Robustness against Read Committed for Transaction Templates with Functional Constraints

Brecht Vandevoort ✉
Hasselt University and Transnational University of Limburg, Belgium

Bas Ketsman ✉
Vrije Universiteit Brussel, Belgium

Christoph Koch ✉
École Polytechnique Fédérale de Lausanne, Switzerland

Frank Neven ✉
Hasselt University and Transnational University of Limburg, Belgium

Abstract

The popular isolation level Multiversion Read Committed (RC) trades some of the strong guarantees of serializability for increased transaction throughput. Sometimes, transaction workloads can be safely executed under RC obtaining serializability at the lower cost of RC. Such workloads are said to be robust against RC. Previous work has yielded a tractable procedure for deciding robustness against RC for workloads generated by transaction programs modeled as transaction templates. An important insight of that work is that, by more accurately modeling transaction programs, we are able to recognize larger sets of workloads as robust. In this work, we increase the modeling power of transaction templates by extending them with functional constraints, which are useful for capturing data dependencies like foreign keys. We show that the incorporation of functional constraints can identify more workloads as robust that otherwise would not be. Even though we establish that the robustness problem becomes undecidable in its most general form, we show that various restrictions on functional constraints lead to decidable and even tractable fragments that can be used to model and test for robustness against RC for realistic scenarios.

2012 ACM Subject Classification Information systems → Database transaction processing

Keywords and phrases concurrency control, robustness, complexity

Digital Object Identifier 10.4230/LIPIcs.CVIT.2016.23

1 Introduction

Many database systems implement several isolation levels, allowing users to trade isolation guarantees for improved performance. The highest, serializability, projects the appearance of a complete absence of concurrency, and thus perfect isolation. Executing transactions concurrently under weaker isolation levels can introduce certain anomalies. Sometimes, a transactional workload can be executed at an isolation level lower than serializability without introducing any anomalies. This is a desirable scenario: a lower isolation level, usually implementable with a cheaper concurrency control algorithm, yields the stronger isolation guarantees of serializability for free. This formal property is called robustness [12, 7]: a set of transactions $\mathcal{T}$ is called *robust against a given isolation level* if every possible interleaving of the transactions in $\mathcal{T}$ that is allowed under the specified isolation level is serializable.

Robustness received quite a bit of attention in the literature. Most existing work focuses on Snapshot Isolation (SI) [2, 4, 12, 13] or higher isolation levels [5, 7, 8, 10]. It is particularly interesting to consider robustness against lower level isolation levels like multi-version Read Committed (referred to as RC from now on). Indeed, RC is widely available, often the default in database systems (see, e.g., [4]), and is generally expected to have better throughput than stronger isolation levels.

In previous work [18], we provided a tractable decision procedure for robustness against RC for workloads generated by transaction programs modeled as transaction templates. The approach is centered on a novel characterization of robustness against RC in the spirit of [12, 14] that improves over the sufficient condition presented in [3], and on a formalization of transaction programs, called *transaction templates*, facilitating fine-grained reasoning for robustness against RC. Conceptually, transaction templates as introduced in [18] are functions with parameters, and can, for instance, be derived from stored procedures inside a database system (c.f. Figure 1 for an example). The abstraction generalizes transactions as usually studied in concurrency control research – sequences of read and write operations – by making the objects worked on variable, determined by input parameters. Such parameters are *typed* to add additional power to the analysis. They support *atomic updates* (that is, a read followed by a write of the same database object, to make a relative change to its value). Furthermore, database objects read and written are considered at the granularity of fields, rather than just entire tuples, decoupling conflicts further and allowing to recognize additional cases that would not be recognizable as robust on the tuple level.

An important insight obtained from [18] is that more accurate modeling of the workload allows to recognize larger sets of transaction programs as robust. Processing workloads under RC increases the throughput of the transactional database system compared to when executing the workload under SI or serializable SI, so larger robust sets mean better performance of the database system. In this work, we increase the modeling power of transaction templates by extending them with *functional constraints*, which are useful for capturing data dependencies like foreign keys (inclusion dependencies). This appears to be a sweet spot for strengthening modelling power – as we show in this paper, it allows us to remain with abstractions that have been well established within database theory, without having to move to general program analysis, and it pushes the robustness frontier on popular transaction processing benchmarks. Generally speaking, workloads can profit more from richer modelling the larger and more complex they get, so the fact that adding functional constraints yields larger robust sets already on these simple benchmarks suggests that these techniques are practically useful. Our contributions can be summarized as follows:

- We argue in Section 2 through the SmallBank and TPC-C benchmarks that the incorporation of functional constraints can identify more workloads as robust that otherwise would not be, and that they reduce the extent to which changes need to be made to workloads to make them robust against RC.
- In Section 4, we establish that robustness in its most general form becomes undecidable. The proof is a reduction from PCP and relies on cyclic dependencies between functions allowing to connect data values through an unbounded application of functions.
- We consider a fragment in Section 5 that only allows a very limited form of cyclic dependencies between functions and assumes additional constraints on templates that, together, imply that functions behave as bijections. Robustness against RC can be decided in NLOGSPACE and this fragment is general enough to model the SmallBank benchmark.
- In Section 6, we obtain an EXPSPACE decision procedure when the schema graph is acyclic (so, no cyclic dependencies between functions). Even for small input sizes, such a result is not practical. We provide various restrictions that lower the complexity to PSPACE and EXPTIME, and which allow to model the TPC-C benchmark as discussed. Notice that, for robustness testing, an exponential time decision procedure is considered to be practical as the size of the input is small and robustness is a static property that can be tested offline.

GoPremium:

$\texttt{U}[\texttt{X} : \text{Account}\{\text{N, C}\}\{\text{I}\}]$
$\texttt{R}[\texttt{Y} : \text{Savings}\{\text{C, I}\}]$
$\texttt{U}[\texttt{Y} : \text{Savings}\{\text{C}\}\{\text{I}\}]$
$\texttt{Y} = f_{A\to S}(\texttt{X}),\ \texttt{X} = f_{S\to A}(\texttt{Y})$

Figure 1 Transaction template.

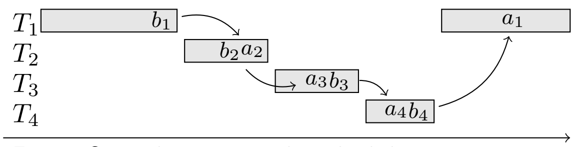

Figure 2 Multiversion split schedule.

Due to space constraints, proofs as well as a more complete description of the SmallBank and TPC-C benchmarks are moved to the appendix.

2 Application

We present a small extension of the SmallBank benchmark [2] to exemplify the modeling power of transaction templates and discuss how the addition of functional constraints can detect larger sets of transaction templates to be robust. Finally, we discuss in the context of the TPC-C benchmark how the incorporation of functional constraints requires less changes to templates in making them robust. A full description of these benchmarks is in Appendix D.

The SmallBank schema consists of three tables: Account(Name, CustomerID, IsPremium), Savings(CustomerID, Balance, InterestRate), and Checking(CustomerID, Balance). Underlined attributes are primary keys. The Account table associates customer names with IDs and keeps track of the premium status (Boolean); CustomerID is a `UNIQUE` attribute. The other tables contain the balance (numeric value) of the savings and checking accounts of customers identified by their ID. Account (CustomerID) is a foreign key referencing both the columns Savings (CustomerID) and Checking (CustomerID). The interest rate on a savings account is based on a number of parameters, including the account status (premium or not). The application code can interact with the database through a fixed number of transaction programs presented in Appendix D.1: Balance, TransactSavings, Amalgamate, WriteCheck, DepositChecking, and GoPremium. We only discuss GoPremium(N), given in Figure 1, which converts the account of the customer with name N to a premium account and updates the interest rate of the corresponding savings account.

In short, a transaction template is a sequence of read (`R`), write (`W`) and update (`U`) statements over typed variables (`X`, `Y`, ...) with additional equality and disequality constraints. For instance, $\texttt{R}[\texttt{Y} : \text{Savings}\{\texttt{C}, \texttt{I}\}]$ indicates that a read operation is performed to a tuple in relation Savings on the attributes CustomerID and InterestRate. We abbreviate the names of attributes by their first letter to save space. The set $\{C, I\}$ is the read set. Write operations have an associated write set while update operations contain a read set followed by a write set: e.g., $\texttt{U}[\texttt{Z} : \text{Account}\{\texttt{N}, \texttt{C}\}\{\texttt{I}\}]$ first reads the Name and CustomerID of tuple `X` and then writes to the attribute InterestRate. To capture the dependencies between tuples induced by the foreign keys, we use two unary functions: $f_{A\to S}$ maps a tuple of type Account to a tuple of type Savings, while $f_{A\to C}$ maps a tuple of type Account to a tuple of type Checking. As Account(CustomerID) is `UNIQUE`, every savings and checking accounts is associated to a unique Account tuple. This is modelled through the functions $f_{C\to A}$ and $f_{S\to A}$ with an analogous interpretation. Notice that the equality constraints for GoPremium imply that these functions are bijections and each others inverses.

A transaction T over a database $\mathbf{D}$ is an *instantiation* of a transaction template τ if there is a variable mapping μ from the variables in τ to tuples in $\mathbf{D}$ that satisfies all the constraints in τ such that with $\mu(\tau) = T$. For instance, consider a database $\mathbf{D}$ with tuples $\texttt{a}_1, \texttt{a}_2, \ldots$ of type Account, $\texttt{s}_1, \texttt{s}_2, \ldots$ of type Savings, and $\texttt{c}_1, \texttt{c}_2, \ldots$ of type Checking with

Delivery:

$\mathtt{U}[\mathtt{S} : \text{Order}\{\text{W, D, O}\}\{\text{Sta}\}]$
$\mathtt{U}[\mathtt{V}_1 : \text{OrderLine}\{\text{W, D, O, OL, Del}\}\{\text{Del}\}]$
$\mathtt{U}[\mathtt{V}_2 : \text{OrderLine}\{\text{W, D, O, OL, Del}\}\{\text{Del}\}]$
$\mathtt{U}[\mathtt{Z} : \text{Customer}\{\text{W, D, C, Bal}\}\{\text{Bal}\}]$
$\mathtt{Z} = f_{O\to C}(S),\ S = f_{L\to O}(\mathtt{V}_1),\ S = f_{L\to O}(\mathtt{V}_2)$

OrderStatus:

$\mathtt{R}[\mathtt{Z} : \text{Customer}\{\text{W, D, C, Inf, Bal}\}]$
$\mathtt{R}[\mathtt{S} : \text{Order}\{\text{W, D, O, C, Sta}\}]$
$\mathtt{R}[\mathtt{V}_1 : \text{OrderLine}\{\text{W, D, O, OL, I, Del, Qua}\}]$
$\mathtt{R}[\mathtt{V}_2 : \text{OrderLine}\{\text{W, D, O, OL, I, Del, Qua}\}]$
$\mathtt{Z} = f_{O\to C}(S),\ S = f_{L\to O}(\mathtt{V}_1),\ S = f_{L\to O}(\mathtt{V}_2)$

Figure 3 Transaction templates Delivery and OrderStatus of the TPC-C benchmark.

$f^{\mathbf{D}}_{A\to S}(\mathtt{a}_i) = \mathtt{s}_i$, $f^{\mathbf{D}}_{A\to C}(\mathtt{a}_i) = \mathtt{c}_i$, $f^{\mathbf{D}}_{S\to A}(\mathtt{s}_i) = \mathtt{a}_i$, $f^{\mathbf{D}}_{C\to A}(\mathtt{c}_i) = \mathtt{a}_i$ for each i. Then, for $\mu_1 = \{\mathtt{X} \to \mathtt{a}_1, \mathtt{Y} \to \mathtt{s}_1\}$, $\mu_1(\text{GoPremium}) = \mathtt{U}[\mathtt{a}_1]\mathtt{R}[\mathtt{s}_1]\mathtt{U}[\mathtt{c}_1]$ is an instantiation of GoPremium whereas $\mu_2(\text{GoPremium})$ with $\mu_2 = \{\mathtt{X} \to \mathtt{a}_1, \mathtt{Y} \to \mathtt{s}_2\}$ is not as the functional constraint $\mathtt{Y} = f_{A\to S}(\mathtt{X})$ is not satisfied. Indeed, $\mu_2(\mathtt{Y}) = \mathtt{s}_2 \neq \mathtt{s}_1 = f^{\mathbf{D}}_{A\to S}(\mathtt{a}_1) = f^{\mathbf{D}}_{A\to S}(\mu_2(\mathtt{X}))$. We then say that a set of transactions is *consistent* with a set of templates if every transaction is an instantiation of a transaction template.

Our previous work [18], which did not consider functional constraints, has shown that {Am,DC,TS}, {Bal,DC}, and {Bal,TS} are maximal robust sets of transaction templates. This means that for any database, for any set of transactions $\mathcal{T}$ that is consistent with one of the three mentioned sets, any possible interleaving of the transactions in $\mathcal{T}$ that is allowed under RC is *always* serializable! Using the results from Section 5, it follows that when functional constraints are taken into account GoPremium can be added to each of these sets as well: {Am,DC,GP,TS}, {Bal,DC,GP}, {Bal,TS,GP} are maximal robust sets.

We argue that incorporating functional constraints is crucial. Indeed, without functional constraints its easy to show that even the set {GoPremium} is not robust. Consider the schedule over two instantiations T_1 and T_2 of GoPremium, where we use the mappings μ_1 and μ_2 as defined above for respectively T_1 and T_2 (we show the read and write sets to facilitate the discussion):

$$\begin{array}{lll} T_1: \mathtt{U}_1[\mathtt{a}_1\{\text{N,C}\}\{\text{I}\}]\,\mathtt{R}_1[\mathtt{s}_1\{\text{C,I}\}] & & \mathtt{U}_1[\mathtt{s}_1\{\text{C}\}\{\text{I}\}]\,\mathtt{C}_1 \\ T_2: & \mathtt{U}_2[\mathtt{a}_2\{\text{N,C}\}\{\text{I}\}]\,\mathtt{R}_2[\mathtt{s}_1\{\text{C,I}\}]\,\mathtt{U}_2[\mathtt{s}_1\{\text{C}\}\{\text{I}\}]\,\mathtt{C}_2 & \end{array}$$

The above schedule is allowed under RC as there is no dirty write, but it is not conflict serializable. Indeed, there is a rw-conflict between $\mathtt{R}_1[\mathtt{s}_1\{\text{C,I}\}]$ and $\mathtt{U}_2[\mathtt{s}_1\{\text{C}\}\{\text{I}\}]$ as the former reads the attribute I that is written to by the latter, which implies that T_1 should occur before T_2 in an equivalent serial schedule. But, there is a ww-conflict between $\mathtt{U}_2[\mathtt{s}_1\{\text{C}\}\{\text{I}\}]$ and $\mathtt{U}_1[\mathtt{s}_1\{\text{C}\}\{\text{I}\}]$ as both write to the common attribute I implying that T_2 should occur before T_1 in an equivalent serial schedule. Consequently, the schedule is not serializable. However, taking functional constraints into account, $\{T_1, T_2\}$ is not consistent with {GoPremium} as $\mu_2(\mathtt{Y}) = \mathtt{s}_1 \neq \mathtt{s}_2 = f_{A\to S}(a_2) = f_{A\to S}(\mu_2(\mathtt{X}))$ implying that the above schedule is *not* a counter example for robustness.

Incorporating functional constraints for TPC-C can not identify larger sets of templates to be robust. However, when a set of transaction templates $\mathcal{P}$ is not robust against RC, an equivalent set of templates $\mathcal{P}'$ can be constructed from $\mathcal{P}$ by *promoting* certain R-operations to U-operations [18]. By incorporating functional constraints it can be shown that fewer R-operations need to be promoted leading to an increase in throughput as R-operations do not take locks whereas U-operations do. Consider for example the subset $\mathcal{P} = \{\text{Delivery}, \text{OrderStatus}\}$ of the TPC-C benchmark, given in Figure 3, where functional constraints are added to express the fact that a tuple of type OrderLine implies the tuple of type Order, which in turn implies the tuple of type Customer. This set $\mathcal{P}$ is not robust against RC, but robustness can be achieved by promoting the R-operation over Customer in OrderStatus to a U-operation. However, without functional constraints, this single promoted operation no longer guarantees robustness, as witnessed by the following schedule:

$$\begin{array}{llll} T_1(\text{Orderstatus}): & \mathtt{U}_1[\mathtt{c}]\,\mathtt{R}_1[\mathtt{a}] & & \mathtt{R}_1[\mathtt{b}_1]\,\mathtt{R}_1[\mathtt{b}_2]\,\mathtt{C}_1 \\ T_2(\text{Delivery}): & & \mathtt{U}_2[\mathtt{a}]\,\mathtt{U}_2[\mathtt{b}_1]\,\mathtt{U}_2[\mathtt{b}_2]\,\mathtt{U}_2[\mathtt{c}']\,\mathtt{C}_2 & \end{array}$$

Notice in particular how this schedule implicitly assumes in T_2 that Order $\mathtt{a}$ belongs to Customer $\mathtt{c}'$ instead of Customer $\mathtt{c}$ to avoid a dirty write on $\mathtt{c}$. Without functional constraints, $\mathcal{P}$ is only robust against RC if *all* R-operations in OrderStatus are promoted to U-operations.

3 Definitions

We recall the necessary definitions from [18] and extend them with functional constraints.

3.1 Databases

A *relational schema* is a pair $(\mathsf{Rels}, \mathsf{Funcs})$ where Rels is a set of relation names and Funcs is a set of function names. A finite set of attribute names $\mathrm{Attr}(R)$ is associated to every relation $R \in \mathsf{Rels}$. Relations will be instantiated by abstract objects that serve as an abstraction of relational tuples. To this end, for every relation $R \in \mathsf{Rels}$, we fix an infinite set of tuples $\mathbf{Tuples}_R$. Furthermore, we assume that $\mathbf{Tuples}_R \cap \mathbf{Tuples}_S = \emptyset$ for all $R, S \in \mathsf{Rels}$ with $R \neq S$. We then denote by $\mathbf{Tuples}$ the set $\bigcup_{R \in \mathsf{Rels}} \mathbf{Tuples}_R$ of all possible tuples. Notice that, by definition, for every $\mathtt{t} \in \mathbf{Tuples}$ there is a unique relation $R \in \mathsf{Rels}$ such that $\mathtt{t} \in \mathbf{Tuples}_R$. In that case, we say that $\mathtt{t}$ is of *type* R and denote the latter by $\mathrm{type}(\mathtt{t}) = R$. Each function name $f \in \mathsf{Funcs}$ has a domain $\mathit{dom}(f) \in \mathsf{Rels}$ and a range $\mathit{range}(f) \in \mathsf{Rels}$. Functions are used to encode relationships between tuples like for instance those implied by foreign-keys constraints. For instance, in the SmallBank example $\mathsf{Funcs} = \{f_{A\to S}, f_{A\to C}\}$, $\mathit{dom}(f_{A\to S}) = \mathit{dom}(f_{A\to C}) = A$, $\mathit{range}(f_{A\to S}) = S$, and $\mathit{range}(f_{A\to C}) = C$. A *database* $\mathbf{D}$ over schema $(\mathsf{Rels}, \mathsf{Funcs})$ assigns to every relation name $R \in \mathsf{Rels}$ a finite set $R^{\mathbf{D}} \subset \mathbf{Tuples}_R$ and to every function name $f \in \mathsf{Funcs}$ a function $f^{\mathbf{D}}$ from $\mathit{dom}(f)^{\mathbf{D}}$ to $\mathit{range}(f)^{\mathbf{D}}$.

3.2 Transactions and Schedules

For a tuple $\mathtt{t} \in \mathbf{Tuples}$, we distinguish three operations $\mathtt{R}[\mathtt{t}]$, $\mathtt{W}[\mathtt{t}]$, and $\mathtt{U}[\mathtt{t}]$ on $\mathtt{t}$, denoting that tuple $\mathtt{t}$ is read, written, or updated, respectively. We say that the operation is on the tuple $\mathtt{t}$. The operation $\mathtt{U}[\mathtt{t}]$ is an atomic update and should be viewed as an atomic sequence of a read of $\mathtt{t}$ followed by a write to $\mathtt{t}$. We will use the following terminology: a *read operation* is an $\mathtt{R}[\mathtt{t}]$ or a $\mathtt{U}[\mathtt{t}]$, and a *write operation* is a $\mathtt{W}[\mathtt{t}]$ or a $\mathtt{U}[\mathtt{t}]$. Furthermore, an R-operation is an $\mathtt{R}[\mathtt{t}]$, a W-operation is a $\mathtt{W}[\mathtt{t}]$, and a U-operation is a $\mathtt{U}[\mathtt{t}]$. We also assume a special *commit* operation denoted $\mathtt{C}$. To every operation o on a tuple of type R, we associate the set of attributes $\mathrm{ReadSet}(o) \subseteq \mathrm{Attr}(R)$ and $\mathrm{WriteSet}(o) \subseteq \mathrm{Attr}(R)$ containing, respectively, the set of attributes that o reads from and writes to. When o is a R-operation then $\mathrm{WriteSet}(o) = \emptyset$. Similarly, when o is a W-operation then $\mathrm{ReadSet}(o) = \emptyset$.

