibility. In *27th International Conference on Concurrency Theory (CONCUR)*, pages 7:1–7:15, 2016. doi:10.4230/LIPIcs.CONCUR.2016.7.

8 Ahmed Bouajjani, Egor Derevenetc, and Roland Meyer. Checking and enforcing robustness against TSO. In *23rd European Symposium on Programming (ESOP)*, pages 533–553, 2013.

9 Ahmed Bouajjani, Constantin Enea, Rachid Guerraoui, and Jad Hamza. On verifying causal consistency. In *44th ACM SIGPLAN Symposium on Principles of Programming Languages (POPL)*, pages 626–638, 2017.

10 Lucas Brutschy, Dimitar Dimitrov, Peter Müller, and Martin Vechev. Serializability for eventual consistency: Criterion, analysis and applications. In *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages (POPL)*. ACM, January 2017.

11 Sebastian Burckhardt. Principles of eventual consistency. *Foundations and Trends in Programming Languages*, 1(1-2):1–150, 2014. doi:10.1561/2500000011.

12 Sebastian Burckhardt, Manuel Fahndrich, Daan Leijen, and Mooly Sagiv. Eventually consistent transactions. In *22nd European Symposium on Programming (ESOP)*, page 67–86, 2012. URL: `https://www.microsoft.com/en-us/research/publication/eventually-consistent-transactions/`.

13 Sebastian Burckhardt, Alexey Gotsman, Hongseok Yang, and Marek Zawirski. Replicated data types: specification, verification, optimality. In *41st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL)*, pages 271–284, 2014.

14 Simon Castellan. Weak memory models using event structures. In *Vingt-septièmes Journées Francophones des Langages Applicatifs (JFLA 2016)*, 2016.

15 Andrea Cerone, Giovanni Bernardi, and Alexey Gotsman. A framework for transactional consistency models with atomic visibility. In *26th International Conference on Concurrency Theory (CONCUR)*, pages 58–71. Dagstuhl, 2015.

16 Andrea Cerone and Alexey Gotsman. Analysing snapshot isolation. In *2016 ACM Symposium on Principles of Distributed Computing (PODC)*, pages 55–64, 2016.

17 Andrea Cerone, Alexey Gotsman, and Hongseok Yang. Algebraic laws for weak consistency (extended version). URL: `https://arxiv.org/abs/1702.06028`.

18 Andrea Cerone, Alexey Gotsman, and Hongseok Yang. Transaction chopping for parallel snapshot isolation. In *29th International Symposium on Distributed Computing (DISC)*, pages 388–404, 2015.

19 Alan Fekete, Dimitrios Liarokapis, Elizabeth O'Neil, Patrick O'Neil, and Dennis Shasha. Making snapshot isolation serializable. *ACM Transactions on Database Systems*, 30(2):492–528, 2005.

20 Alexey Gotsman and Hongseok Yang. Composite replicated data types. In Jan Vitek, editor, *24th European Symposium on Programming (ESOP)*, pages 585–609, 2015.

21 Alan Jeffrey and James Riely. On thin air reads towards an event structures model of relaxed memory. In *31st ACM/IEEE Symposium on Logic in Computer Science (LICS)*, pages 759–767, 2016.

22 Dexter Kozen and Frederick Smith. Kleene algebra with tests: Completeness and decidability. In *10th International Workshop on Computer Science Logic (CSL)*, pages 244–259. Springer-Verlag, 1996.

23 Cheng Li, Daniel Porto, Allen Clement, Johannes Gehrke, Nuno Preguiça, and Rodrigo Rodrigues. Making geo-replicated systems fast as possible, consistent when necessary. In *10th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, pages 265–278, 2012.

24 Wyatt Lloyd, Michael J. Freedman, Michael Kaminsky, and David G. Andersen. Don't settle for eventual: scalable causal consistency for wide-area storage with COPS. In *23rd ACM Symposium on Operating Systems Principles (SOSP)*, pages 401–416, 2011.

25 Microsoft SQL server documentation, Set Transaction Isolation Level. URL: `https://docs.microsoft.com/en-us/sql/t-sql/statements/set-transaction-isolation-level-transact-sql`.

26 M. Saeida Ardekani, P. Sutra, and M. Shapiro. Non-monotonic snapshot isolation: Scalable and strong consistency for geo-replicated transactional systems. In *32nd International Symposium on Reliable Distributed Systems (SRDS)*, pages 163–172, 2013.

27 D. Shasha, F. Llirbat, E. Simon, and P. Valduriez. Transaction chopping: Algorithms and performance studies. *ACM Trans. Database Syst.*, 20(3):325–363, 1995.

28 Y. Sovran, R. Power, M. K. Aguilera, and J. Li. Transactional storage for geo-replicated systems. In *23rd ACM Symposium on Operating Systems Principles (SOSP)*, pages 385–400, 2011.

29 Douglas B Terry, Alan J Demers, Karin Petersen, Mike J Spreitzer, Marvin M Theimer, and Brent B Welch. Session guarantees for weakly consistent replicated data. In *3rd*

International Conference on Parallel and Distributed Information Systems (PDIS), pages 140–149. IEEE, 1994.

30 Paolo Viotti and Marko Vukolić. Consistency in non-transactional distributed storage systems. *ACM Computing Surveys*, 49(1):19:1–19:34, 2016. doi:10.1145/2926965.

31 Kamal Zellag and Bettina Kemme. Consistency anomalies in multi-tier architectures: Automatic detection and prevention. *The VLDB Journal*, 23(1):147–172, 2014.