A *transaction* T is a sequence of read and write operations followed by a commit. We assume that a transactions starts when its first operation is executed, but no earlier. Formally, we model a transaction as a linear order $(T, \leq_T)$, where T is the set of (read, write and commit) operations occurring in the transaction and $\leq_T$ encodes the ordering of the operations. As usual, we use $<_T$ to denote the strict ordering.

When considering a set $\mathcal{T}$ of transactions, we assume that every transaction in the set has a unique id i and write T_i to make this id explicit. Similarly, to distinguish the operations of different transactions, we add this id as a subscript to the operation. That is, we write

$\texttt{W}_i[\texttt{t}]$, $\texttt{R}_i[\texttt{t}]$, and $\texttt{U}_i[\texttt{t}]$ to denote a $\texttt{W}[\texttt{t}]$, $\texttt{R}[\texttt{t}]$, and $\texttt{U}[\texttt{t}]$ occurring in transaction T_i; similarly $\texttt{C}_i$ denotes the commit operation in transaction T_i. This convention is consistent with the literature (see, *e.g.* [6, 12]). To avoid ambiguity of notation, we assume that a transaction performs at most one write, one read, and one update per tuple. The latter is a common assumption (see, *e.g.* [12]). All our results carry over to the more general setting in which multiple writes and reads per tuple are allowed.

A *(multiversion) schedule* s over a set $\mathcal{T}$ of transactions is a tuple $(O_s, \leq_s, \ll_s, v_s)$ where O_s is the set containing all operations of transactions in $\mathcal{T}$ as well as a special operation op_0 conceptually writing the initial versions of all existing tuples, $\leq_s$ encodes the ordering of these operations, $\ll_s$ is a *version order* providing for each tuple $\texttt{t}$ a total order over all write operations on $\texttt{t}$ occurring in s, and v_s is a *version function* mapping each read operation a in s to either op_0 or to a write[1] operation different from a in s. We require that $op_0 \leq_s a$ for every operation $a \in O_s$, $op_0 \ll_s a$ for every write operation $a \in O_s$, and that $a <_T b$ implies $a <_s b$ for every $T \in \mathcal{T}$ and every $a, b \in T$. We furthermore require that for every read operation a, $v_s(a) <_s a$ and, if $v_s(a) \neq op_0$, then the operation $v_s(a)$ is on the same tuple as a. Intuitively, op_0 indicates the start of the schedule, the order of operations in s is consistent with the order of operations in every transaction $T \in \mathcal{T}$, and the version function maps each read operation a to the operation that wrote the version observed by a. If $v_s(a)$ is op_0, then a observes the initial version of this tuple. The version order $\ll_s$ represents the order in which different versions of a tuple are installed in the database. For a pair of write operations on the same tuple, this version order does not necessarily coincide with $\leq_s$. For example, under RC the version order is based on the commit order instead.

We say that a schedule s is a *single version schedule* if $\ll_s$ coincides with $\leq_s$ and every read operation always reads the last written version of the tuple. Formally, for each pair of write operations a and b on the same tuple, $a \ll_s b$ iff $a <_s b$, and for every read operation a there is no write operation c on the same tuple as a with $v_s(a) <_s c <_s a$. A single version schedule over a set of transactions $\mathcal{T}$ is *single version serial* if its transactions are not interleaved with operations from other transactions. That is, for every $a, b, c \in O_s$ with $a <_s b <_s c$ and $a, c \in T$ implies $b \in T$ for every $T \in \mathcal{T}$.

The absence of aborts in our definition of schedule is consistent with the common assumption [12, 7] that an underlying recovery mechanism will rollback aborted transactions. We only consider isolation levels that only read committed versions. Therefore there will never be cascading aborts.

3.3 Conflict Serializability

Let a_j and b_i be two operations on the same tuple from different transactions T_j and T_i in a set of transactions $\mathcal{T}$. We then say that a_j is *conflicting* with b_i if:

- *(ww-conflict)* $\text{WriteSet}(a_j) \cap \text{WriteSet}(b_i) \neq \emptyset$; or,
- *(wr-conflict)* $\text{WriteSet}(a_j) \cap \text{ReadSet}(b_i) \neq \emptyset$; or,
- *(rw-conflict)* $\text{ReadSet}(a_j) \cap \text{WriteSet}(b_i) \neq \emptyset$.

In this case, we also say that a_j and b_i are conflicting operations. Furthermore, commit operations and the special operation op_0 never conflict with any other operation. When a_j and b_i are conflicting operations in $\mathcal{T}$, we say that a_j *depends on* b_i in a schedule s over $\mathcal{T}$, denoted $b_i \rightarrow_s a_j$ if:[2]

---

[1] Recall that a write operation is either a $\texttt{W}[\texttt{x}]$ or a $\texttt{U}[\texttt{x}]$.

[2] Throughout the paper, we adopt the following convention: a b operation can be understood as a 'before'

- *(ww-dependency)* b_i is ww-conflicting with a_j and $b_i \ll_s a_j$; or,
- *(wr-dependency)* b_i is wr-conflicting with a_j and $b_i = v_s(a_j)$ or $b_i \ll_s v_s(a_j)$; or,
- *(rw-antidependency)* b_i is rw-conflicting with a_j and $v_s(b_i) \ll_s a_j$.

Intuitively, a ww-dependency from b_i to a_j implies that a_j writes a version of a tuple that is installed after the version written by b_i. A wr-dependency from b_i to a_j implies that b_i either writes the version observed by a_j, or it writes a version that is installed before the version observed by a_j. A rw-antidependency from b_i to a_j implies that b_i observes a version installed before the version written by a_j.

Two schedules s and s' are *conflict equivalent* if they are over the same set $\mathcal{T}$ of transactions and for every pair of conflicting operations a_j and b_i, $b_i \rightarrow_s a_j$ iff $b_i \rightarrow_{s'} a_j$.

▶ **Definition 1.** *A schedule s is* conflict serializable *if it is conflict equivalent to a single version serial schedule.*

A *conflict graph* $CG(s)$ for schedule s over a set of transactions $\mathcal{T}$ is the graph whose nodes are the transactions in $\mathcal{T}$ and where there is an edge from T_i to T_j if T_i has an operation b_i that conflicts with an operation a_j in T_j and $b_i \rightarrow_s a_j$.

▶ **Theorem 2** ([15])**.** *A schedule s is conflict serializable iff the conflict graph for s is acyclic.*

3.4 Multiversion Read Committed

Let s be a schedule for a set $\mathcal{T}$ of transactions. Then, *s exhibits a dirty write* iff there are two ww-conflicting operations a_j and b_i in s on the same tuple $\mathtt{t}$ with $a_j \in T_j$, $b_i \in T_i$ and $T_j \neq T_i$ such that $b_i <_s a_j <_s \mathtt{C}_i$. That is, transaction T_j writes to an attribute of a tuple that has been modified earlier by T_i, but T_i has not yet issued a commit.

For a schedule s, the version order $\ll_s$ corresponds to the commit order in s if for every pair of write operations $a_j \in T_j$ and $b_i \in T_i$, $b_i \ll_s a_j$ iff $\mathtt{C}_i <_s a_j$. We say that a schedule s is *read-last-committed (RLC)* if $\ll_s$ corresponds to the commit order and for every read operation a_j in s on some tuple $\mathtt{t}$ the following holds:

- $v_s(a_j) = op_0$ or $\mathtt{C}_i <_s a_j$ with $v_s(a_j) \in T_i$; and
- there is no write[3] operation $c_k \in T_k$ on $\mathtt{t}$ with $\mathtt{C}_k <_s a_j$ and $v_s(a_j) \ll_s c_k$.

So, a_j observes the most recent version of $\mathtt{t}$ (according to the order of commits) that is committed before a_j. Note in particular that a schedule cannot exhibit dirty reads, defined in the traditional way [6], if it is read-last-committed.

▶ **Definition 3.** *A schedule is* allowed under isolation level *read committed (RC) if it is read-last-committed and does not exhibit dirty writes.*

3.5 Transaction Templates

Transaction templates are transactions where operations are defined over typed variables together with functional constraints on these variables. Types of variables are relation names in Rels and indicate that variables can only be instantiated by tuples from the respective type. We fix an infinite set of variables **Var** that is disjoint from **Tuples**. Every variable $\mathtt{X} \in \mathbf{Var}$ has an associated relation name in Rels as type that we denote by type($\mathtt{X}$). For an operation o_i in a template, $var(o_i)$ denotes the variable in o_i. An *equality constraint* is an

while an a can be interpreted as an 'after'.

[3] Recall that a write operation is either a $\mathtt{W}$ or a $\mathtt{U}$-operation.

expression of the form $\mathtt{X} = f(\mathtt{Y})$ where $\mathtt{X}, \mathtt{Y} \in \mathbf{Var}$, $dom(f) = \text{type}(\mathtt{Y})$ and $range(f) = \text{type}(\mathtt{X})$. A *disequality constraint* is an expression of the form $\mathtt{X} \neq \mathtt{Y}$ where $\text{type}(\mathtt{X}) = \text{type}(\mathtt{Y})$.

▶ **Definition 4.** *A* transaction template *is a transaction τ over* **Var** *together with a set $\Gamma(\tau)$ of equality and disequality constraints. In addition, for every operation o in τ over a variable* $\mathtt{X}$*, $ReadSet(o) \subseteq Attr(type(\mathtt{X}))$ and $WriteSet(o) \subseteq Attr(type(\mathtt{X}))$.*

Recall that we denote variables by capital letters $\mathtt{X}, \mathtt{Y}, \mathtt{Z}$ and tuples by small letters $\mathtt{t}, \mathtt{v}$. A variable assignment μ is a mapping from **Var** to **Tuples** such that $\mu(\mathtt{X}) \in \mathbf{Tuples}_{\text{type}(\mathtt{X})}$. Furthermore, μ *satisfies* a constraint $\mathtt{X} = f(\mathtt{Y})$ (resp., $\mathtt{X} \neq \mathtt{Y}$) over a database $\mathbf{D}$ when $\mu(\mathtt{X}) = f^{\mathbf{D}}(\mu(\mathtt{Y}))$ (resp., $\mu(\mathtt{X}) \neq \mu(\mathtt{Y})$). A variable assignment μ for a transaction template τ is *admissible* for $\mathbf{D}$ if it satisfies all constraints in $\Gamma(\tau)$ over $\mathbf{D}$. By $\mu(\tau)$, we denote the transaction obtained by replacing each variable $\mathtt{X}$ in τ with $\mu(\mathtt{X})$.

A set of transactions $\mathcal{T}$ is *consistent* with a set of transaction templates $\mathcal{P}$ and database $\mathbf{D}$, if for every transaction T in $\mathcal{T}$ there is a transaction template $\tau \in \mathcal{P}$ and a variable mapping μ_T that is admissible for $\mathbf{D}$ such that $\mu_T(\tau) = T$.

3.6 Robustness

We define the robustness property [7] (also called *acceptability* in [12, 13]), which guarantees serializability for all schedules of a given set of transactions for a given isolation level.

▶ **Definition 5** (Transaction Robustness). *A set $\mathcal{T}$ of transactions is* robust *against RC if every schedule for $\mathcal{T}$ that is allowed under RC is conflict serializable.*

In the next definition, we represent conflicting operations from transactions in a set $\mathcal{T}$ as quadruples (T_i, b_i, a_j, T_j) with b_i and a_j conflicting operations, and T_i and T_j their respective transactions in $\mathcal{T}$. We call these quadruples *conflicting quadruples* for $\mathcal{T}$. Further, for an operation $b \in T$, we denote by $\mathsf{prefix}_b(T)$ the restriction of T to all operations that are before or equal to b according to $\leq_T$. Similarly, we denote by $\mathsf{postfix}_b(T)$ the restriction of T to all operations that are strictly after b according to $\leq_T$. Throughout the paper, we interchangeably consider transactions both as linear orders as well as sequences. Therefore, T is then equal to the sequence $\mathsf{prefix}_b(T)$ followed by $\mathsf{postfix}_b(T)$ which we denote by $\mathsf{prefix}_b(T) \cdot \mathsf{postfix}_b(T)$ for every $b \in T$.

▶ **Definition 6** (Multiversion split schedule). *Let $\mathcal{T}$ be a set of transactions and $C = (T_1, b_1, a_2, T_2), (T_2, b_2, a_3, T_3), \ldots, (T_m, b_m, a_1, T_1)$ a sequence of conflicting quadruples for $\mathcal{T}$ such that each transaction in $\mathcal{T}$ occurs in at most two different quadruples. A* multiversion split schedule *for $\mathcal{T}$ based on C is a multiversion schedule that has the following form:*

$$\mathsf{prefix}_{b_1}(T_1) \cdot T_2 \cdot \ldots \cdot T_m \cdot \mathsf{postfix}_{b_1}(T_1) \cdot T_{m+1} \cdot \ldots \cdot T_n,$$

where

1. *there is no write operation in $\mathsf{prefix}_{b_1}(T_1)$ ww-conflicting with a write operation in any of the transactions $T_2, \ldots, T_m$;*
2. *$b_1 <_{T_1} a_1$ or b_m is rw-conflicting with a_1; and,*
3. *b_1 is rw-conflicting with a_2.*

Furthermore, $T_{m+1}, \ldots, T_n$ are the remaining transactions in $\mathcal{T}$ (those not mentioned in C) in an arbitrary order.

Figure 2 depicts a schematic multiversion split schedule. The name stems from the fact that the schedule is obtained by splitting one transaction in two (T_1 at operation b_1 in

Figure 2) and placing all other transactions in C in between. The figure does not display the trailing transactions $T_{m+1}, T_{m+2}, \dots$ and assumes $b_1 <_{T_1} a_1$.

The following theorem characterizes non-robustness in terms of the existence of a multiversion split schedule.

▶ **Theorem 7** ([18]). *For a set of transactions $\mathcal{T}$, the following are equivalent:*
1. *$\mathcal{T}$ is not robust against RC;*
2. *there is a multiversion split schedule s for $\mathcal{T}$ based on some C.*

Let $\mathcal{P}$ be a set of transaction templates and $\mathbf{D}$ be a database. Then, $\mathcal{P}$ is *robust against RC over* $\boldsymbol{D}$ if for every set of transactions $\mathcal{T}$ that is consistent with $\mathcal{P}$ and $\mathbf{D}$, it holds that $\mathcal{T}$ is robust against RC.

▶ **Definition 8** (Template Robustness). *A set of transaction templates $\mathcal{P}$ is* robust *against RC if $\mathcal{P}$ is robust against RC for every database $\boldsymbol{D}$.*

We say that a transaction template (τ, Γ) is a *variable transaction template* when $\Gamma = \emptyset$ and an *equality transaction template* when all constraints in Γ are equalities. We denote these sets by **VarTemp** and **EqTemp**, respectively. For an isolation level $\mathcal{I}$ and a class of transaction templates $\mathcal{C}$, T-ROBUSTNESS($\mathcal{C}$,$\mathcal{I}$) is the problem to decide if a given set of transaction templates $\mathcal{P} \in \mathcal{C}$ is robust against $\mathcal{I}$. When $\mathcal{C}$ is the class of all transaction templates, we simply write T-ROBUSTNESS($\mathcal{I}$).

▶ **Theorem 9** ([18]). T-ROBUSTNESS*(**VarTemp**,RC) is decidable in* PTIME.

4 Robustness for Templates

We start out with a negative result and show that the robustness problem in its most general form is undecidable (even when disequalities are not allowed). The proof is a reduction from *Post's Correspondence Problem (PCP)* [16] and relies on cyclic dependencies between functional constraints. The proof can be found in Appendix A and is quite elaborate but the basic intuition is simple: the counterexample split schedule will build up the two strings that need to be generated by the PCP instance by repeated application of functional constraints.

▶ **Theorem 10.** T-ROBUSTNESS*(**EqTemp**,RC) is undecidable.*

It might be tempting to relate the above result to the undecidability of the implication problem for functional and inclusion dependencies [11]. Functional constraints indeed allow to define inclusion dependencies (as in the SmallBank example) but they always relate complete tuples and are not suited to define functional dependencies. Furthermore, the proof of Theorem 10 makes use of only unary relations, for which the implication problem for functional dependencies and inclusion dependencies is known to be decidable.

To obtain decidable fragments, we introduce restrictions on the structure of functional constraints. The *schema graph* $SG(\mathsf{Rels}, \mathsf{Funcs})$ of a schema $(\mathsf{Rels}, \mathsf{Funcs})$ is a directed multigraph having the relations in Rels as nodes, and in which there are as many edges from a node $R \in \mathsf{Rels}$ to node $S \in \mathsf{Rels}$ as there are functions $f \in \mathsf{Funcs}$ with $dom(f) = R$ and $range(f) = S$. We say that a schema $(\mathsf{Rels}, \mathsf{Funcs})$ is *acyclic* if the multigraph $SG(\mathsf{Rels}, \mathsf{Funcs})$ is acyclic and that it is a *multi-tree* if there is at most one directed path between any two nodes in $SG(\mathsf{Rels}, \mathsf{Funcs})$.

▶ **Example 11.** Consider the schema $(\{P, Q, R, S\}, \{f_{P,R}, f_{Q,R}, f_{R,S}\})$ with $dom(f_{i,j}) = i$ and $range(f_{i,j}) = j$ for each function $f_{i,j}$. The corresponding schema graph with solid lines is

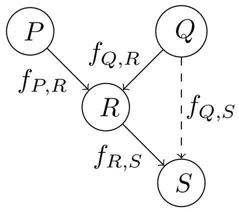

Figure 4 Acyclic schema graph for schema $(\{P, Q, R, S\}, \{f_{P,R}, f_{Q,R}, f_{R,S}, f_{Q,S}\})$. If we remove function name $f_{Q,S}$ (dashed edge), the resulting schema graph is a multi-tree.

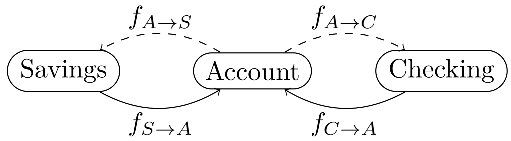

Figure 5 Schema graph for the SmallBank benchmark. The dashed edges correspond to the multi-tree schema graph for the schema restricted to $f_{A\to S}$ and $f_{A\to C}$.

given in Figure 4. This schema is a multi-tree, as there is at most one path between any pair of nodes. Notice that the definition of a multi-tree is more general than a forest, as a node can still have multiple parents (e.g., node R in our example). Adding the function name $f_{Q,S}$ with $dom(f_{Q,S}) = Q$ and $range(f_{Q,S}) = S$ results in the schema graph given in Figure 4 that is still acyclic, but no longer a multi-tree as there are now two paths from Q to S. ◻

The schema graph constructed in the proof of Theorem 10 contains several cycles (cf., Figure 6 in Appendix A). We consider in Section 5 robustness for a fragment where a restricted form of cycles in the schema graph is allowed but where additional constraints on the templates are assumed. We consider robustness for acyclic schema graphs in Section 6.

5 Robustness for Templates admitting Multi-Tree Bijectivity

We say that a set of transaction templates $\mathcal{P}$ over a schema (Rels, Funcs) admits *multi-tree bijectivity* if a disjoint partitioning of Funcs in pairs $(f_1, g_1), (f_2, g_2), \ldots, (f_n, g_n)$ exists such that $dom(f_i) = range(g_i)$ and $dom(g_i) = range(f_i)$ for every pair of function names (f_i, g_i); every schema graph $SG(\mathsf{Rels}, \{h_1, h_2, \ldots, h_n\})$ over the schema restricted to function names $\{h_1, h_2, \ldots, h_n\}$ (with $h_i = f_i$ or $h_i = g_i$) is a multi-tree; and, for every pair of function names (f_i, g_i) and for every pair of variables $\mathtt{X}, \mathtt{Y}$ occurring in a template $\tau_j \in \mathcal{P}$, we have $f_i(\mathtt{X}) = \mathtt{Y} \in \Gamma_j$ iff $g_i(\mathtt{Y}) = \mathtt{X} \in \Gamma_j$. Intuitively, we can think of f_i as a bijective function, with g_i its inverse. We denote the class of all sets of templates admitting multi-tree bijectivity by **MTBTemp**. The SmallBank benchmark given in Figure 8 is in **MTBTemp**, witnessed by the partitioning $\{(f_{A\to C}, f_{C\to A}), (f_{A\to S}, f_{S\to A})\}$. For example, the schema graph restricted to $f_{A\to C}$ and $f_{A\to S}$ is a tree and therefore also a multi-tree, as illustrated in Figure 5.

The next theorem allows disequalities whereas Theorem 10 does not require them.

▶ **Theorem 12.** t-robustness*(**MTBTemp**,RC) is decidable in* nlogspace.

The approach followed in the proof of Theorem 12 is to repeatedly pick a transaction template while maintaining an overall consistent variable mapping in search for a counter-example multiversion split schedule that by Theorem 7 suffices to show that robustness does not hold. The main challenge is to show that a variable mapping consistent with all functional constraints can be maintained in logarithmic space and that all requirements for a multiversion split schedule can be verified in nlogspace.

Central to our approach is a generalization of conflicting operations. Let $\mathcal{P}$ be a set of transaction templates. For τ_i and τ_j in $\mathcal{P}$, we say that an operation $o_i \in \tau_i$ is *potentially conflicting* with an operation $o_j \in \tau_j$ if o_i and o_j are operations over a variable of the same type, and at least one of the following holds:

- $\text{WriteSet}(o_i) \cap \text{WriteSet}(o_j) \neq \emptyset$ (potentially ww-conflicting);
- $\text{WriteSet}(o_i) \cap \text{ReadSet}(o_j) \neq \emptyset$ (potentially wr-conflicting); or
- $\text{ReadSet}(o_i) \cap \text{WriteSet}(o_j) \neq \emptyset$ (potentially rw-conflicting).

Intuitively, potentially conflicting operations lead to conflicting operations when the variables of these operations are mapped to the same tuple by a variable assignment. In analogy to conflicting quadruples over a set of transactions as in Definition 6, we consider *potentially conflicting quadruples* $(\tau_i, o_i, p_j, \tau_j)$ over $\mathcal{P}$ with $\tau_i, \tau_j \in \mathcal{P}$, and $o_i \in \tau_i$ an operation that is potentially conflicting with an operation $p_j \in \tau_j$. For a sequence of potentially conflicting quadruples $D = (\tau_1, o_1, p_2, \tau_2), \ldots, (\tau_m, o_m, p_1, \tau_1)$ over $\mathcal{P}$, we write $\text{Trans}(D)$ to denote the set $\{\tau_1, \ldots, \tau_m\}$ of transaction templates mentioned in D. For ease of exposition, we assume a variable renaming such that any pair of templates in $\text{Trans}(D)$ uses a disjoint set of variables.[4] The sequence D induces a sequence of conflicting quadruples $C = (T_1, b_1, a_2, T_2), \ldots, (T_m, b_m, a_1, T_1)$ by applying a variable assignment μ_i to each τ_i in $\text{Trans}(D)$. We call such a set of variable assignments simply a *variable mapping* for D, denoted $\bar{\mu}$, and write $\bar{\mu}(D) = C$. For a variable $\mathtt{X}$ occurring in a template τ_i, we write $\bar{\mu}(\mathtt{X})$ as a shorthand notation for $\mu_i(\mathtt{X})$, with μ_i the variable assignment over τ_i in $\bar{\mu}$. This is well-defined as all templates in $\text{Trans}(D)$ are variable-disjoint. Furthermore, $\bar{\mu}(var(o_i)) = \bar{\mu}(var(p_j))$ for each potentially conflicting quadruple $(\tau_i, o_i, p_j, \tau_j)$ in D as otherwise the induced quadruple (T_i, b_i, a_j, T_j) is not a valid conflicting quadruple in C. We say that a variable mapping $\bar{\mu}$ is admissible for a database $\mathbf{D}$ if every variable assignment μ_i in $\bar{\mu}$ is admissible for $\mathbf{D}$.

A basic insight is that if there is a multiversion split schedule s for some C over a set of transactions $\mathcal{T}$ consistent with $\mathcal{P}$ and a database $\mathbf{D}$, then there is a sequence of potentially conflicting quadruples D such that $\bar{\mu}(D) = C$ for some $\bar{\mu}$. We will verify the existence of such a C, satisfying the properties of Definition 6, by nondeterministically constructing D on-the-fly together with a mapping $\bar{\mu}$. We show in Lemma 14 that when $\mathcal{P} \in \mathbf{MTBTemp}$, $\bar{\mu}$ is a collection of disjoint type mappings (that map variables of the same type to the same tuple) such that variables that are "connected" in D (in a way that we will make precise next) are mapped using the same type mapping. Lemma 15 then shows that already a constant number of those type mappings suffice.

We introduce the necessary notions to capture when two variables are connected in D. We can think of equality constraints $\mathtt{Y} = f(\mathtt{X})$ in a template τ as constraints on the possible variable assignments μ for τ when a database $\mathbf{D}$ is given. Indeed, if we fix $\mu(\mathtt{X})$ to a tuple in $\mathbf{D}$, then $\mu(\mathtt{Y}) = f^{\mathbf{D}}(\mu(\mathtt{X}))$ is immediately implied. These constraints can cause a chain reaction of implications. If for example $\mathtt{Z} = g(\mathtt{Y})$ is a constraint in τ as well, then $\mu(\mathtt{X})$ immediately implies $\mu(\mathtt{Z}) = g^{\mathbf{D}}(f^{\mathbf{D}}(\mu(\mathtt{X})))$. We formalize this notion of implication next. We use sequences of function names $F = f_1 \cdots f_n$, denoting the empty sequence as ε and the concatenation of two sequences F and G by $F \cdot G$. For two variables $\mathtt{X}, \mathtt{Y}$ occurring in a template τ and a (possibly empty) sequence of function names F, we say that $\mathtt{X}$ *implies* $\mathtt{Y}$ *by* F *in* τ, denoted $\mathtt{X} \overset{F}{\rightsquigarrow}_\tau \mathtt{Y}$, if $\mathtt{X} = \mathtt{Y}$ and $F = \varepsilon$ or if there is a variable $\mathtt{Z}$ such that $\mathtt{Y} = f(\mathtt{Z})$ is a constraint in τ, $\mathtt{X} \overset{F'}{\rightsquigarrow}_\tau \mathtt{Z}$ and $F = F' \cdot f$. We next extend the notions of implication to sequences of potentially conflicting quadruples. Let $D = (\tau_1, o_1, p_2, \tau_2), \ldots, (\tau_m, o_m, p_1, \tau_1)$ be a sequence of potentially conflicting quadruples, and let $\mathtt{X}$ and $\mathtt{Y}$ be two variables occurring in templates τ_i and τ_j in $\text{Trans}(D)$, respectively. Then $\mathtt{X}$ *implies* $\mathtt{Y}$ *by a sequence of function names* F *in* D, denoted $\mathtt{X} \overset{F}{\rightsquigarrow}_D \mathtt{Y}$ if

[4] To be formally correct, the latter would require to add every such variable-renamed template to $\mathcal{P}$ creating a larger set $\mathcal{P}'$. This does not influence the complexity of Theorem 12 as $\text{Trans}(D)$ nor $\mathcal{P}'$ are used in the algorithm. Their only purpose is to reason about properties of $\bar{\mu}$.

- $i = j$ and $\mathtt{X} \overset{F}{\rightsquigarrow}_{\tau_i} \mathtt{Y}$ (implication within the same template);
- $F = \varepsilon$ and $(\tau_i, o_i, p_j, \tau_j)$ or $(\tau_j, o_j, p_i, \tau_i)$ is a potentially conflicting quadruple in D with o_i (respectively p_i) an operation over $\mathtt{X}$ and p_j (respectively o_j) an operation over $\mathtt{Y}$ (implication between templates, notice that $\mathtt{X} \overset{\varepsilon}{\rightsquigarrow}_D \mathtt{Y}$ iff $\mathtt{Y} \overset{\varepsilon}{\rightsquigarrow}_D \mathtt{X}$); or
- there exists a variable $\mathtt{Z}$ such that $\mathtt{X} \overset{F_1}{\rightsquigarrow}_D \mathtt{Z}$ and $\mathtt{Z} \overset{F_2}{\rightsquigarrow}_D \mathtt{Y}$ with $F = F_1 \cdot F_2$.

Two variables $\mathtt{X}$ and $\mathtt{Y}$ occurring in Trans(D) are *connected in* D, denoted $\mathtt{X} \approx\!\approx_D \mathtt{Y}$, if $\mathtt{X} \overset{F}{\rightsquigarrow}_D \mathtt{Y}$ or $\mathtt{Y} \overset{F}{\rightsquigarrow}_D \mathtt{X}$, or if there is a variable $\mathtt{Z}$ with $\mathtt{X} \approx\!\approx_D \mathtt{Z}$ and either $\mathtt{Z} \overset{F}{\rightsquigarrow}_D \mathtt{Y}$ or $\mathtt{Y} \overset{F}{\rightsquigarrow}_D \mathtt{Z}$ for some sequence F. Furthermore, two variables $\mathtt{X}$ and $\mathtt{Y}$ occurring in a template τ are *connected in* τ, denoted $\mathtt{X} \approx\!\approx_\tau \mathtt{Y}$, if $\mathtt{X} \overset{F}{\rightsquigarrow}_\tau \mathtt{Y}$ or $\mathtt{Y} \overset{F}{\rightsquigarrow}_\tau \mathtt{X}$, or if there is a variable $\mathtt{Z}$ with $\mathtt{X} \approx\!\approx_\tau \mathtt{Z}$ and either $\mathtt{Z} \overset{F}{\rightsquigarrow}_\tau \mathtt{Y}$ or $\mathtt{Y} \overset{F}{\rightsquigarrow}_\tau \mathtt{Z}$ for some sequence F. These definitions of connectedness can be trivially extended to operations over variables: two operations in D (respectively τ) are connected in D (respectively τ) if they are over variables that are connected in D (respectively τ). When F is not important we drop it from the notation. For instance, we denote by $\mathtt{X} \rightsquigarrow_D \mathtt{Y}$ that there is an F with $\mathtt{X} \overset{F}{\rightsquigarrow}_D \mathtt{Y}$.

▶ **Lemma 13.** *Let D be a sequence of potentially conflicting quadruples over $\mathcal{P} \in \boldsymbol{MTBTemp}$. Then $\mathtt{X} \approx\!\approx_D \mathtt{Y}$ implies $\mathtt{X} \rightsquigarrow_D \mathtt{Y}$ and $\mathtt{Y} \rightsquigarrow_D \mathtt{X}$. Furthermore, if $type(\mathtt{X}) = type(\mathtt{Y})$ then $\bar{\mu}(\mathtt{X}) = \bar{\mu}(\mathtt{Y})$ for every variable mapping $\bar{\mu}$ for D that is admissible for some database $\boldsymbol{D}$.*

It follows from Lemma 13 that, if we group connected variables, then the same tuple is assigned to all variables of the same type in this group. We encode this choice of tuples for variables through (total) functions $c : \mathsf{Rels} \to \mathbf{Tuples}$ that we call *type mappings* and which map a relation onto a particular tuple of that relation's type. For instance, in SmallBank, a type mapping c is determined by an Account tuple $\mathtt{a}$, a Savings tuple $\mathtt{s}$, and a Checking tuple $\mathtt{c}$. The following lemma makes explicit how $\bar{\mu}$ can be decomposed into type mappings such that connected variables use the same type mapping and disequalities enforce the use of different type mappings.

▶ **Lemma 14.** *For a multiversion split schedule s based on a sequence of conflicting quadruples C over a set of transactions $\mathcal{T}$ consistent with a $\mathcal{P} \in \boldsymbol{MTBTemp}$ and a database $\boldsymbol{D}$, let $\bar{\mu}$ be the variable mapping for a sequence of potentially conflicting quadruples D over $\mathcal{P}$ with $\bar{\mu}(D) = C$. Then, a set $\mathcal{S}$ of type mappings over disjoint ranges and a function $\varphi_{\mathcal{S}} : \mathbf{Var} \to \mathcal{S}$ exist with:*

- *$\bar{\mu}(\mathtt{X}) = c(type(\mathtt{X}))$ for every variable $\mathtt{X}$, with $c = \varphi_{\mathcal{S}}(\mathtt{X})$;*
- *$\varphi_{\mathcal{S}}(\mathtt{X}) = \varphi_{\mathcal{S}}(\mathtt{Y})$ whenever $\mathtt{X} \approx\!\approx_D \mathtt{Y}$; and,*
- *$\varphi_{\mathcal{S}}(\mathtt{X}) \neq \varphi_{\mathcal{S}}(\mathtt{Y})$ for every constraint $\mathtt{X} \neq \mathtt{Y}$ occurring in a template $\tau \in Trans(D)$.*

From $D = (\tau_1, o_1, p_2, \tau_2), \ldots, (\tau_m, o_m, p_1, \tau_1)$ and $\varphi_{\mathcal{S}}$ as in Lemma 14 we can derive a sequence of quintuples $E = (\tau_1, o_1, c_{o_1}, p_1, c_{p_1}), \ldots, (\tau_m, o_m, c_{o_m}, p_m, c_{p_m})$ such that $c_{o_i} = \varphi_{\mathcal{S}}(var(o_i))$ and $c_{p_i} = \varphi_{\mathcal{S}}(var(p_i))$ for $i \in [1, m]$. Intuitively, this sequence of quintuples can be used to reconstruct the original multiversion split schedule s. The next lemma shows that we can decide robustness against RC over a set of transaction templates admitting multi-tree bijectivity by searching for a specific sequence of quintuples over at most four type mappings.

▶ **Lemma 15.** *Let $\mathcal{P} \in \boldsymbol{MTBTemp}$ and let $\mathcal{S} = \{c_1, c_2, c_3, c_4\}$ be a set consisting of four type mappings with disjoint ranges. Then, $\mathcal{P}$ is not robust against RC iff there is a sequence of quintuples $E = (\tau_1, o_1, c_{o_1}, p_1, c_{p_1}), \ldots, (\tau_m, o_m, c_{o_m}, p_m, c_{p_m})$ with $m \geq 2$ such that for each quintuple $(\tau_i, o_i, c_{o_i}, p_i, c_{p_i})$ in E:*

1. *o_i and p_i are operations in τ_i, and $c_{o_i}, c_{p_i} \in \mathcal{S}$;*
2. *$\mathtt{X}_i \not\approx\!\approx_{\tau_i} \mathtt{Y}_i$ for each constraint $\mathtt{X}_i \neq \mathtt{Y}_i$ in τ_i;*
3. *$c_{o_i} = c_{p_i}$ if $o_i \approx\!\approx_{\tau_i} p_i$;*

4. $c_{o_i} \neq c_{p_i}$ *if there is a constraint* $\mathtt{X}_i \neq \mathtt{Y}_i$ *in* τ_i *with* $\mathtt{X}_i \approx\!\approx_{\tau_i} var(o_i)$ *and* $\mathtt{Y}_i \approx\!\approx_{\tau_i} var(p_i)$*;*
5. *if* $i \neq 1$ *and* $c_{q_i} = c_{q_1}$ *for some* $q_i \in \{o_i, p_i\}$ *and* $q_1 \in \{o_1, p_1\}$*, then there is no operation* o'_i *in* τ_i *potentially ww-conflicting with an operation* o'_1 *in* $\mathsf{prefix}_{o_1}(\tau_1)$ *with* $var(o'_i) \approx\!\approx_{\tau_i} var(q_i)$ *and* $var(o'_1) \approx\!\approx_{\tau_1} var(q_1)$*.*

Furthermore, for each pair of adjacent quintuples $(\tau_i, o_i, c_{o_i}, p_i, c_{p_i})$ *and* $(\tau_j, o_j, c_{o_j}, p_j, c_{p_j})$ *in* E *with* $j = i + 1$*, or* $i = m$ *and* $j = 1$*:*

6. o_i *is potentially conflicting with* p_j *and* $c_{o_i} = c_{p_j}$*;*
7. *if* $i = 1$ *and* $j = 2$*, then* o_1 *is potentially rw-conflicting with* p_2*; and*
8. *if* $i = m$ *and* $j = 1$*, then* $o_1 <_{\tau_1} p_1$ *or* o_m *is potentially rw-conflicting with* p_1*.*

The items have the following meaning: (2) τ_i is satisfiable; (3) connected operations are assigned the same type mapping; (4) variables connected through an inequality are assigned a different type mapping; (5) φ_S only assigns the same type mapping to o_1 or p_1 in τ_1 and o_i or p_i in τ_i if it does not introduce a dirty write in the resulting multiversion split schedule (cf. Condition (1) in Definition 6); (6) each pair of variables in operations used for conflicts are assigned the same type mapping; (7, 8) the operations used for conflicts between τ_1, τ_2 and τ_m are restricted to satisfy respectively Condition (3) and (2) in Definition 6 in the resulting multiversion split schedule.

The characterization for t-robustness(**MTBTemp**,RC) in Lemma 15 implies an nlogspace algorithm guessing the counterexample sequence E, thereby proving Theorem 12. Indeed, the algorithm guesses the sequence of quintuples E, verifying all conditions for each newly guessed quintuple while only requiring logarithmic space. Notice in particular that we only need to keep track of two other quintuples when verifying all conditions for the newly guessed quintuple, namely the first quintuple over τ_1 and the quintuple immediately preceding the newly guessed one. As usual, we can think of the encoding of templates and operations mentioned in each quintuple as pointers referring to the corresponding templates and operations on the input tape. Furthermore, we do not encode the four type mappings explicitly as such a representation of a mapping might require polynomial space. Since we are only interested in (dis)equality between type mappings, an encoding where these four type mappings are represented by four arbitrary strings of constant size suffices. More details can be found in Appendix B.4.

6 Robustness for Templates over Acyclic Schemas

We denote by **AcycTemp** the class of all sets of transaction templates over acyclic schemas.

▶ **Theorem 16.** t-robustness*(**AcycTemp**,RC) is decidable in* expspace*.*

We provide some intuition for the proof. For a given acyclic schema graph SG, $R \overset{F}{\rightsquigarrow}_{SG} S$ denotes the directed path from node R to node S in SG with F the sequence of edge labels on the path. The next lemma relates implication between variables to paths in SG.

▶ **Lemma 17.** *Let* D *be a sequence of potentially conflicting quadruples over a set of transaction templates* $\mathcal{P} \in$ ***AcycTemp****. For every pair of variables* $\mathtt{X}, \mathtt{Y}$ *occurring in* $Trans(D)$*, if* $\mathtt{X} \overset{F}{\rightsquigarrow}_D \mathtt{Y}$*, then* $type(\mathtt{X}) \overset{F}{\rightsquigarrow}_{SG} type(\mathtt{Y})$*, with* SG *the corresponding schema graph.*

Notice that an assignment of a tuple to a variable $\mathtt{X}$ determines the tuples assigned to all variables $\mathtt{Y}$ with $\mathtt{X} \overset{F}{\rightsquigarrow}_D \mathtt{Y}$ for some sequence of function names F. From Lemma 17 it follows that each such implied tuple is witnessed by a path in the corresponding schema graph SG. Therefore, the maximal number of different tuples implied by $\mathtt{X}$ corresponds to the number of paths in SG starting in type($\mathtt{X}$), which is finite when SG is acyclic. Because there can be multiple paths between nodes in the schema graph, it is no longer the case as in the previous section that variables of the same type connected in D must be assigned the same value. So, instead of using type mappings, we introduce *tuple-contexts* to represent the sets of all tuples implied by the assignment of a given variable. Formally, a *tuple-context for a type* $R \in \mathsf{Rels}$ is a function from paths with source R in $SG(\mathsf{Rels}, \mathsf{Funcs})$ to tuples in **Tuples** of the appropriate type. That is, for each tuple-context c for type R and for each path $R \overset{F}{\rightsquigarrow}_{SG} S$ in SG, $\text{type}(c(R \overset{F}{\rightsquigarrow}_{SG} S)) = S$.

Similar to Lemma 14, we show that we can represent a counterexample schedule based on D by assigning a tuple-context to each variable in $\text{Trans}(D)$, taking special care when assigning contexts to variables connected in D to make sure that they are properly related to each other. For this, we introduce a (partial) function $\varphi_{\mathcal{A}} : \mathbf{Var} \to \mathcal{A}$ mapping (a subset of) variables in $\text{Trans}(D)$ to tuple-contexts in $\mathcal{A}$ (for $\mathcal{A}$ a set of tuple-contexts)and refer to it as a *(partial) context assignment for* D *over* $\mathcal{A}$. In a sequence of lemma's, we show that $\varphi_{\mathcal{A}}$ can always be expanded into a total function and an approach based on enumeration of quintuples analogous to Lemma 15 suffices to decide robustness. A major difference with the previous section is that there is no longer a constant bound on the number of tuple-contexts that are needed and consistency between between tuple-contexts in connected variables needs to be maintained. A full proof can be found in Appendix C.

Next, we consider restrictions that lower the complexity. To this end, we say that two variables $\mathtt{X}$ and $\mathtt{Y}$ occurring in a transaction template τ are *equivalent in* τ, denoted $\mathtt{X} \equiv_\tau \mathtt{Y}$ if

- $\mathtt{X} = \mathtt{Y}$;
- there exists a pair of variables $\mathtt{Z}$ and $\mathtt{W}$ in τ and a sequence of function names F with $\mathtt{Z} \equiv_\tau \mathtt{W}$, $\mathtt{Z} \overset{F}{\rightsquigarrow}_\tau \mathtt{X}$ and $\mathtt{W} \overset{F}{\rightsquigarrow}_\tau \mathtt{Y}$; or
- there exists a variable $\mathtt{Z}$ with $\mathtt{X} \equiv_\tau \mathtt{Z}$ and $\mathtt{Y} \equiv_\tau \mathtt{Z}$.

Then, a transaction template τ is *restricted* if for every combination of variables $\mathtt{X}, \mathtt{Y}, \mathtt{W}, \mathtt{Z}$ in τ with $\mathtt{X} \rightsquigarrow_\tau \mathtt{W}$ and $\mathtt{Y} \rightsquigarrow_\tau \mathtt{Z}$, either $\mathtt{W} \equiv_\tau \mathtt{Z}$, $\mathtt{W} \rightsquigarrow_\tau \mathtt{Z}$ or $\mathtt{Z} \rightsquigarrow_\tau \mathtt{W}$. We denote by **AcycResTemp** the class of all sets of restricted transaction templates over acyclic schemas.

▶ **Theorem 18.** **1.** T-ROBUSTNESS*(**AcycResTemp**,RC) is decidable in* EXPTIME.
2. T-ROBUSTNESS*(**AcycTemp**,RC) is decidable in* PSPACE *when the number of paths between any two nodes in the schema graph is bounded by a constant k.*

Regarding (1), all templates in TPC-C with the exception of NewOrder are restricted. Regarding (2), when the schema graph is a multi-tree then $k = 1$ and for TPC-C $k = 2$ (recall that in general there can be an exponential number of paths), leading to a more practical algorithm for robustness in those cases.

7 Related Work

Transaction Programs. Previous work on static robustness testing [13, 3] for transaction programs is based on the following key insight: when a *schedule* is not serializable, then the dependency graph constructed from that schedule contains a cycle satisfying a condition specific to the isolation level at hand (*dangerous structure* for SNAPSHOT ISOLATION and the presence of a *counterflow edge* for RC). That insight is extended to a workload of *transaction programs* through the construction of a so-called static dependency graph where each program is represented by a node, and there is a conflict edge from one program to another if there can be a schedule that gives rise to that conflict. The absence of a cycle satisfying the condition specific to that isolation level then guarantees robustness while the presence of a cycle does not necessarily imply non-robustness.

Other work studies robustness within a framework for uniformly specifying different isolation levels in a declarative way [8, 7, 9]. A key assumption here is *atomic visibility* requiring that either all or none of the updates of each transaction are visible to other transactions. These approaches aim at higher isolation levels and cannot be used for RC, as RC does not admit *atomic visibility*.

Transaction Templates. The static robustness approach based on transaction templates [18] differs in two ways. First, it makes more underlying assumptions explicit within the formalism of transaction templates (whereas previous work departs from the static dependency graph that should be constructed in some way by the dba). Second, it allows for a decision procedure that is sound and complete for robustness testing against RC, allowing to detect larger subsets of transactions to be robust [18].

The formalisation of transactions and conflict serializability in [18] and this paper is based on [12], generalized to operations over attributes of tuples and extended with `U`-operations that combine `R`- and `W`-operations into one atomic operation. These definitions are closely related to the formalization presented by Adya et al. [1], but we assume a total rather than a partial order over the operations in a schedule. There are also a few restrictions to the model: there needs to be a fixed set of read-only attributes that cannot be updated and which are used to select tuples for update. The most typical example of this are primary key values passed to transaction templates as parameters. The inability to update primary keys is not an important restriction in many workloads, where keys, once assigned, never get changed, for regulatory or data integrity reasons.

In [18], a PTIME decision procedure is obtained for robustness against RC for templates without functional constraints and the present paper improves that result to NLOGSPACE. In addition, an experimental study was performed showing how an approach based on robustness and making transactions robust through promotion can improve transaction throughput.

Transactions. Fekete [12] is the first work that provides a necessary and sufficient condition for deciding robustness against SNAPSHOT ISOLATION for a workload of concrete transactions (not transaction programs). That work provides a characterization for acceptable allocations when every transaction runs under either SNAPSHOT ISOLATION or strict two-phase locking (S2PL). The allocation then is acceptable when every possible execution respecting the allocated isolation levels is serializable. As a side result, this work indirectly provides a necessary and sufficient condition for robustness against SNAPSHOT ISOLATION, since robustness against SNAPSHOT ISOLATION holds iff the allocation where each transaction is allocated to SNAPSHOT ISOLATION is acceptable. Ketsman et al. [14] provide full characterisations for robustness against READ COMMITTED and READ UNCOMMITTED under lock-based semantics. In addition, it is shown that the corresponding decision problems are complete for coNP and LOGSPACE, respectively, which should be contrasted with the polynomial time characterization obtained in [18] for robustness against *multiversion* read committed.

8 Conclusion

This paper falls within a more general research line investigating how transaction throughput can be improved through an approach based on robustness testing that can be readily applied without making any changes to the underlying database system. As argued in Section 2, incorporating functional constraints can detect larger sets of templates to be robust and requires less `R`-operations to be promoted to `U`-operations. In future work, we plan to look at lower bounds, restrictions that lower complexity, and consider other referential integrity constraints to further enlarge the modelling power of transaction templates.

References

1. Atul Adya, Barbara Liskov, and Patrick E. O'Neil. Generalized isolation level definitions. In *ICDE*, pages 67–78, 2000.
2. Mohammad Alomari, Michael Cahill, Alan Fekete, and Uwe Rohm. The cost of serializability on platforms that use snapshot isolation. In *ICDE*, pages 576–585, 2008.
3. Mohammad Alomari and Alan Fekete. Serializable use of read committed isolation level. In *AICCSA*, pages 1–8, 2015.
4. Sidi Mohamed Beillahi, Ahmed Bouajjani, and Constantin Enea. Checking robustness against snapshot isolation. In *CAV*, pages 286–304, 2019.
5. Sidi Mohamed Beillahi, Ahmed Bouajjani, and Constantin Enea. Robustness against transactional causal consistency. In *CONCUR*, pages 1–18, 2019.
6. Hal Berenson, Philip A. Bernstein, Jim Gray, Jim Melton, Elizabeth J. O'Neil, and Patrick E. O'Neil. A critique of ANSI SQL isolation levels. In *SIGMOD*, pages 1–10, 1995.
7. Giovanni Bernardi and Alexey Gotsman. Robustness against consistency models with atomic visibility. In *CONCUR*, pages 7:1–7:15, 2016.
8. Andrea Cerone, Giovanni Bernardi, and Alexey Gotsman. A framework for transactional consistency models with atomic visibility. In *CONCUR*, pages 58–71, 2015.
9. Andrea Cerone and Alexey Gotsman. Analysing snapshot isolation. *J.ACM*, 65(2):1–41, 2018.
10. Andrea Cerone, Alexey Gotsman, and Hongseok Yang. Algebraic Laws for Weak Consistency. In *CONCUR*, pages 26:1–26:18, 2017.
11. Ashok K. Chandra and Moshe Y. Vardi. The implication problem for functional and inclusion dependencies is undecidable. *SIAM J. Comput.*, 14(3):671–677, 1985. URL: `https://doi.org/10.1137/0214049`, `doi:10.1137/0214049`.
12. Alan Fekete. Allocating isolation levels to transactions. In *PODS*, pages 206–215, 2005.
13. Alan Fekete, Dimitrios Liarokapis, Elizabeth J. O'Neil, Patrick E. O'Neil, and Dennis E. Shasha. Making snapshot isolation serializable. *ACM Trans. Database Syst.*, 30(2):492–528, 2005.
14. Bas Ketsman, Christoph Koch, Frank Neven, and Brecht Vandevoort. Deciding robustness for lower SQL isolation levels. In *PODS*, pages 315–330, 2020.
15. Christos H. Papadimitriou. *The Theory of Database Concurrency Control.* Computer Science Press, 1986.
16. Emil L. Post. A variant of a recursively unsolvable problem. *Bull. Amer. Math. Soc.*, pages 264–268, 1946.
17. TPC-C. On-line transaction processing benchmark. `http://www.tpc.org/tpcc/`.
18. Brecht Vandevoort, Bas Ketsman, Christoph Koch, and Frank Neven. Robustness against read committed for transaction templates (full version of submitted paper). `https://github.com/fneven-uh/paper`, 2021.

A Proofs for Section 4

Here, we present the proof of Theorem 10.

▶ **Theorem 10.** T-ROBUSTNESS*(**EqTemp**,RC) is undecidable.*

A domino is a pair $(\mathbf{a}, \mathbf{b})$ of two non-empty strings over Σ. Henceforth we call $\mathbf{a}$ its *top value* and $\mathbf{b}$ its *bottom value*. Given a set of dominoes $\mathcal{D}$, the PCP asks if a non-empty sequence $d_1, d_2, \ldots, d_r$ of dominoes in $\mathcal{D}$ exists such that, with $d_i = (\mathbf{a_i}, \mathbf{b_i})$, the strings $\mathbf{a_1 a_2} \ldots \mathbf{a_r}$ and $\mathbf{b_1 b_2} \ldots \mathbf{b_r}$ are identical.

For the reduction to non-robustness against RC, we construct a set $\mathcal{P}$ of transaction templates consisting of the transaction templates in Figure 7 for $\mathcal{D}$. There are the transactions Split, First and Last (whose meaning will be explained next) and for every domino in $\mathcal{D}$ there is a template in Figure 7 representing that domino and the action of appending that domino to a sequence of dominoes. The schema consists of the relations $\{$`Boolean`, `InitialConflict`, `String`, `PCPSolution`, `DominoSequence`$\}$ whose meaning will be explained below together with a discussion of all the functions. The schema graph is presented in Figure 6 and contains various cycles.

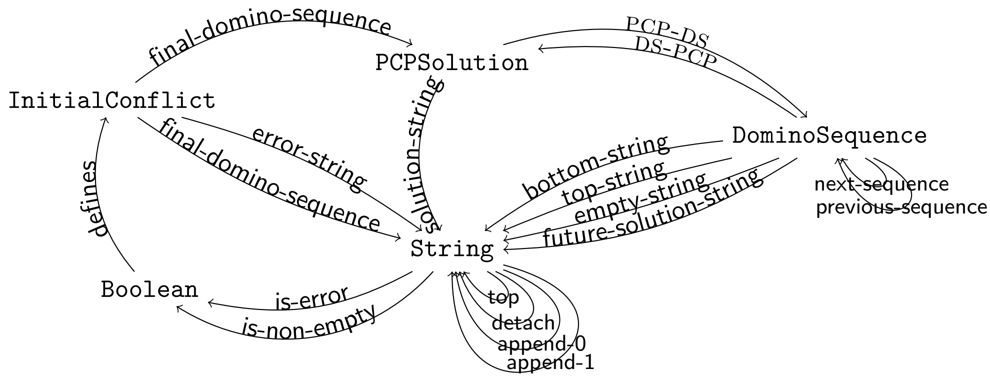

Figure 6 Schema graph for the transaction templates in Figure 7 (for any set of dominoes).

To prove Theorem 10, we will show that there is a solution for PCP if and only if $\mathcal{P}$ is not robust against RC. For the only-if direction, we show that, if there is a solution $\mathbf{d} = d_1, d_2, \ldots, d_r$ for the PCP problem over $\mathcal{D}$, then there is a multiversion split schedule that encodes this solution in a particular way: in this schedule the split transaction is an instantiation of transaction template Split, the next transaction is an instantiation of First, then followed by instantiations of transaction templates $\text{Domino}_{\mathbf{d_1}}, \ldots, \text{Domino}_{\mathbf{d_r}}$ representing the sequence of dominoes in solution $\mathbf{d}$, and finally an instantiation of transaction template Last. Henceforth, we call a schedule that encodes a sequence of dominoes $\mathbf{d}$ in this way a *schedule-encoding of* $\mathbf{d}$. For the if-direction, we first show that every multiversion split schedule consistent with the transaction templates in Figure 7 for some set $\mathcal{D}$ of dominoes is a schedule-encoding for some sequence $\mathbf{d}$ of dominoes from $\mathcal{D}$, and then that for every schedule-encoding of a sequence $\mathbf{d}$ of dominoes, $\mathbf{d}$ is always a solution for the PCP problem over a set of dominoes containing those in $\mathbf{d}$.

Split($\mathtt{I}$):

$\mathtt{W}[\mathtt{X}_1 : \mathtt{Boolean}]$
$\mathtt{R}[\mathtt{I} : \mathtt{InitialConflict}]$
$\mathtt{R}[\mathtt{S}_1 : \mathtt{String}]$
$\mathtt{R}[\mathtt{S}_e : \mathtt{String}]$
$\mathtt{W}[\mathtt{C} : \mathtt{PCPSolution}]$
$\mathtt{X}_1 = f_{\text{is-non-empty}}(\mathtt{S}_1)$
$\mathtt{X}_1 = f_{\text{is-error}}(\mathtt{S}_e)$
$\mathtt{C} = f_{\text{final-domino-sequence}}(\mathtt{I})$
$\mathtt{S}_1 = f_{\text{final-dominoes-string}}(\mathtt{I})$
$\mathtt{S}_e = f_{\text{error-string}}(\mathtt{I})$
$\mathtt{S}_1 = f_{\text{solution-string}}(\mathtt{C})$
$\mathtt{I} = f_{\text{defines}}(\mathtt{X}_1)$

First($\mathtt{S}_1$):

$\mathtt{W}[\mathtt{X}_2 : \mathtt{Boolean}]$
$\mathtt{W}[\mathtt{I} : \mathtt{InitialConflict}]$
$\mathtt{R}[\mathtt{S}_0 : \mathtt{String}]$
$\mathtt{R}[\mathtt{S}_e : \mathtt{String}]$
$\mathtt{R}[\mathtt{S}_1 : \mathtt{String}]$
$\mathtt{W}[\mathtt{B} : \mathtt{DominoSequence}]$
$\mathtt{X}_2 = f_{\text{is-non-empty}}(\mathtt{S}_0)$
$\mathtt{X}_2 = f_{\text{is-error}}(\mathtt{S}_0)$
$\mathtt{S}_1 = f_{\text{final-dominoes-string}}(\mathtt{I})$
$\mathtt{S}_0 = f_{\text{top-string}}(\mathtt{B})$
$\mathtt{S}_0 = f_{\text{bottom-string}}(\mathtt{B})$
$\mathtt{S}_0 = f_{\text{empty-string}}(\mathtt{B})$
$\mathtt{S}_1 = f_{\text{future-solution-string}}(\mathtt{B})$
$\mathtt{S}_e = f_{\text{detach}}(\mathtt{S}_0)$
$\mathtt{S}_e = f_{\text{detach}}(\mathtt{S}_e)$
$\mathtt{I} = f_{\text{defines}}(\mathtt{X}_2)$
$\mathtt{B} = f_{\text{empty-domino-sequence}}(\mathtt{S}_1)$

Last($\mathtt{B}$):

$\mathtt{W}[\mathtt{B} : \mathtt{DominoSequence}]$
$\mathtt{R}[\mathtt{S}_t : \mathtt{String}]$
$\mathtt{R}[\mathtt{S}_b : \mathtt{String}]$
$\mathtt{R}[\mathtt{S}_1 : \mathtt{String}]$
$\mathtt{W}[\mathtt{C} : \mathtt{PCPSolution}]$
$\mathtt{S}_t = f_{\text{top-string}}(\mathtt{B})$
$\mathtt{S}_b = f_{\text{bottom-string}}(\mathtt{B})$
$\mathtt{S}_1 = f_{\text{future-solution-string}}(\mathtt{B})$
$\mathtt{C} = f_{\text{DS}\to\text{PCP}}(\mathtt{B})$
$\mathtt{S}_t = f_{\text{solution-string}}(\mathtt{C})$
$\mathtt{S}_b = f_{\text{solution-string}}(\mathtt{C})$
$\mathtt{S}_1 = f_{\text{solution-string}}(\mathtt{C})$
$\mathtt{B} = f_{\text{PCP}\to\text{DS}}(\mathtt{C})$

For every domino $d_i = (a_1 a_2 \ldots a_h, b_1 b_2 \ldots b_k) \in \mathcal{D}$ a transaction template Domino$_\mathtt{i}$($\mathtt{B}$):

$\mathtt{W}[\mathtt{B} : \mathtt{DominoSequence}]$
$\mathtt{R}[\mathtt{S}_0 : \mathtt{String}]$
$\mathtt{R}[\mathtt{S}_1 : \mathtt{String}]$
$\mathtt{R}[\mathtt{S}_\mathtt{t} : \mathtt{String}]$
$\mathtt{R}[\mathtt{S}_{\mathtt{t}a_1} : \mathtt{String}]$
$\mathtt{R}[\mathtt{S}_{\mathtt{t}a_1 a_2} : \mathtt{String}]$
$\ldots$
$\mathtt{R}[\mathtt{S}_{\mathtt{t}a_1 a_2 \ldots a_h} : \mathtt{String}]$
$\mathtt{R}[\mathtt{S}_\mathtt{b} : \mathtt{String}]$
$\mathtt{R}[\mathtt{S}_{\mathtt{b}b_1} : \mathtt{String}]$
$\mathtt{R}[\mathtt{S}_{\mathtt{b}b_1 b_2} : \mathtt{String}]$
$\ldots$
$\mathtt{R}[\mathtt{S}_{\mathtt{b}b_1 b_2 \ldots b_k} : \mathtt{String}]$
$\mathtt{W}[\mathtt{B}_\mathtt{next} : \mathtt{DominoSequence}]$

$\mathtt{S}_\mathtt{t} = f_{\text{top-string}}(\mathtt{B})$
$\mathtt{S}_{\mathtt{t}a_1} = f_{\text{append-}a_1}(\mathtt{S}_\mathtt{t})$
$\mathtt{S}_{\mathtt{t}a_1 a_2} = f_{\text{append-}a_2}(\mathtt{S}_{\mathtt{t}a_1})$
$\ldots$
$\mathtt{S}_{\mathtt{t}a_1 a_2 \ldots a_h} = f_{\text{append-}a_h}(\mathtt{S}_{\mathtt{t}a_1 \ldots a_{h-1}})$
$\mathtt{S}_\mathtt{t} = f_{\text{detach}}(\mathtt{S}_{\mathtt{t}a_1})$
$\mathtt{S}_{\mathtt{t}a_1} = f_{\text{detach}}(\mathtt{S}_{\mathtt{t}a_1 a_2})$
$\ldots$
$\mathtt{S}_{\mathtt{t}a_1 a_2 \ldots a_{h-1}} = f_{\text{detach}}(\mathtt{S}_{\mathtt{t}a_1 a_2 \ldots a_h})$
$\mathtt{S}_{\mathtt{t}a_1 a_2 \ldots a_h} = f_{\text{top-string}}(\mathtt{B}_{\text{next}})$
$\mathtt{S}_{a_1} = f_{\text{top}}(\mathtt{S}_{\mathtt{t}a_1})$
$\mathtt{S}_{a_2} = f_{\text{top}}(\mathtt{S}_{\mathtt{t}a_1 a_2})$
$\ldots$
$\mathtt{S}_{a_h} = f_{\text{top}}(\mathtt{S}_{\mathtt{t}a_1 a_2 \ldots a_h})$
$\mathtt{S}_1 = f_{\text{future-solution-string}}(\mathtt{B})$
$\mathtt{S}_0 = f_{\text{empty-string}}(\mathtt{B})$

$\mathtt{S}_\mathtt{b} = f_{\text{bottom-string}}(\mathtt{B})$
$\mathtt{S}_{\mathtt{b}b_1} = f_{\text{append-}b_1}(\mathtt{S}_\mathtt{b})$
$\mathtt{S}_{\mathtt{b}b_1 b_2} = f_{\text{append-}b_2}(\mathtt{S}_{\mathtt{b}b_1})$
$\ldots$
$\mathtt{S}_{\mathtt{b}b_1 b_2 \ldots b_k} = f_{\text{append-}b_k}(\mathtt{S}_{\mathtt{b}b_1 \ldots b_{k-1}})$
$\mathtt{S}_\mathtt{b} = f_{\text{detach}}(\mathtt{S}_{\mathtt{t}b_1})$
$\mathtt{S}_{\mathtt{b}b_1} = f_{\text{detach}}(\mathtt{S}_{\mathtt{b}b_1 b_2})$
$\ldots$
$\mathtt{S}_{\mathtt{b}b_1 b_2 \ldots b_{k-1}} = f_{\text{detach}}(\mathtt{S}_{\mathtt{b}b_1 b_2 \ldots b_k})$
$\mathtt{S}_{\mathtt{b}b_1 b_2 \ldots b_k} = f_{\text{bottom-string}}(\mathtt{B}_{\text{next}})$
$\mathtt{S}_{b_1} = f_{\text{top}}(\mathtt{S}_{\mathtt{b}b_1})$
$\mathtt{S}_{b_2} = f_{\text{top}}(\mathtt{S}_{\mathtt{b}b_1 b_2})$
$\ldots$
$\mathtt{S}_{b_k} = f_{\text{top}}(\mathtt{S}_{\mathtt{b}b_1 b_2 \ldots b_k})$
$\mathtt{S}_1 = f_{\text{future-solution-string}}(\mathtt{B}_{\text{next}})$
$\mathtt{S}_0 = f_{\text{empty-string}}(\mathtt{B}_{\text{next}})$
$\mathtt{B}_{\text{next}} = f_{\text{next-sequence}}(\mathtt{B})$
$\mathtt{B} = f_{\text{previous-sequence}}(\mathtt{B}_{\text{next}})$

Figure 7 Transaction templates for the proof of Theorem 10.

A.1 Only-if direction for the proof of Theorem 10

▶ **Proposition 19** (Only-if part of Theorem 10). *Let $\mathcal{D}$ be a set of dominoes with a solution $\mathbf{d}$ for the PCP problem for $\mathcal{D}$. Then there exists a schedule-encoding of $\mathbf{d}$ that is consistent with the transaction templates in Figure 7 and some database $\boldsymbol{D}$.*

Proof. Let $\mathbf{d} = d_1, d_2, \ldots, d_r$ be a solution to the PCP problem for $\mathcal{D}$. Let $\mathbf{a_1 a_2} \ldots \mathbf{a_r}$ be the read of top values and $\mathbf{b_1 b_2} \ldots \mathbf{b_r}$ be the read of bottom values, which thus represent an identical string $\mathbf{c} = c_1 \cdots c_n$, with $c_i \in \Sigma$. We now construct a schedule s and database $\mathbf{D}$ as in Definition 6 with transactions based on the transaction templates $\mathcal{P}$ in Figure 7.

Relation PCPSolution contains a tuple that we interpret as the PCP solution $\mathbf{d} = d_1, d_2, \ldots, d_r$. Relation DominoSequence contains $r+1$ tuples, one for every prefix of $\mathbf{d}$, including the empty sequence $()$ and the PCP solution $\mathbf{d}$ itself. For convenience of notation, we will henceforth often represent tuples by their interpretation, which is justified by the fact that every tuple in a particular relation will have a different interpretation, and the relation itself can always be derived from the context (e.g., the function signature).

Since the PCP solution has an interpretation in both the relations PCPSolution and DominoSequence, we assume two functional constraints, $f_{\text{PCP}\to\text{DS}}$: PCPSolution $\to$ DominoSequence and $f_{\text{DS}\to\text{PCP}}$: DominoSequence $\to$ PCPSolution that map these interpretations on one another. That is, $f^{\mathbf{D}}_{\text{PCP}\to\text{DS}}(\mathbf{d}) = \mathbf{d}$ and $f^{\mathbf{D}}_{\text{typecase-to-C}}(\mathbf{d}) = \mathbf{d}$.

Further, we have functions $f_{\text{next-sequence}}$: DominoSequence $\to$ DominoSequence and $f_{\text{previous-sequence}}$: DominoSequence $\to$ DominoSequence with following interpretation:

$$
\begin{aligned}
f^{\mathbf{D}}_{\text{next-sequence}}(\mathbf{d}') &= \mathbf{d}'d && \text{with } d' \text{ a strict prefix of } \mathbf{d} \text{ followed by domino } d \text{ in } \mathbf{d},\\
f^{\mathbf{D}}_{\text{next-sequence}}(\mathbf{d}) &= \mathbf{d},\\
f^{\mathbf{D}}_{\text{previous-sequence}}(\mathbf{d}'d) &= \mathbf{d}' && \text{with } d' \text{ a strict prefix of } \mathbf{d} \text{ followed by domino } d \text{ in } \mathbf{d},\\
f^{\mathbf{D}}_{\text{previous-sequence}}(()) &= ().
\end{aligned}
$$

Relation $\mathsf{String}^{\mathbf{D}}$ contains a tuple representing the read $\mathbf{c}$ of PCP-solution sequence $\mathbf{d}$, a tuple representing an error $\langle\text{error}\rangle$, and a tuple for every substring of $\mathbf{c}$, including the empty string $\langle\rangle$. We assume that all these tuples are different. We use notation $\langle\rangle$ to denote the empty string to distinguish it from $()$, which denotes the empty sequence of dominoes.

Functions $f_{\text{append-0}}$: String $\to$ String, $f_{\text{append-1}}$: String $\to$ String, f_{detach} : String $\to$ String, and f_{top} : String $\to$ String simulate standard string operations for the interpretations of tuples in relation String. Thus, tuples representing a (possibly empty) string $\mathbf{e}$:

$$
f^{\mathbf{D}}_{\text{append-}c}(\langle\mathbf{e}\rangle) = \begin{cases} \langle\mathbf{e}c\rangle & \text{with } \mathbf{e} \text{ a (possibly empty) string over } \Sigma,\ c \in \Sigma, \text{ and}\\ & \langle\mathbf{e}c\rangle \text{ a substring of } \mathbf{c},\\ \langle\text{error}\rangle & \text{otherwise,} \end{cases}
$$

$$
\begin{aligned}
f^{\mathbf{D}}_{\text{detach}}(\langle\mathbf{e}c\rangle) &= \langle\mathbf{e}\rangle \text{ with } \mathbf{e} \text{ a (possibly empty) string over } \Sigma, \text{ and } c \in \Sigma,\\
f^{\mathbf{D}}_{\text{detach}}(\langle\rangle) &= f_{\text{detach}}(\langle\text{error}\rangle) = \langle\text{error}\rangle,\\
f^{\mathbf{D}}_{\text{top}}(\langle\mathbf{e}c\rangle) &= \langle c\rangle \text{ with } \mathbf{e} \text{ a (possibly empty) string over } \Sigma, \text{ and } c \in \Sigma,\\
f^{\mathbf{D}}_{\text{top}}(\langle\rangle) &= f_{\text{top}}(\langle\text{error}\rangle) = \langle\text{error}\rangle.
\end{aligned}
$$

Notice that these function interpretations are closed under $\mathbf{D}$, that is, every tuple from relation $\mathsf{String}^{\mathbf{D}}$ maps onto a tuple that is in relation $\mathsf{String}^{\mathbf{D}}$.

Every tuple in DominoSequence is associated with three tuples in String representing, respectively, the read of top values, the read of bottom values, and the empty string. The association is made via functions $f_{\text{top-string}}$: DominoSequence $\to$ String, $f_{\text{bottom-string}}$:

DominoSequence $\rightarrow$ String, and $f_{\text{empty-string}}$: DominoSequence $\rightarrow$ String with following interpretations in $\mathbf{D}$:

$$f^{\mathbf{D}}_{\text{top-string}}(\mathbf{d}') = \mathbf{e}, \text{ with } \mathbf{e} \text{ the (possibly empty) read of top values on dominoes in } \mathbf{d}',$$
$$f^{\mathbf{D}}_{\text{bottom-string}}(\mathbf{d}') = \mathbf{e}, \text{ with } \mathbf{e} \text{ the (possibly empty) read of bottom values on dominoes in } \mathbf{d}', \text{ and}$$
$$f^{\mathbf{D}}_{\text{empty-string}}(\mathbf{d}') = \langle\rangle.$$

Finally, for function $f_{\text{future-solution-string}}$: DominoSequence $\rightarrow$ String we consider the interpretation that associates every domino sequence $\mathbf{d}'$ represented by a tuple in relation DominoSequence in $\mathbf{D}$ to the final read $f^{\mathbf{D}}_{\text{future-solution-string}}(\mathbf{d}') = \mathbf{c}$. Function $f_{\text{solution-string}}$: PCPSolution $\rightarrow$ String does the same for the single tuple representing $\mathbf{d}$ in PCPSolution, thus with $f^{\mathbf{D}}_{\text{solution-string}}(\mathbf{d}) = \mathbf{c}$. Function $f_{\text{empty-domino-sequence}}$: String $\rightarrow$ DominoSequence is interpreted to map every tuple in String onto the tuple from DominoSequence representing the empty sequence ().

All other relations and functions have as purpose to pass tuples from one transaction to another in a schedule and to enforce that certain tuples do not collide, which is useful for the (if)-part of the proof.

Relation $\text{Boolean}^{\mathbf{D}}$ contains two tuples, which we interpret as Boolean values 0 and 1. Function $f_{\text{is-non-empty}}$: String $\rightarrow$ Boolean and $f_{\text{is-error}}$: String $\rightarrow$ Boolean are interpreted as follows:

$$f^{\mathbf{D}}_{\text{is-non-empty}}(s) = \begin{cases} 1 & \text{if } s \neq \langle\rangle, \\ 0 & \text{otherwise,} \end{cases} \text{, and}$$

$$f^{\mathbf{D}}_{\text{is-error}}(s) = \begin{cases} 1 & \text{if } s = \langle\text{error}\rangle, \\ 0 & \text{otherwise,} \end{cases} \text{, and.}$$

Finally, relation $\text{InitialConflict}^{\mathbf{D}}$ contains a single tuple, which we refer to by $\langle\text{init}\rangle$. The interpretation of f_{defines} : Boolean $\rightarrow$ InitialConflict maps 1 and 0 onto $\langle\text{init}\rangle$. Function $f_{\text{error-string}}$: InitialConflict $\rightarrow$ String maps $\langle\text{init}\rangle$ onto $\langle\text{error}\rangle$. Functions $f_{\text{final-domino-string}}$: InitialConflict $\rightarrow$ DominoSequence and $f_{\text{final-domino-sequence}}$: InitialConfict $\rightarrow$ PCPSolution map $\langle\text{init}\rangle$ onto the solution domino sequence $\mathbf{d}$, respectively on the final read $\mathbf{c}$ of $\mathbf{d}$.

Now the schedule $\mathsf{prefix}_{b_1}(T_1) \cdot T_2 \cdot \ldots \cdot T_m \cdot \mathsf{postfix}_{b_1}(T_1)$, taking $T_1 = \text{Split}(\langle\text{init}\rangle)$, $T_2 = \text{First}(\langle\text{init}\rangle)$, for $i : 1 \leq i \leq r$, transaction $T_{i+2} = \text{Domino}_{\mathtt{i}}((d_1, \ldots, d_i))$, $T_m = \text{Last}((d_1, \ldots, d_r))$ and $b_1 = \langle\text{init}\rangle$ has the conditions of Definition 6. Indeed, it is based on sequence of conflict quadruples $(T_1, \mathtt{R}_1[\langle\text{init}\rangle], \mathtt{W}_2[\langle\text{init}\rangle], T_2), (T_2, \mathtt{W}_2[()], \mathtt{W}_3[()], T_3), (T_3, \mathtt{W}_3[(\mathtt{d}_1)], \mathtt{W}_4[(\mathtt{d}_2)], T_4), \ldots, (T_{r+2}, \mathtt{W}_{r+2}[(\mathtt{d}_1, \ldots, \mathtt{d}_\mathtt{r})], \mathtt{W}_{r+3}[(\mathtt{d}_1, \ldots, \mathtt{d}_\mathtt{r})], T_{r+3}), (T_{r+3}, \mathtt{W}_{r+3}[\mathbf{d}], \mathtt{W}_1[\mathbf{d}], T_1)$.

Condition (1) is true because there is no ww-conflict between a write operation in $\mathsf{prefix}_{b_1}(T_1)$ and a write operation in any of the transactions $T_2, \ldots, T_m$, since the first write operation, respectively second write operation, in $\text{Split}(\langle\text{init}\rangle)$ has a type that only occurs before the conflict with $\text{First}(\langle\text{init}\rangle)$, and is the conflict with $\text{Last}((d_1, \ldots, d_r))$, respectively. Furthermore (2) is true because $b_1 <_{T_1} a_1$ and Condition (3) is true because b_1 and a_2 are rw-conflicting. ◀

A.2 Helpful lemma

▶ **Lemma 20.** *If a set $\mathcal{P}$ of transaction templates is not robust against RC then there is a* multiversion split schedule $\mathsf{prefix}_{b_1}(T_1) \cdot T_2 \cdot \ldots \cdot T_m \cdot \mathsf{postfix}_{b_1}(T_1)$ *for a set $\mathcal{T} = \{T_1, \ldots, T_m\}$ of transactions consistent with $\mathcal{P}$ in which an operation from a transaction T_j depends on an operation from transaction T_i only if $j = i + 1$ or $i = m$ and $j = 1$.*

Proof. If $\mathcal{P}$ is not robust against RC, then there is a database $\mathbf{D}$ and a multiversion split schedule $s = \mathsf{prefix}_{b_1}(T_1) \cdot T_2 \cdot \ldots \cdot T_m \cdot \mathsf{postfix}_{b_1}(T_1) \cdot T_{m+1} \cdot \ldots \cdot T_n$ based on a sequence of conflict quadruples C for a set of transactions $\mathcal{T}$ that is consistent with $\mathcal{P}$ and $\mathbf{D}$ having the properties of Definition 6.

We can assume that $n = m$. Otherwise removing the transactions $T_{m+1}, \ldots, T_n$ from $\mathcal{T}$, s, and C. We can also assume that s is read-last-committed. Otherwise, choosing an appropriate version order $\ll_s$ and version function v_s.

Now suppose that there is a transaction T_j with an operation a'_j that depends on an operation b'_i from transaction T_i and with $j \neq i+1$ or $i = m$ and $j \neq 1$. Clearly, by definition of dependency and the structure of a multiversion split schedule, $i < j$ or $j = 1$.

We proceed the proof by a construction showing that, then, there is an alternative schedule s' that is also a multiversion split schedule, but for a strict subset of transactions in $\mathcal{T}$ (thus also still consistent with $\mathcal{P}$ and $\mathbf{D}$). The result of the lemma then follows from the observation that repeated application of this construction must lead to a schedule with the properties of the lemma, without existence of such a dependency.

For the construction, we proceed by case distinction.

If $i \neq 1$ and $j \neq 1$, we construct a schedule s' from s by removing all operations from transactions T_h with $i < h < j$. Notice that we remove at least one transaction, since $i < i+1 < j$. We can derive a sequence of conflict quadruples C' from C by removing all occurrences of these transactions T_h and adding the conflict quadruple (T_i, b'_i, a'_j, T_j) instead. By construction, s' is a multiversion split schedule based on C' over a set of transactions consistent with $\mathcal{P}$ and $\mathbf{D}$. It remains to show that the newly constructed schedule s' has the properties of Definition 6. The latter is straightforward since C and C' agree on their first and last quadruple, due to assumption $i \neq 1$ and $j \neq 1$.

If $i = 1$, it follows that $i < j$ and thus $j \neq 1$. Then, we construct a schedule s' from s by removing all operations from transactions T_h with $i < h < j$ and updating the prefix and postfix of T_1, now based on b'_i. Notice that we again remove at least one transaction, since $i < i+1 < j$ and that we can derive a sequence of conflict quadruples C' from C in the same way as before, by removing all occurrences of these transactions T_h and adding the conflict quadruple (T_i, b'_i, a'_j, T_j) instead. By construction, s' is a multiversion split schedule based on C' over a set of transactions consistent with $\mathcal{P}$ and $\mathbf{D}$. It remains to show that the newly constructed schedule s' has the properties of Definition 6.

First, we observe that b'_1 and a'_j are rw-conflicting, which immediately implies that Condition (3) is true for s'. The argument is by exclusion. Indeed, if b'_1 and a'_j would be ww-conflicting, then $b'_1 \ll_s a'_j$ implying $b'_1 <_s a'_j$ (due to the assumed read-last committed) and thus $b'_1 \leq_s b_1$, which is not allowed by condition (1) on s. It follows from a similar argument that b'_1 and a'_j are not wr-conflicting: both $b'_1 = v_s(a'_j)$ and $b'_1 \ll_s v_s(a'_j)$ imply $b'_1 <_s C_1 <_s a'_j$, which contradicts with C_1 being the last operation in s.

Since b'_1 is rw-conflicting with a'_j, we have $v_s(b'_1) \ll_s a'_j$, implying $b'_1 <_s a'_j$ (due to read-last-committed and the structure of a multiversion split schedule), thus $b'_1 \leq_s b_1$. Therefore, condition (1) again transfers from s to s'. For similar reasons condition (2) applies on s': If $b_1 <_{T_1} a_1$ then $b'_1 \leq_{T_1} b_1 <_{T_1} a_1$,

Otherwise, if $j = 1$, it follows that $1 < i$. Then, we construct a schedule s' from s by removing all operations from transactions T_h with $i < h$. Notice that we remove at least one transaction, since $i < m$. We can derive a sequence of conflicting quadruples C' from C by removing all occurrences of these transactions T_h and adding the conflicting quadruple (T_i, b'_i, a'_j, T_j) instead.

In this schedule s', condition (1) and (3) transfer from s by its construction. To see that condition (2) is true on s', simply notice that if b'_i and a'_1 are ww or wr-conflicting, then either $b'_i \ll_s a'_j$ or $b'_i = v_s(a'_j)$ or $b'_i \ll_s v_s(a'_j)$, which all imply $b_i <_s C_i <_s a'_1$ and thus that $b_1 <_s a'_1$, implying $b_1 <_{s'} a'_1$. ◀

A.3 If direction for the proof of Theorem 10

A.3.1 First step

Next, we show that, if there exists a multiversion split schedule for the set of transaction templates in Figure 7 for some set $\mathcal{D}$ of dominoes, then this schedule is always a schedule-encoding of a sequence of dominoes in $\mathcal{D}$.

▶ **Proposition 21.** *Let $\mathcal{D}$ be a set of dominoes. If there is a multiversion split schedule s for a set of transactions consistent with the transaction template in Figure 7 for $\mathcal{D}$ and some database $\boldsymbol{D}$, then this schedule s is a schedule-encoding of some sequence $\mathbf{d}$ of dominoes in $\mathcal{D}$.*

For the proof, let $\mathbf{D}$ be a database and $s = \mathsf{prefix}_{b_1}(T_1) \cdot T_2 \cdot \ldots \cdot T_m \cdot \mathsf{postfix}_{b_1}(T_1)$ a multiversion split schedule for a set of transactions $\mathcal{T}$ consistent with $\mathcal{P}$ and $\mathbf{D}$, with the conditions of Lemma 20 and based on some sequence of conflict quadruples $C = (T_1, b_1, a_2, T_2), (T_2, b_2, a_3, T_3) \ldots, (T_m, b_m, a_1, T_1)$. We show through a sequence of properties (Lemmas 23,24, 25, and 26), that s is a schedule-encoding of a sequence $\mathbf{d}$ of dominoes in $\mathcal{D}$.

As a first property (22), we observe that transaction templates in $\mathcal{P}$ heavily constrain the possible variable instantiations. For transaction template Split, for example, a variable mapping depends entirely on the choice of the value for variable $\mathtt{I}$. Since Lemma 20 forbids the presence of duplicate transactions in $\mathcal{T}$, two transactions T_i and T_j (with $i \neq j$) based on transaction template Split cannot agree on their choice for variable $\mathtt{I}$ in s. By applying this argument to other transaction templates, we obtain the following corollary of Lemma 20. Here, for each transaction T_i in s, we write τ_i to denote the transaction template in $\mathcal{P}$ that it is based on, and by μ_i the associated variable mapping for τ_i, with $\mu_i(\tau_i) = T_i$.

▶ **Lemma 22.** *for two transactions T_i and T_j in s, with $i \neq j$:*
- *if T_i and T_j are based on Split, then $\mu_i(\mathtt{I}) \neq \mu_j(\mathtt{I})$;*
- *if T_i and T_j are based on First then $\mu_i(\mathtt{S}_1) \neq \mu_j(\mathtt{S}_1)$;*
- *if T_i and T_j are based on Last then $\mu_i(\mathtt{B}) \neq \mu_j(\mathtt{B})$ and $\mu_i(\mathtt{C}) \neq \mu_j(\mathtt{C})$;*
- *if T_i and T_j are based on domino transaction templates then $\mu_i(\mathtt{B}) \neq \mu_j(\mathtt{B})$ and $\mu_i(\mathtt{B}_{next}) \neq \mu_j(\mathtt{B}_{next})$.*

We conclude the proof of Proposition 21 with the necessary arguments that s is indeed a schedule-encoding for some sequence of dominoes.

▶ **Lemma 23.** *Transaction T_1 is based on Split, T_2 is based on First, and $\mu_1(\mathtt{I}) = \mu_2(\mathtt{I})$, $\mu_1(\mathtt{X}_1) \neq \mu_2(\mathtt{X}_2)$, and $\mu_1(\mathtt{S}_1) = \mu_2(\mathtt{S}_1) \neq \mu_2(\mathtt{S}_0)$.*

Proof. Since b_1 and a_2 are rw-conflicting (cf, Definition 6), and there are no updates in the considered transaction templates, operation b_1 must be a read. Since $\mathsf{InitialConflict}$ is the only type allowing for conflicts involving a read, it is immediate that T_1 must be based on Split and T_2 based on First, with $\mu_1(\mathtt{I}) = \mu_2(\mathtt{I})$. From this equality and function $f_{\text{final-dominoes-string}}$ it follows that $\mu_1(\mathtt{S}_1) = \mu_2(\mathtt{S}_1)$. From Definition 6, particularly that there is no ww-conflict between a write operation in $\mathsf{prefix}_{b_1}(T_1)$ and a write operation in any of the transactions $T_2, \ldots, T_m$, it follows that $\mu_1(\mathtt{X}_1) \neq \mu_2(\mathtt{X}_2)$. Finally, function $f_{\text{is-non-empty}}$, which maps $\mathtt{S}_1$ onto $\mathtt{X}_1$ in transaction template Split and $\mathtt{S}_0$ onto $\mathtt{X}_2$ in transaction template First, implies $\mu_1(\mathtt{S}_1) \neq \mu_2(\mathtt{S}_0)$. ◀

▶ **Lemma 24.** *There is a transaction T_3 in s and it is based on a domino transaction template, with $\mu_3(\mathtt{S}_1) = \mu_2(\mathtt{S}_1)$.*

Proof. First, suppose towards a contradiction that $m = 2$. We already know from Lemma 23 that $\mu_1(\mathtt{I}) = \mu_2(\mathtt{I})$ and $\mu_1(\mathtt{X}_1) \neq \mu_2(\mathtt{X}_2)$, thus $a_1 = b_1 = \mathtt{R}[\mu_1(\mathtt{I})]$ and $b_2 = a_2 = \mathtt{W}[\mu_2(\mathtt{I})]$, indicating $b_1 \to_s a_2$, particularly, $v_s(b_1) \ll_s a_2$, thus implying that a_1 cannot depend on b_2, which is the desired contradiction.

The remainder of the proof is by exclusion. Transaction T_3 is not based on transaction template First, because all possible conflicts between T_2 and an instantiation of transaction template First (implying either $\mu_2(\mathtt{X}_2) = \mu_3(\mathtt{X}_2)$, $\mu_2(\mathtt{B}) = \mu_3(\mathtt{B})$, or $\mu_2(\mathtt{I}) = \mu_3(\mathtt{I})$) would imply the equality $\mu_2(\mathtt{S}_1) = \mu_3(\mathtt{S}_1)$ (through functional constraints $\mathtt{I} = f_{\text{defines}}(\mathtt{X}_2)$, $\mathtt{S}_1 = f_{\text{future-solution-string}}(\mathtt{B})$, and $\mathtt{S}_1 = f_{\text{final-dominoes-string}}(\mathtt{I})$), which is forbidden by Lemma 22. The argument that transaction T_3 cannot based on transaction template Split is similar: every possible conflict between T_2 and an instantiation of Split implies $\mu_1(\mathtt{I}) = \mu_2(\mathtt{I}) = \mu_3(\mathtt{I})$ either directly (taking $\mu_2(\mathtt{I}_2) = \mu_3(\mathtt{I}_1)$ as conflict) or, when taking $\mu_2(\mathtt{X}_2) = \mu_3(\mathtt{X}_1)$ as conflict, through constraints $\mathtt{I} = f_{\text{defines}}(\mathtt{X}_1)$ and $\mathtt{I} = f_{\text{defines}}(\mathtt{X}_2)$ in T_2 and T_3, respectively. Either way, $\mu_1(\mathtt{I}) = \mu_3(\mathtt{I})$ is forbidden by Lemma 22. Finally, to see that T_3 is not based on transaction template Last, we observe that a conflict between T_2 and an instantiation of transaction template Last must be ww-conflicting involving variables $\mathtt{B}$, thus with $\mu_2(\mathtt{B}) = \mu_3(\mathtt{B})$. Then, $\mu_2(\mathtt{S}_0) = \mu_3(\mathtt{S}_t)$, due to functional constraint $\mathtt{S}_0 = f_{\text{top-string}}(\mathtt{B})$ in T_2 and $\mathtt{S}_t = f_{\text{top-string}}(\mathtt{B})$ in T_3, and $\mu_2(\mathtt{S}_1) = \mu_3(\mathtt{S}_1)$, due to functional constraint $\mathtt{S}_1 = f_{\text{future-solution-string}}(\mathtt{B})$ in T_2 and T_3. However, we also have $\mu_3(\mathtt{S}_1) = \mu_3(\mathtt{S}_t)$, due to constraints $\mathtt{S}_1 = f_{\text{solution-string}}(\mathtt{B})$ and $\mathtt{S}_t = f_{\text{solution-string}}(\mathtt{B})$, thus implying $\mu_2(\mathtt{S}_0) = \mu_3(\mathtt{S}_t) = \mu_3(\mathtt{S}_1) = \mu_2(\mathtt{S}_1)$, which contradicts with earlier proven Lemma 23. We conclude that T_3 is indeed based on a domino transaction template. Therefore, the conflict quadruple (T_2, b_2, a_3, T_3) must admit ww-conflicting operations over variable $\mathtt{B}$ in T_2 and either variable $\mathtt{B}$ or $\mathtt{B}_{next}$ in T_3. We notice that $\mu_2(\mathtt{S}_1) = f_{\text{future-solution-string}}(\mu_2(\mathtt{B}))$, $\mu_3(\mathtt{S}_1) = f_{\text{future-solution-string}}(\mu_3(\mathtt{B}))$, and $\mu_3(\mathtt{S}_1) = f_{\text{future-solution-string}}(\mu_3(\mathtt{B}_{next}))$, thus independent of the variable $\mathtt{B}_{next}$ or $\mathtt{B}$ in T_3, we have $\mu_2(\mathtt{S}_1) = \mu_3(\mathtt{S}_1)$. ◀

▶ **Lemma 25.** *For a transaction T_i, with $i \geq 4$, for which all T_j's, with $j \in \{3, \ldots, i-1\}$, are based on domino transaction templates, transaction T_i is based on a domino transaction template or on transaction template Last. Furthermore $\mu_2(\mathtt{S}_1) = \mu_i(\mathtt{S}_1)$.*

Proof. Since domino transaction templates do not mention variables of type InitialConflict and write only to variables of type DominoSequence, it remains to show that T_{i+1} is not based on transaction template First.

For this, observe that $\mu_2(\mathtt{S}_1) = \mu_{i-1}(\mathtt{S}_1)$. Indeed, every conflict quadruple $(T_i, b_i, a_{i+1}, T_{i+1})$, with $i \in \{3, \ldots, i-1\}$, admits ww-conflicting operations with variables of type DominoSequence. No matter if the conflict is via a variable $\mathtt{B}$ or $\mathtt{B}_{next}$, the constraints $\mathtt{S}_1 = f_{\text{future-solution-string}}(\mathtt{B})$ and $\mathtt{S}_1 = f_{\text{future-solution-string}}(\mathtt{B}_{next})$ ensure $\mu_2(\mathtt{S}_1) = \mu_{i-1}(\mathtt{S}_1)$.

Now, assume towards a contradiction that T_{i+1} is based on First, thus admitting a conflict quadruple $(T_i, b_i, a_{i+1}, T_{i+1})$ in C. Then either $b_i = \mu_i(\mathtt{B})$ and $a_{i+1} = \mu_{i+1}(\mathtt{B})$ *or* $b_i = \mu_i(\mathtt{B}_{\text{next}})$ and $a_{i+1} = \mu_{i+1}(\mathtt{B})$. Both of these equalities imply $\mu_i(\mathtt{S}_1) = \mu_{i+1}(\mathtt{S}_1)$ due to constraints $\mathtt{S}_1 = f_{\text{future-solution-string}}(\mathtt{B})$ and $\mathtt{S}_1 = f_{\text{future-solution-string}}(\mathtt{B}_{next})$, thus implying $\mu_i(\mathtt{S}_1) = \mu_2(\mathtt{S}_1)$, this contradict with Lemma 22. We conclude that T_i is indeed based on a domino transaction template or on transaction template Last. That $\mu_{i-1}(\mathtt{S}_1) = \mu_i(\mathtt{S}_1)$ follows again from the constraints using function $f_{\text{future-solution-string}}$. ◀

▶ **Lemma 26.** *If T_i is based on transaction template Last, then $i = m$.*

Proof. Let T_j be the transaction following T_i. We already know about T_j that either $j = 1$ or must be a transaction that is different to all foregoing transactions $T_1, \ldots, T_i$ (due to Lemma 20).

We first show, by exclusion, that transaction T_j is based on Split: Transaction T_j cannot be based on Last, as then either $\mu_i(\mathtt{B}) = \mu_j(\mathtt{B})$ or $\mu_i(\mathtt{C}) = \mu_j(\mathtt{C})$, which directly contradicts Lemma 22. Similarly, transaction T_j cannot be based on First, as then $\mu_i(\mathtt{B}) = \mu_j(\mathtt{B})$ implying $\mu_2(\mathtt{S}_1) = \mu_i(\mathtt{S}_1) = \mu_j(\mathtt{S}_1)$, due to the constraints involving function $f_{\text{future-solution-string}}$. Finally, transaction T_j cannot be based on a domino transaction template, because then $\mu_j(\mathtt{B}_{\text{next}}) = \mu_i(\mathtt{B}) = \mu_j(\mathtt{B})$ or $\mu_{i-1}(\mathtt{B}_{\text{next}}) = \mu_i(\mathtt{B}) = \mu_j(\mathtt{B}_{\text{next}})$, thus with T_i and T_j contradicting Lemma 22. We can thus indeed conclude that transaction T_j is based on Split.

To see that $j = 1$, recall that $\mu_1(\mathtt{S}_1) = \mu_{i-1}(\mathtt{S}_1)$ and the only possible conflict between T_i and T_j implies $\mu_i(\mathtt{C}) = \mu_j(\mathtt{C})$. From the latter we obtain $\mu_{i-1}(\mathtt{S}_1) = \mu_i(\mathtt{S}_1)$, due to $\mu_{i-1}(\mathtt{B}_{\text{next}}) = \mu_i(\mathtt{B})$ and function $f_{\text{future-solution-string}}$. From this it follows that $\mu_1(\mathtt{I}) = \mu_j(\mathtt{I})$ through $\mathtt{I} = (f_{\text{defines}} \circ f_{\text{is-non-empty}})(\mathtt{S}_1)$ in transaction template Split. That $j = 1$ then follows from Lemma 22. ◀

A.3.2 Final step

Finally, we show that if there is a multiversion split schedule with the properties of Lemma 20 that is a schedule-encoding for a sequence of dominoes $\mathbf{d}$, then this sequence $\mathbf{d}$ is also a solution to the respective PCP problem. The next Proposition thus finalized the proof for the if-direction of Theorem 10.

▶ **Proposition 27.** *Let $\mathcal{D}$ be a set of dominoes. Let s be a multiversion split schedule with the properties of Lemma 20 that is consistent with the transaction templates in Figure 7 for $\mathcal{D}$ and with some database $\boldsymbol{D}$. If s is a schedule-encoding of a sequence $\mathbf{d}$ of dominoes in $\mathcal{D}$, then $\mathbf{d}$ is a solution for the PCP problem on input $\mathcal{D}$.*

Proof. Let $a_1a_2 \ldots a_h$ and $b_1b_2 \ldots b_k$ be the two strings (with $a_i, b_i \in \Sigma$) obtained by reading from left to right, symbol by symbol, the values on the top, respectively, the bottom of dominoes $d_1, \ldots, d_r$. Let us say that $a_1a_2 \ldots a_h = \mathbf{a_1 a_2} \ldots \mathbf{a_r}$ and $b_1b_2 \ldots b_k = \mathbf{b_1 b_2} \ldots \mathbf{b_r}$. Notice that h and k are not necessarily equal to r as the top and bottom value of an individual domino can be of different length.

For convenience of notation, we introduce for every $i \in \{0, \ldots, h\}$ and $j \in \{1, \ldots, k\}$ the following notation:

$$\alpha_i := (f_{\text{append-}a_i} \circ f_{\text{append-}a_{i-1}} \circ \cdots \circ f_{\text{append-}a_1})(\mu_2(\mathtt{S}_0)).$$

$$\beta_j := (f_{\text{append-}b_j} \circ f_{\text{append-}b_{j-1}} \circ \cdots \circ f_{\text{append-}b_1})(\mu_2(\mathtt{S}_0)).$$

First, we show that

$$\alpha_h = \mu_2(\mathtt{S}_1) = \beta_k. \tag{1}$$

This result follows from the assumed structure of schedule s. More precisely, since an instantiation of First with an instantiation of $\text{Domino}_{\mathbf{d_1}}$ can only have conflicts on instantiations of $\mathtt{W}[\mathtt{B} : \mathtt{DominoSequence}]$, we have $\mu_2(\mathtt{B}) = \mu_3(\mathtt{B})$, from which it follows that $\mu_2(\mathtt{S}_1) = \mu_3(\mathtt{S}_1)$.

For every individual instantiation of $\text{Domino}_{\mathbf{d_i}}$ in s, we have that $f_{\text{append-}a_i^{\ell_a}} \circ \cdots \circ f_{\text{append-}a_i^1}(\mu_i(\mathtt{S}_t)) = \mu_i(\mathtt{S}_{t\mathbf{a_i}})$ and $f_{\text{append-}b_i^{\ell_b}} \circ \cdots \circ f_{\text{append-}b_i^1}(\mu_i(\mathtt{S}_b)) = \mu_i(\mathtt{S}_{b\mathbf{b_i}})$, with $\mathbf{a_i} = a_1a_2 \ldots a_{\ell_a}$ and $\mathbf{b_i} = b_1b_2 \ldots b_{\ell_b}$.

For transactions T_i, with $i \in \{3, \ldots, m+1\}$, (thus representing an instantiation of $\text{Domino}_{\mathbf{d_{i-2}}}$ which is followed in s by an instantiation of $\text{Domino}_{\mathbf{d_{i-1}}}$), the only possible conflict is between the instantiation of $\mathtt{W}[\mathtt{B_{next}} : \mathtt{DominoSequence}]$ (T_i) and of $\mathtt{W}[\mathtt{B} : \mathtt{DominoSequence}]$ in (T_{i+1}) – notice that this is indeed the only option due to Lemma 22 – thus with $\mu_i(\mathtt{B}_{next}) = \mu_{i+1}(\mathtt{B})$, implying $\mu_i(\mathtt{S}_{t\mathbf{a_i}}) = \mu_{i+1}(\mathtt{S}_t)$, $\mu_i(\mathtt{S}_{b\mathbf{b_i}}) = \mu_{i+1}(\mathtt{S}_b)$, and $\mu_i(\mathtt{S}_1) = \mu_{i+1}(\mathtt{S}_1)$.

Finally, transaction T_{m-1} (an instantiation of $\text{Domino}_{\mathbf{d_m}}$) can only conflict with transaction T_m (an instantiation of Last) on instantiations of $\mathtt{B}_{next}$ (in $\text{Domino}_{\mathbf{d_m}}$, and $\mathtt{B}$ (in Last), thus with $\mu_{m-1}(\mathtt{B}_{next}) = \mu_m(\mathtt{B})$, implying $\mu_{m-1}(\mathtt{S}_{t\mathbf{a_r}}) = \mu_m(\mathtt{S}_t) = \mu_m(\mathtt{S}_b) = \mu_{m-1}(\mathtt{S}_{b\mathbf{b_r}})$.

Combining the above equalities indeed proves Condition (1).

From Condition (1) we can now derive that,

$$\alpha_i = \mu_2(\mathtt{S}_1) = \beta_i, \text{ for every } i \in \{1, \ldots \min\{h, k\}\}, \tag{2}$$

by following an analogous approach. Indeed, in every instantiation of $\text{Domino}_{\mathbf{i}}$, there is a functional constraint for every application of the append function that requires its input to be the result of the detach function applied over its output, which indeed implies Condition (2).

To see that $k = h$, we observe that $k \neq h$ implies an application of the detach function over the instantiation of $\mathtt{S}_1$ (for which we already argued it has the same tuple assigned for every domino instantiation) for the shortest string, which contradicts with Condition (1) because such an application results in the same instantiation as $\mathtt{S}_e$, which can never equal the instantiation for $\mathtt{S}_1$.

The desired result that the individual symbols in the top and bottom reads of dominoes in sequence $\mathbf{d}$ are the same now follows from the functional constraint that every interpretation of a string sequence mapped via function f_{top} onto either the interpretation for $\mathtt{S}_1$ (representing symbol $1 \in \Sigma$) or $\mathtt{S}_0$ (representing symbol $0 \in \Sigma$). ◀

B Proofs for Section 5

B.1 Proof for Lemma 13

Before proving the correctness of Lemma 13, we first present two additional lemmas that will be used in the correctness proof.

▶ **Lemma 28.** *Let* $(\mathsf{Rels}, \mathsf{Funcs})$ *be a schema for which a disjoint partitioning of* Funcs *in pairs* $P = (f_1, g_1), (f_2, g_2), \ldots, (f_n, g_n)$ *exists such that* $dom(f_i) = range(g_i)$ *and* $dom(g_i) = range(f_i)$ *for every* $(f_i, g_i) \in P$ *and every schema graph* $SG(\mathsf{Rels}, \{h_1, h_2, \ldots, h_n\})$ *over the schema restricted to function names* $\{h_1, h_2, \ldots, h_n\}$ *with* $h_i \in (f_i, g_i)$ *is a multi-tree. Then:*

1. *there is no function name* $f \in \mathsf{Funcs}$ *with* $dom(f) = range(f)$*; and*
2. *for every path in* $SG(\mathsf{Rels}, \mathsf{Funcs})$*, say visiting the nodes* $R_1, R_2, \ldots, R_{m-1}, R_m$*, if* $R_1 = R_m$ *and* $R_1 \neq R_i$ *for every* $i \in [2, m-1]$*, then* $R_2 = R_{m-1}$ *and* (f, g) *is a pair in* P *with* f *the edge from* R_1 *to* R_2 *and* g *the edge from* R_{m-1} *to* R_m*.*

Proof. Towards a contradiction, assume *(1)* does not hold. That is, there is a function name f_i with $dom(f_i) = range(f_i) = R$ for some type R. Let g_i be the function name such that (f_i, g_i) is a pair in P. By definition, $dom(g_i) = range(g_i) = R$. But then we cannot pick a $h_i \in (f_i, g_i)$ such that the resulting schema graph is a multi-tree. Indeed, in both cases, there is a self-loop on R, leading to the desired contradiction.

For *(2)*, assume towards a contradiction that $R_2 \neq R_{m-1}$. Without loss of generality, we can assume that each node is visited only once in $R_2, \ldots, R_{m-1}$. Otherwise, $R_2, \ldots, R_{m-1}$ contains a loop that can be removed from this sequence without altering R_2 and R_{m-1}. Since

$R_1, R_2, \ldots, R_{m-1}, R_m$ is a path in $SG(\mathsf{Rels}, \mathsf{Funcs})$, there is a sequence of function names $F = e_1 \cdots e_{m-1}$ such that each e_i is an edge from R_i to R_{i+1} in $SG(\mathsf{Rels}, \mathsf{Funcs})$, implying $dom(e_i) = R_i$ and $range(e_i) = R_{i+1}$. By assumption that each type R_i occurs only once in $R_2, \ldots, R_{m-1}$ (notice that, for $i = 1$, this follows from Condition (2) of the lemma) and type $R_1 = R_m$ does not appear in $R_2, \ldots, R_{m-1}$, there is no pair of function names e_i and e_j in F with $i \neq j$, $dom(e_i) = range(e_j)$ and $range(e_i) = dom(e_j)$. Therefore, at most one function name of each pair in P appears in F. But then we can choose $h_i = e_i$ for each such pair in P, with e_i the function name appearing in F. Since F describes a cycle in $SG(\mathsf{Rels}, \mathsf{Funcs})$, the resulting schema graph restricted to these h_i cannot be a multi-tree, as it contains a cycle.

It remains to argue that if f is the edge from R_1 to R_2 and g is the edge from R_{m-1} to R_m on this path, then (f, g) is a pair in P. To this end, note that $dom(f) = range(g)$ and $range(f) = dom(g)$, as $R_1 = R_m$ and $R_2 = R_{m-1}$. If (f, g) is not a pair in P, then there are two pairs (f, f') and (g, g') in P with $dom(f) = dom(g') = range(f') = range(g) = R_1$ and $range(f) = range(g') = dom(f') = dom(g) = R_2$. Then we can choose f in (f, f') and g in (g, g'). Since the resulting schema graph cannot be a multi-tree, as there is a cycle between R_1 and R_2, this choice leads to a contradiction. ◀

▶ **Lemma 29.** *Let D be a sequence of potentially conflicting quadruples over $\mathcal{P} \in$ **MTBTemp**. Then*

1. $\mathtt{X} \rightsquigarrow_\tau \mathtt{Y}$ *iff* $\mathtt{Y} \rightsquigarrow_\tau \mathtt{X}$ *for every pair of variables* $\mathtt{X}$ *and* $\mathtt{Y}$ *occurring in a template τ; and*
2. $\mathtt{X} \rightsquigarrow_D \mathtt{Y}$ *iff* $\mathtt{Y} \rightsquigarrow_D \mathtt{X}$ *for every pair of variables* $\mathtt{X}$ *and* $\mathtt{Y}$ *occurring in D.*

Proof. *(1)* We argue by induction on the definition of $\mathtt{X} \rightsquigarrow_\tau \mathtt{Y}$ that $\mathtt{X} \rightsquigarrow_\tau \mathtt{Y}$ implies $\mathtt{Y} \rightsquigarrow_\tau \mathtt{X}$. The other direction is analogous. The base case is immediate, as $\mathtt{X} = \mathtt{Y}$ implies $\mathtt{Y} \rightsquigarrow_\tau \mathtt{X}$ by definition. For the inductive case, assume a variable $\mathtt{Z}$ such that $\mathtt{Y} = f(\mathtt{Z})$ is a constraint in τ and $\mathtt{X} \rightsquigarrow_\tau \mathtt{Z}$. By the induction hypothesis, $\mathtt{Z} \overset{F}{\rightsquigarrow}_\tau \mathtt{X}$ for some sequence of function names F. Since $\mathcal{P} \in \mathbf{MTBTemp}$, there is a constraint $\mathtt{Z} = f'(\mathtt{Y})$ in τ as well. It follows that $\mathtt{Y} \overset{F'}{\rightsquigarrow}_\tau \mathtt{X}$ with $F' = f' \cdot F$.

(2) We argue by induction on the definition of $\mathtt{X} \rightsquigarrow_D \mathtt{Y}$ that $\mathtt{X} \rightsquigarrow_D \mathtt{Y}$ implies $\mathtt{Y} \rightsquigarrow_D \mathtt{X}$. The other direction is again analogous. The first base case is now immediate, as we already argued that $\mathtt{X} \rightsquigarrow_\tau \mathtt{Y}$ implies $\mathtt{Y} \rightsquigarrow_\tau \mathtt{X}$. For the second base case, assume $\mathtt{X} \overset{\varepsilon}{\rightsquigarrow}_D \mathtt{Y}$ and $(\tau_i, o_i, p_j, \tau_j)$ is a potentially conflicting quadruple in D with $var(o_i) = \mathtt{X}$ and $var(p_j) = \mathtt{Y}$ (the case for $(\tau_j, o_j, p_i, \tau_i)$ is analogous). $\mathtt{Y} \overset{\varepsilon}{\rightsquigarrow}_D \mathtt{X}$ then follows by definition. For the inductive case, let $\mathtt{Z}$ be a variable such that $\mathtt{X} \overset{F_1}{\rightsquigarrow}_D \mathtt{Z}$ and $\mathtt{Z} \overset{F_2}{\rightsquigarrow}_D \mathtt{Y}$. Then by induction hypothesis $\mathtt{Z} \overset{F_1'}{\rightsquigarrow}_D \mathtt{X}$ and $\mathtt{Y} \overset{F_2'}{\rightsquigarrow}_D \mathtt{Z}$ for some sequence of function names F_1' and F_2'. By definition, $\mathtt{Y} \overset{F'}{\rightsquigarrow}_D \mathtt{X}$ with $F' = F_2' \cdot F_1'$. ◀

▶ **Lemma 13.** *Let D be a sequence of potentially conflicting quadruples over $\mathcal{P} \in$ **MTBTemp**. Then $\mathtt{X} \approx_D \mathtt{Y}$ implies $\mathtt{X} \rightsquigarrow_D \mathtt{Y}$ and $\mathtt{Y} \rightsquigarrow_D \mathtt{X}$. Furthermore, if $type(\mathtt{X}) = type(\mathtt{Y})$ then $\bar{\mu}(\mathtt{X}) = \bar{\mu}(\mathtt{Y})$ for every variable mapping $\bar{\mu}$ for D that is admissible for some database **D**.*

Proof. (1) Assuming $\mathtt{X} \approx_D \mathtt{Y}$, we first show by induction on the definition of connectedness that $\mathtt{X} \rightsquigarrow_D \mathtt{Y}$. By Lemma 29, $\mathtt{Y} \rightsquigarrow_D \mathtt{X}$ then follows. For the base case, both $\mathtt{X} \rightsquigarrow_D \mathtt{Y}$ and $\mathtt{Y} \rightsquigarrow_D \mathtt{X}$ imply $\mathtt{X} \rightsquigarrow_D \mathtt{Y}$, where the former is immediate and the latter is by Lemma 29. For the inductive case, let $\mathtt{Z}$ be a variable with $\mathtt{X} \approx_D \mathtt{Z}$ and either $\mathtt{Z} \rightsquigarrow_D \mathtt{Y}$ or $\mathtt{Y} \rightsquigarrow_D \mathtt{Z}$. Again, $\mathtt{Z} \overset{F_2}{\rightsquigarrow}_D \mathtt{Y}$ for some sequence of function names F_2 is implied in both cases. By induction hypothesis, $\mathtt{X} \overset{F_1}{\rightsquigarrow}_D \mathtt{Z}$ for some sequence of function names F_1. As a result, $\mathtt{X} \overset{F}{\rightsquigarrow}_D \mathtt{Y}$ with $F = F_1 \cdot F_2$.

(2) Next, let $\mathtt{X}$ and $\mathtt{Y}$ be two variables occurring in $\mathrm{Trans}(D)$ with $\mathtt{X} \approx_D \mathtt{Y}$ and $\mathrm{type}(\mathtt{X}) = \mathrm{type}(\mathtt{Y})$ and let $\bar{\mu}$ be a variable mapping for D that is admissible for a database $\mathbf{D}$. We prove that $\bar{\mu}(\mathtt{X}) = \bar{\mu}(\mathtt{Y})$.

We already argued that $\mathtt{X} \approx\!\approx_D \mathtt{Y}$ implies $\mathtt{X} \leadsto_D \mathtt{Y}$. By definition of $\mathtt{X} \leadsto_D \mathtt{Y}$, there is a sequence of variables $\mathtt{X}_1, \mathtt{X}_2 \ldots, \mathtt{X}_n$ with $\mathtt{X}_1 = \mathtt{X}$ and $\mathtt{X}_n = \mathtt{Y}$ such that for each pair of adjacent variables $\mathtt{X}_i$ and $\mathtt{X}_{i+1}$:

(†) $\mathtt{X}_i$ and $\mathtt{X}_{i+1}$ both occur in the same template $\tau \in \text{Trans}(D)$ and $\mathtt{X}_{i+1} = f(\mathtt{X}_i) \in \Gamma(\tau)$ for some function name f; or

(‡) $\text{type}(\mathtt{X}_i) = \text{type}(\mathtt{X}_{i+1})$ and there is a potentially conflicting quadruple $(\tau_j, o_j, p_k, \tau_k)$ in D with either $var(o_j) = \mathtt{X}_i$ and $var(p_k) = \mathtt{X}_{i+1}$ or $var(p_k) = \mathtt{X}_i$ and $var(o_j) = \mathtt{X}_{i+1}$.

In the remainder of this proof, we show that for each pair of variables $\mathtt{X}_i$ and $\mathtt{X}_j$ in this sequence with $\text{type}(\mathtt{X}_i) = \text{type}(\mathtt{X}_j)$ that $\bar{\mu}(\mathtt{X}_i) = \bar{\mu}(\mathtt{X}_j)$. The desired $\bar{\mu}(\mathtt{X}) = \bar{\mu}(\mathtt{Y})$ then follows immediately as $\mathtt{X} = \mathtt{X}_1$ and $\mathtt{Y} = \mathtt{X}_n$. Note that it suffices to show this property only for pairs of variables $\mathtt{X}_i$ and $\mathtt{X}_j$ for which no variable $\mathtt{X}_k$ exists with $i < k < j$ and $\text{type}(\mathtt{X}_i) = \text{type}(\mathtt{X}_j) = \text{type}(\mathtt{X}_k)$. Indeed, if such an $\mathtt{X}_k$ exists, we can recursively argue that $\bar{\mu}(\mathtt{X}_i) = \bar{\mu}(\mathtt{X}_k)$ and $\bar{\mu}(\mathtt{X}_k) = \bar{\mu}(\mathtt{X}_j)$. The argument is by induction on the number of variables between $\mathtt{X}_i$ and $\mathtt{X}_j$.

If $j = i + 1$ *(base case)*, then (‡) applies to $\mathtt{X}_i$ and $\mathtt{X}_j$. Indeed, if (†) would apply instead, then there would be a function name f with $dom(f) = \text{type}(\mathtt{X}_i) = \text{type}(\mathtt{X}_j) = range(f)$, contradicting Condition *(1)* of Lemma 28. By definition of $\bar{\mu}$, we have $\bar{\mu}(\mathtt{X}_i) = \bar{\mu}(\mathtt{X}_j)$.

Next, let $i + 1 < j$ *(inductive case)*, and assume that $\bar{\mu}(\mathtt{X}_k) = \bar{\mu}(\mathtt{X}_\ell)$ for all $\mathtt{X}_k$ and $\mathtt{X}_\ell$ with $i < k \leq \ell < j$ and $\text{type}(\mathtt{X}_k) = \text{type}(\mathtt{X}_\ell)$ *(induction hypothesis)*. From this sequence $\mathtt{X}_i, \ldots \mathtt{X}_j$, we derive a sequence of function names $F = f_1 \cdots f_{m-1}$, where each function name f_i is based on an application of (†) on adjacent variables (notice that applications of (‡) do not result in a function name being added to F). By assumption on the types of variables $\mathtt{X}_k$ with $i < k < j$, we have in particular $\text{type}(\mathtt{X}_{i+1}) \neq \text{type}(\mathtt{X}_i)$ and $\text{type}(\mathtt{X}_{j-1}) \neq \text{type}(\mathtt{X}_j)$. This implies that (†) is applicable for $\mathtt{X}_i$ and $\mathtt{X}_{i+1}$ (respectively $\mathtt{X}_{j-1}$ and $\mathtt{X}_j$). Furthermore, $\mathtt{X}_i$ and $\mathtt{X}_{i+1}$ appear in the same template, say τ_i (respectively τ_j for $\mathtt{X}_{j-1}$ and $\mathtt{X}_j$), and $\mathtt{X}_{i+1} = f_1(\mathtt{X}_i) \in \Gamma(\tau_i)$ (respectively $\mathtt{X}_j = f_{m-1}(\mathtt{X}_{j-1}) \in \Gamma(\tau_j)$). By construction, F then describes a path in $SG(\mathsf{Rels}, \mathsf{Funcs})$ visiting the nodes $R_1, R_2, \ldots, R_{m-1}, R_m$ with $\text{type}(\mathtt{X}_i) = R_1$, $\text{type}(\mathtt{X}_{i+1}) = R_2$, $\text{type}(\mathtt{X}_{j-1}) = R_{m-1}$ and $\text{type}(\mathtt{X}_j) = R_m$. Since this path satisfies Condition 2 in Lemma 28, it follows that $\text{type}(\mathtt{X}_{i+1}) = \text{type}(\mathtt{X}_{j-1})$ and (f_1, f_{m-1}) is a pair in the pairwise partitioning of Funcs witnessing $\mathcal{P} \in \mathbf{MTBTemp}$. By definition of $\mathbf{MTBTemp}$, $\mathtt{X}_{i+1} = f_1(\mathtt{X}_i) \in \Gamma(\tau_i)$ then implies $\mathtt{X}_i = f_{m-1}(\mathtt{X}_{i+1}) \in \Gamma(\tau_i)$. According to the induction hypothesis, $\bar{\mu}(\mathtt{X}_{i+1}) = \bar{\mu}(\mathtt{X}_{j-1})$. Since $\bar{\mu}$ is admissible for $\mathbf{D}$, we conclude that $\bar{\mu}(\mathtt{X}_i) = f^{\mathbf{D}}_{m-1}(\bar{\mu}(\mathtt{X}_{i+1})) = f^{\mathbf{D}}_{m-1}(\bar{\mu}(\mathtt{X}_{j-1})) = \bar{\mu}(\mathtt{X}_j)$. ◀

B.2 Proof for Lemma 14

▶ **Lemma 14.** *For a multiversion split schedule s based on a sequence of conflicting quadruples C over a set of transactions $\mathcal{T}$ consistent with a $\mathcal{P} \in$ **MTBTemp** and a database **D**, let $\bar{\mu}$ be the variable mapping for a sequence of potentially conflicting quadruples D over $\mathcal{P}$ with $\bar{\mu}(D) = C$. Then, a set $\mathcal{S}$ of type mappings over disjoint ranges and a function $\varphi_{\mathcal{S}} : \mathbf{Var} \to \mathcal{S}$ exist with:*

- *$\bar{\mu}(\mathtt{X}) = c(type(\mathtt{X}))$ for every variable $\mathtt{X}$, with $c = \varphi_{\mathcal{S}}(\mathtt{X})$;*
- *$\varphi_{\mathcal{S}}(\mathtt{X}) = \varphi_{\mathcal{S}}(\mathtt{Y})$ whenever $\mathtt{X} \approx\!\approx_D \mathtt{Y}$; and,*
- *$\varphi_{\mathcal{S}}(\mathtt{X}) \neq \varphi_{\mathcal{S}}(\mathtt{Y})$ for every constraint $\mathtt{X} \neq \mathtt{Y}$ occurring in a template $\tau \in Trans(D)$.*

Proof. To aid the construction of $\mathcal{S}$ and $\varphi_{\mathcal{S}}$, we first define a coloring function λ that assigns a color to each tuple occurring in the schedule s such that the following holds: for every pair of tuples $\mathtt{t}$ and $\mathtt{v}$ occurring in s:

- connected tuples are mapped to the same color: if $\bar{\mu}(\mathtt{X}) = \mathtt{t}$, $\bar{\mu}(\mathtt{Y}) = \mathtt{v}$ and $\mathtt{X} \approx_D \mathtt{Y}$ for some variables $\mathtt{X}, \mathtt{Y}$ occurring in Trans(D), then $\lambda(\mathtt{t}) = \lambda(\mathtt{v})$; and
- different tuples of the same type are mapped to different colors: if type$(\mathtt{t})$ = type$(\mathtt{v})$ and $\mathtt{t} \neq \mathtt{v}$, then $\lambda(\mathtt{t}) \neq \lambda(\mathtt{v})$.

Note that we can always construct such a function λ as by Lemma 13, it cannot be the case that type$(\mathtt{t})$ = type$(\mathtt{v})$, $\mathtt{t} \neq \mathtt{v}$ and there is a pair of variables $\mathtt{X}, \mathtt{Y}$ with $\bar{\mu}(\mathtt{X}) = \mathtt{t}$, $\bar{\mu}(\mathtt{Y}) = \mathtt{v}$, and $\mathtt{X} \approx_D \mathtt{Y}$.

For $\alpha \in \mathit{range}(\lambda)$, define the type mapping c_α as follows: for every type $R \in \mathsf{Rels}$:

$$c_\alpha(R) = \begin{cases} \mathtt{t} & \text{if } \lambda(\mathtt{t}) = \alpha \text{ and type}(\mathtt{t}) = R, \\ \mathtt{v}_{c,R} & \text{otherwise,} \end{cases}$$

where $\mathtt{v}_{c,R}$ is an arbitrary tuple of type R not occurring in s or any other type mapping c_β for $\beta \in \mathit{range}(\lambda)$. Define $\mathcal{S} = \{c_\alpha \mid \alpha \in \mathit{range}(\lambda)\}$. By construction, every type mapping in $\mathcal{S}$ is well defined and all type mappings are over disjoint ranges. Furthermore, $c_\alpha \neq c_\beta$ whenever $\alpha \neq \beta$.

We now construct $\varphi_\mathcal{S}$ as follows: $\varphi_\mathcal{S}(\mathtt{X}) = c_\alpha$ with $\alpha = \lambda(\bar{\mu}(\mathtt{X}))$ for every variable $\mathtt{X}$ occurring in Trans(D). It remains to argue that $\varphi_\mathcal{S}$ indeed satisfies all properties stated in Lemma 14. By construction of $\mathcal{S}$ and $\varphi_\mathcal{S}$, we have $\bar{\mu}(\mathtt{X}) = c(\text{type}(\mathtt{X}))$ for every variable $\mathtt{X}$, with $c = \varphi_\mathcal{S}(\mathtt{X})$. Towards the second property, notice that $\mathtt{X} \approx_D \mathtt{Y}$ implies $\varphi_\mathcal{S}(\mathtt{X}) = c_{\lambda(\bar{\mu}(\mathtt{X}))} = c_{\lambda(\bar{\mu}(\mathtt{Y}))} = \varphi_\mathcal{S}(\mathtt{Y})$ by definition of λ and $\varphi_\mathcal{S}$. For the last property, assume $\mathtt{X} \neq \mathtt{Y}$ occurs in a template $\tau \in$ Trans(D) and type$(\mathtt{X})$ = type$(\mathtt{Y})$. Since $\bar{\mu}$ is admissible for database $\mathbf{D}$, $\bar{\mu}(\mathtt{X}) \neq \bar{\mu}(\mathtt{Y})$. Then, by definition of λ and $\varphi_\mathcal{S}$, we have $\varphi_\mathcal{S}(\mathtt{X}) = c_{\lambda(\bar{\mu}(\mathtt{X}))} \neq c_{\lambda(\bar{\mu}(\mathtt{Y}))} = \varphi_\mathcal{S}(\mathtt{Y})$. ◀

B.3 Proof for Lemma 15

▶ **Lemma 30.** *Let D be a sequence of potentially conflicting quadruples. If $\mathtt{X} \approx_D \mathtt{Y}$ and $\mathtt{Y} \approx_D \mathtt{Z}$ then $\mathtt{X} \approx_D \mathtt{Z}$ for every triple of variables $\mathtt{X}, \mathtt{Y}, \mathtt{Z}$ occurring in Trans(D).*

Proof. According to Lemma 13, $\mathtt{X} \approx_D \mathtt{Y}$ and $\mathtt{Y} \approx_D \mathtt{Z}$ imply respectively $\mathtt{X} \rightsquigarrow_D \mathtt{Y}$ and $\mathtt{Y} \rightsquigarrow_D \mathtt{Z}$. By definition, $\mathtt{X} \rightsquigarrow_D \mathtt{Z}$ and hence $\mathtt{X} \approx_D \mathtt{Z}$. ◀

▶ **Lemma 15.** *Let $\mathcal{P} \in$ **MTBTemp** and let $\mathcal{S} = \{c_1, c_2, c_3, c_4\}$ be a set consisting of four type mappings with disjoint ranges. Then, $\mathcal{P}$ is not robust against RC iff there is a sequence of quintuples $E = (\tau_1, o_1, c_{o_1}, p_1, c_{p_1}), \ldots, (\tau_m, o_m, c_{o_m}, p_m, c_{p_m})$ with $m \geq 2$ such that for each quintuple $(\tau_i, o_i, c_{o_i}, p_i, c_{p_i})$ in E:*

1. *o_i and p_i are operations in τ_i, and $c_{o_i}, c_{p_i} \in \mathcal{S}$;*
2. *$\mathtt{X}_i \not\approx_{\tau_i} \mathtt{Y}_i$ for each constraint $\mathtt{X}_i \neq \mathtt{Y}_i$ in τ_i;*
3. *$c_{o_i} = c_{p_i}$ if $o_i \approx_{\tau_i} p_i$;*
4. *$c_{o_i} \neq c_{p_i}$ if there is a constraint $\mathtt{X}_i \neq \mathtt{Y}_i$ in τ_i with $\mathtt{X}_i \approx_{\tau_i} var(o_i)$ and $\mathtt{Y}_i \approx_{\tau_i} var(p_i)$;*
5. *if $i \neq 1$ and $c_{q_i} = c_{q_1}$ for some $q_i \in \{o_i, p_i\}$ and $q_1 \in \{o_1, p_1\}$, then there is no operation o'_i in τ_i potentially ww-conflicting with an operation o'_1 in $\mathsf{prefix}_{o_1}(\tau_1)$ with $var(o'_i) \approx_{\tau_i} var(q_i)$ and $var(o'_1) \approx_{\tau_1} var(q_1)$.*

Furthermore, for each pair of adjacent quintuples $(\tau_i, o_i, c_{o_i}, p_i, c_{p_i})$ and $(\tau_j, o_j, c_{o_j}, p_j, c_{p_j})$ in E with $j = i + 1$, or $i = m$ and $j = 1$:

6. *o_i is potentially conflicting with p_j and $c_{o_i} = c_{p_j}$;*
7. *if $i = 1$ and $j = 2$, then o_1 is potentially rw-conflicting with p_2; and*
8. *if $i = m$ and $j = 1$, then $o_1 <_{\tau_1} p_1$ or o_m is potentially rw-conflicting with p_1.*

Proof. *(if)* Let $D = (\tau_1, o_1, p_2, \tau_2), \ldots, (\tau_m, o_m, p_1, \tau_1)$ be the sequence of potentially conflicting quadruples derived from E. Notice in particular that D is indeed a sequence of potentially conflicting quadruples by (1) and (6). We construct a variable mapping $\bar{\mu}$ for D admissible for a database $\mathbf{D}$ such that the sequence of conflicting quadruples $C = \bar{\mu}(D)$ satisfies the conditions in Definition 6, thereby proving that $\mathcal{P}$ is not robust against RC.

Let $\varphi_{\mathcal{S}} : \mathbf{Var} \to \mathcal{S}$ be the (partial) function assigning a type mapping in $\mathcal{S}$ to each variable occurring in an operation in E:

$$\varphi_{\mathcal{S}}(\mathtt{X}) = \begin{cases} c_{o_i} & \text{if } \mathit{var}(o_i) = \mathtt{X} \text{ for some } (\tau_i, o_i, c_{o_i}, p_i, c_{p_i}) \in E, \\ c_{p_i} & \text{if } \mathit{var}(p_i) = \mathtt{X} \text{ for some } (\tau_i, o_i, c_{o_i}, p_i, c_{p_i}) \in E. \end{cases}$$

This function $\varphi_{\mathcal{S}}$ is well defined: if there is a $(\tau_i, o_i, c_{o_i}, p_i, c_{p_i}) \in E$ with $\mathit{var}(o_i) = \mathit{var}(p_i) = \mathtt{X}$, then $o_i \approx\!\!\approx_{\tau_i} p_i$ and hence $c_{o_i} = c_{p_i}$ by (3). Recall that we assume that templates in E are variable-disjoint. We argue that $\varphi_{\mathcal{S}}(\mathtt{X}) = \varphi_{\mathcal{S}}(\mathtt{Y})$ if $\mathtt{X} \approx\!\!\approx_D \mathtt{Y}$ for each pair of variables $\mathtt{X}$ and $\mathtt{Y}$ for which $\varphi_{\mathcal{S}}$ is defined. From Lemma 13, it follows that $\mathtt{X} \rightsquigarrow_D \mathtt{Y}$ whenever $\mathtt{X} \approx\!\!\approx_D \mathtt{Y}$. Let τ_i and τ_j be the template in which respectively $\mathtt{X}$ and $\mathtt{Y}$ occur. The argument is now by induction on the definition of $\mathtt{X} \rightsquigarrow_D \mathtt{Y}$:

- If $i = j$ and $\mathtt{X} \rightsquigarrow_{\tau_i} \mathtt{Y}$, then $\varphi_{\mathcal{S}}(\mathtt{X}) = \varphi_{\mathcal{S}}(\mathtt{Y})$ is immediate by (3);
- If $(\tau_i, o_i, p_j, \tau_j) \in D$ with $\mathit{var}(o_i) = \mathtt{X}$ and $\mathit{var}(p_j) = \mathtt{Y}$ (respectively $(\tau_j, o_j, p_i, \tau_i) \in D$ with $\mathit{var}(o_j) = \mathtt{Y}$ and $\mathit{var}(p_i) = \mathtt{X}$), then $\varphi_{\mathcal{S}}(\mathtt{X}) = \varphi_{\mathcal{S}}(\mathtt{Y})$ is immediate by (6);
- Otherwise, if $\mathtt{X} \rightsquigarrow_D \mathtt{Z}$ and $\mathtt{Z} \rightsquigarrow_D \mathtt{Y}$ for some variable $\mathtt{Z}$, then by induction $\varphi_{\mathcal{S}}(\mathtt{X}) = \varphi_{\mathcal{S}}(\mathtt{Z}) = \varphi_{\mathcal{S}}(\mathtt{Y})$.

By Lemma 30, $\approx\!\!\approx_D$ is an equivalence relation. For $\mathtt{X}$ occurring in $\mathrm{Trans}(D)$, denote by $[\mathtt{X}]$ the equivalence class of $\mathtt{X}$. Let $\mathcal{S}'$ be obtained by extending $\mathcal{S}$ with a type mapping $c_{[\mathtt{X}]}$ for each equivalence class where no variable $\mathtt{Y} \in [\mathtt{X}]$ is defined in $\varphi_{\mathcal{S}}$. Furthermore, each of the $c_{[\mathtt{X}]}$ are picked such that all type mappings in $\mathcal{S}'$ have disjoint ranges.

Next, we extend $\varphi_{\mathcal{S}}$ to a function $\varphi_{\mathcal{S}'} : \mathbf{Var} \to \mathcal{S}'$ assigning a type mapping to each variable $\mathtt{X}$ occurring in $\mathrm{Trans}(D)$ as follows:

$$\varphi_{\mathcal{S}'}(\mathtt{X}) = \begin{cases} \varphi_{\mathcal{S}}(\mathtt{X}) & \text{if } \varphi_{\mathcal{S}} \text{ is defined for } \mathtt{X}, \\ \varphi_{\mathcal{S}}(\mathtt{Y}) & \text{if } \varphi_{\mathcal{S}} \text{ is defined for } \mathtt{Y} \text{ but not for } \mathtt{X} \text{ and } \mathtt{X} \approx\!\!\approx_D \mathtt{Y}, \\ c_{[\mathtt{X}]} & \text{otherwise.} \end{cases}$$

Notice, furthermore, that in the second case $\mathtt{X}$ might be connected in D to multiple variables for which $\varphi_{\mathcal{S}}$ is defined, say $\mathtt{Y}_1$ and $\mathtt{Y}_2$. Then, by Lemma 30, $\mathtt{Y}_1 \approx\!\!\approx_D \mathtt{Y}_2$ and hence $\varphi_{\mathcal{S}}(\mathtt{Y}_1) = \varphi_{\mathcal{S}}(\mathtt{Y}_2)$. We therefore conclude that $\varphi_{\mathcal{S}'}(\mathtt{X})$ is well defined. We argue that $\varphi_{\mathcal{S}'}(\mathtt{X}) = \varphi_{\mathcal{S}'}(\mathtt{Y})$ if $\mathtt{X} \approx\!\!\approx_D \mathtt{Y}$ for each pair of variables $\mathtt{X}$ and $\mathtt{Y}$. If $\varphi_{\mathcal{S}}$ is defined for both $\mathtt{X}$ and $\mathtt{Y}$, then the result is immediate by $\varphi_{\mathcal{S}}(\mathtt{X}) = \varphi_{\mathcal{S}}(\mathtt{Y})$. If $\varphi_{\mathcal{S}}$ is defined for one of these two variables, say $\mathtt{X}$, then $\varphi_{\mathcal{S}'}(\mathtt{X}) = \varphi_{\mathcal{S}}(\mathtt{X}) = \varphi_{\mathcal{S}'}(\mathtt{Y})$ by construction of $\varphi_{\mathcal{S}'}$. If $\varphi_{\mathcal{S}}$ is not defined for both $\mathtt{X}$ and $\mathtt{Y}$, then either there exists a variable $\mathtt{Z}$ for which $\varphi_{\mathcal{S}}$ is defined and $\mathtt{Z}$ is connected in D to one of these two variables, say $\mathtt{X}$, or no such variable $\mathtt{Z}$ exists. In the former case, $\mathtt{Z} \approx\!\!\approx_D \mathtt{Y}$ follows from Lemma 30, implying $\varphi_{\mathcal{S}'}(\mathtt{X}) = \varphi_{\mathcal{S}}(\mathtt{Z}) = \varphi_{\mathcal{S}'}(\mathtt{Y})$. In the latter case, $\varphi_{\mathcal{S}'}(\mathtt{X}) = c_{[\mathtt{X}]} = c_{[\mathtt{Y}]} = \varphi_{\mathcal{S}'}(\mathtt{Y})$ by construction of $\varphi_{\mathcal{S}'}$.

We now define the variable mapping $\bar{\mu}$ from $\varphi_{\mathcal{S}'}$ as $\bar{\mu}(\mathtt{X}) = c(\mathrm{type}(\mathtt{X}))$ for each variable $\mathtt{X}$, where $c = \varphi_{\mathcal{S}}(\mathtt{X})$. Next, we construct the database $\mathbf{D}$. For each template τ_i and corresponding variable mapping μ_i in $\bar{\mu}$, we add all tuples in $\mu_i(\tau_i)$ to the database $\mathbf{D}$. Furthermore, for each constraint $\mathtt{X} = f(\mathtt{Y})$ in $\Gamma(\tau_i)$, we have $f^{\mathbf{D}}(\mu_i(\mathtt{X})) = \mu_i(\mathtt{Y})$ in $\mathbf{D}$. This is well defined for each function $f^{\mathbf{D}}$. Towards a contradiction, assume we have $f^{\mathbf{D}}(\bar{\mu}(\mathtt{X}_i)) = \bar{\mu}(\mathtt{Y}_i)$ witnessed by a template τ_i and $f^{\mathbf{D}}(\bar{\mu}(\mathtt{X}_j)) = \bar{\mu}(\mathtt{Y}_j)$ witnessed by a template τ_j, where $\bar{\mu}(\mathtt{X}_i) = \bar{\mu}(\mathtt{X}_j)$, but $\bar{\mu}(\mathtt{Y}_i) \neq \bar{\mu}(\mathtt{Y}_j)$. Since $\mathtt{X}_i \approx\!\!\approx_D \mathtt{Y}_i$, $\mathtt{X}_j \approx\!\!\approx_D \mathtt{Y}_j$ and $\bar{\mu}(\mathtt{X}_i) = \bar{\mu}(\mathtt{X}_j)$, we have $\varphi_{\mathcal{S}'}(\mathtt{Y}_i) = \varphi_{\mathcal{S}'}(\mathtt{X}_i) = \varphi_{\mathcal{S}'}(\mathtt{X}_j) = \varphi_{\mathcal{S}'}(\mathtt{Y}_j)$. Then, $\bar{\mu}(\mathtt{Y}_i) = \bar{\mu}(\mathtt{Y}_j)$, leading to a contradiction.

In order to argue that $\bar{\mu}$ is indeed admissible for $\mathbf{D}$, it remains to show that for each constraint $\mathtt{X} \neq \mathtt{Y}$ in a template τ_i, we have $\bar{\mu}(\mathtt{X}) \neq \bar{\mu}(\mathtt{Y})$. Again towards a contradiction, assume $\bar{\mu}(\mathtt{X}) = \bar{\mu}(\mathtt{Y})$, and let $(\tau_i, o_i, c_{o_i}, p_i, c_{p_i})$ be the corresponding quintuple in E. By definition of $\bar{\mu}$, we have $\varphi_{\mathcal{S}'}(\mathtt{X}) = \varphi_{\mathcal{S}'}(\mathtt{Y})$. It follows from $\varphi_{\mathcal{S}'}$ that either $\mathtt{X} \approx_{\tau_i} \mathtt{Y}$; or $\mathtt{X} \approx_{\tau_i} var(o_i)$ (respectively $\mathtt{Y} \approx_{\tau_i} var(o_i)$), $\mathtt{Y} \approx_{\tau_i} var(p_i)$ (respectively $\mathtt{X} \approx_{\tau_i} var(p_i)$) and $c_{o_i} = c_{p_i}$. However, the former is contradicted by (2) and the latter by (4). We therefore conclude that $\bar{\mu}$ is admissible for $\mathbf{D}$, as it satisfies all constraints.

It remains to argue that the sequence of conflicting quadruples $C = \bar{\mu}(D)$ satisfies all conditions stated in Definition 6. The second and third condition are immediate by respectively (8) and (7). Towards a contradiction, assume the first condition holds. Then, there is an operation b'_1 in $\mathsf{prefix}_{\bar{\mu}o_1}(\bar{\mu}(\tau_1))$ ww-conflicting with an operation b'_i in a transaction $\bar{\mu}\tau_i$. Let $b'_1 = \bar{\mu}(o'_1)$ with $var(o'_1) = \mathtt{X}_1$ and $b'_i = \bar{\mu}(o'_i)$ with $var(o'_i) = \mathtt{X}$, and let $(\tau_1, o_1, c_{o_1}, p_1, c_{p_1})$ and $(\tau_i, o_i, c_{o_i}, p_i, c_{p_i})$ be the corresponding quintuples in E. Note that o'_1 is potentially ww-conflicting with o'_i and $\bar{\mu}(\mathtt{X}_1) = \bar{\mu}(\mathtt{X}_i)$. Then, $\varphi_{\mathcal{A}'}(\mathtt{X}_1) = \varphi_{\mathcal{A}'}(\mathtt{X}_i)$. By construction of $\varphi_{\mathcal{A}'}$, this can only hold if $\mathtt{X}_1 \approx_{\tau_1} var(q_1)$, $\mathtt{X}_i \approx_{\tau_i} var(q_i)$ and $c_{q_1} = c_{q_i}$ for some $q_1 \in o_1, p_1$ and $q_i \in o_i, p_i$, thereby contradicting (5).

(only if) If $\mathcal{P}$ is not robust against RC, then there exists a multiversion split schedule s based on a sequence of conflicting quadruples C over a set of transactions $\mathcal{T}$ consistent with $\mathcal{P}$ and a database $\mathbf{D}$. Let $\bar{\mu}$ be the variable mapping for a sequence of potentially conflicting quadruples $D = (\tau_1, o_1, p_1, \tau_2), \ldots, (\tau_m, o_m, p_1, \tau_1)$ over $\mathcal{P}$ with $\bar{\mu}(C) = D$, and let $\mathcal{S}$ and $\varphi_{\mathcal{S}}$ be as in Lemma 14.

From this sequence D and function $\varphi_{\mathcal{S}}$, we derive the sequence of quintuples $E = (\tau_1, o_1, \varphi_{\mathcal{S}}(var(o_1)), p_1, \varphi_{\mathcal{S}}(var(p_1)))), \ldots, (\tau_m, o_m, \varphi_{\mathcal{S}}(var(o_m)), p_m, \varphi_{\mathcal{S}}(var(p_m)))$. Let $\varphi'_{\mathcal{S}} = \{c_1, c_2, c_3, c_4\}$ be a set consisting of four type mappings with disjoint ranges. We adapt each quintuple in E in order, thereby creating a sequence E' satisfying the properties stated in Lemma 15. First, we add $(\tau_1, o_1, c_1, p_1, c_k)$ to E', where $c_k = c_1$ if $\varphi_{\mathcal{S}}(var(o_1)) = \varphi_{\mathcal{S}}(var(p_1))$, and $c_k = c_2$ otherwise. For each of the remaining quintuples in E, let $(\tau_{i-1}, o_{i-1}, c_{o_{i-1}}, p_{i-1}, c_{p_{i-1}})$ be the quintuple previously added to E'. We then add $(\tau_i, o_i, c_{o_i}, p_i, c_{p_i})$ to E' where $c_{o_i} = c_{p_{i-1}}$ and

$$c_{p_i} = \begin{cases} c_{o_i} & \text{if } \varphi_{\mathcal{S}}(var(o_i)) = \varphi_{\mathcal{S}}(var(p_i)), \\ c_1 & \text{if } \varphi_{\mathcal{S}}(var(o_i)) = \varphi_{\mathcal{S}}(var(o_1)), \\ c_2 & \text{if } \varphi_{\mathcal{S}}(var(o_i)) = \varphi_{\mathcal{S}}(var(p_1)) \text{ and } \varphi_{\mathcal{S}}(var(o_1)) \neq \varphi_{\mathcal{S}}(var(p_1)), \\ c_3 & \text{if } \varphi_{\mathcal{S}}(var(o_i)) \neq \varphi_{\mathcal{S}}(var(p_i)) \text{ and } c_{p_i} \neq c_3, \\ c_4 & \text{otherwise.} \end{cases}$$

By construction, for every quintuple $(\tau_i, o_i, c_{o_i}, p_i, c_{p_i})$ in E' we now have

- $c_{o_i} = c_{p_i}$ iff $\varphi_{\mathcal{S}}(var(o_i)) = \varphi_{\mathcal{S}}(var(p_i))$; and
- $c_{q_i} = c_{q_1}$ iff $\varphi_{\mathcal{S}}(var(q_i)) = \varphi_{\mathcal{S}}(var(q_1))$ for every $q_i \in \{o_i, p_i\}$ and $q_1 \in \{o_1, p_1\}$.

It remains to argue that E' indeed satisfies all required properties. (1) is trivial by construction. (2) If $\mathtt{X}_i \neq \mathtt{Y}_i$ is a constraint in τ_i, then $\varphi_{\mathcal{S}}(\mathtt{X}) \neq \varphi_{\mathcal{S}}(\mathtt{Y})$ and $\mathtt{X} \not\approx_D \mathtt{Y}$ according to Lemma 14. (3) If $o_i \approx_{\tau_i} p_i$, then $\varphi_{\mathcal{S}}(var(o_i)) = \varphi_{\mathcal{S}}(var(p_i))$ by Lemma 14, and hence $c_{o_i} = c_{p_i}$. (4) Assume there is a constraint $\mathtt{X}_i \neq \mathtt{Y}_i$ in a template τ_i with $\mathtt{X}_i \approx_{\tau_i} var(o_i)$ and $\mathtt{Y}_i \approx_{\tau_i} var(p_i)$. By Lemma 14, $\varphi_{\mathcal{S}}(\mathtt{X}_i) = \varphi_{\mathcal{S}}(var(o_i)) \neq \varphi_{\mathcal{S}}(var(p_i)) = \varphi_{\mathcal{S}}(\mathtt{Y}_i)$, and therefore $c_{o_i} \neq c_{p_i}$. (5) Let $c_{q_i} = c_{q_1}$ for some $q_i \in \{o_i, p_i\}$ and $q_1 \in \{o_1, p_1\}$, with $i \neq 1$. Assume towards a contradiction that there is an operation o'_i in τ_i potentially ww-conflicting with an operation o'_1 in $\mathsf{prefix}_{o_1}(\tau_1)$ with $var(o'_i) \approx_{\tau_i} var(q_i)$ and $var(o'_1) \approx_{\tau_1} var(q_1)$. But then $\varphi_{\mathcal{S}}(var(o'_i)) = \varphi_{\mathcal{S}}(var(q_i)) = \varphi_{\mathcal{S}}(var(q_1)) = \varphi_{\mathcal{S}}(var(o'_1))$, implying that $\bar{\mu}(o'_i)$ is ww-conflicting with $\bar{\mu}(o'_1)$, contradicting the properties of C stated in Definition 6. (6) is again trivial by construction. (7) By Definition 6, $\bar{\mu}(o_1)$ is rw-conflicting with $\bar{\mu}(p_2)$ in C. Therefore, o_1 is potentially rw-conflicting with p_2. (8) By Definition 6, $\bar{\mu}(o_1) <_{\bar{\mu}(\tau_1)} \bar{\mu}(p_1)$ or $\bar{\mu}(o_m)$ is rw-conflicting with $\bar{\mu}(p_1)$ in C. As a result, $o_1 <_{\tau_1} p_1$ or o_m is rw-conflicting with p_1. ◀

B.4 NLOGSPACE upper bound

Theorem 12 relies on the characterization provided by Lemma 15 and the ability to guess a sequence of quintuples and verify with logarithmic space if it has the properties of Lemma 15.

The algorithm goes as follows: We start by guessing three initial quintuples, representing respectively the first, second and last quintuple of the possible sequence of quintuples as in Lemma 15. Consistent with previously used notation, we refer to these quintuples by E_1, E_2, and E_m, with $E_i = (\tau_i, o_i, c_{o_i}, p_i, c_{p_i})$. Note that the indices we use here are not part of the algorithm. They are only used to distinguish between the different considered quintuples in the proof argument.

We store all three quintuples using a logarithmic amount of space, by storing pointers to the respective transaction templates in $\mathcal{P}$, the positions of operations in the respective transaction templates, and the number $1, 2, 3$ or 4 for the type mappings.

At this point, we verify that Condition (7) and (8) are true, that Conditions (1-5) are true for all chosen transaction templates and operations, and that Condition (6) is true for τ_1 and τ_2, and τ_2 and τ_m. We reject the guessed quintuples if any of the conditions is false.

If all previous checks are true, we proceed by inserting another step. Let $i = 2$. We guess a new quintuple E_{i+1} and verify that Condition (5) is true for τ_i and τ_{i+1} and that Conditions (1-6) are true for τ_{i+1} and reject the entire construction if one of these conditions failed. Notice that all Conditions, including Condition (5) can be checked easily, particularly because quintuple E_1 is stored. To proceed, we discard quintuple E_i and store E_{i+1} instead, thus without increasing the amount of space we use.

If τ_{i+1} and τ_m (from quintuple E_m) have Condition (6), the algorithm emits an accept. Indeed, then the sequence $E_1, \ldots, E_i, E_{i+1}, E_m$ of guessed quintuples has all the properties of Lemma 15. Otherwise, the algorithm proceeds with another insertion step, for $i = i + 1$.

C Proofs for Section 6

C.1 Proof for Lemma 17

▶ **Lemma 17.** *Let D be a sequence of potentially conflicting quadruples over a set of transaction templates $\mathcal{P} \in$ **AcycTemp**. For every pair of variables $\mathtt{X}, \mathtt{Y}$ occurring in $Trans(D)$, if $\mathtt{X} \overset{F}{\rightsquigarrow}_D \mathtt{Y}$, then type$(\mathtt{X}) \overset{F}{\rightsquigarrow}_{SG}$ type$(\mathtt{Y})$, with SG the corresponding schema graph.*

Proof. Let τ_i and τ_j be the templates in D in which $\mathtt{X}$ and $\mathtt{Y}$ occur, respectively. The proof is by induction on the definition of $\mathtt{X} \overset{F}{\rightsquigarrow}_D \mathtt{Y}$.

(Implication within the same template) If $i = j$ and $\mathtt{X} \overset{F}{\rightsquigarrow}_{\tau_i} \mathtt{Y}$, then either $F = \varepsilon$ and $\mathtt{X} = \mathtt{Y}$, or there is a variable $\mathtt{Z}$ such that $\mathtt{Y} = f(\mathtt{Z})$ is a constraint in $\Gamma(\tau_i)$, $\mathtt{X} \overset{F'}{\rightsquigarrow}_{\tau_i} \mathtt{Z}$ and $F = F' \cdot f$. In the former case, type$(\mathtt{X})$ = type$(\mathtt{Y})$, so type$(\mathtt{X}) \overset{\varepsilon}{\rightsquigarrow}_{SG}$ type$(\mathtt{Y})$ is immediate. In the latter case, it follows by induction that type$(\mathtt{X}) \overset{F'}{\rightsquigarrow}_{SG}$ type$(\mathtt{Z})$. Since $dom(f)$ = type$(\mathtt{Z})$ and $range(f)$ = type$(\mathtt{Y})$, it follows by definition that type$(\mathtt{Z}) \overset{f}{\rightsquigarrow}_{SG}$ type$(\mathtt{Y})$ and furthermore type$(\mathtt{X}) \overset{F}{\rightsquigarrow}_{SG}$ type$(\mathtt{Z})$ holds.

(Implication between templates) If $F = \varepsilon$ and $(\tau_i, o_i, p_j, \tau_j)$ (respectively $(\tau_j, o_j, p_i, \tau_i)$) is a potentially conflicting quadruple in D, with $var(o_i) = \mathtt{X}$ and $var(p_j) = \mathtt{Y}$ (respectively

$var(p_i) = \mathtt{X}$ and $var(o_j) = \mathtt{Y}$), then type($\mathtt{X}$) = type($\mathtt{Y}$) by definition of potentially conflicting operations. So, type($\mathtt{X}$) $\overset{\varepsilon}{\rightsquigarrow}_{SG}$ type($\mathtt{Y}$) is again immediate.

(Inductive case) If $\mathtt{X} \overset{F_1}{\rightsquigarrow}_D \mathtt{Z}$ and $\mathtt{Z} \overset{F_2}{\rightsquigarrow}_D \mathtt{Y}$ for some variable $\mathtt{Z}$ with $F = F_1 \cdot F_2$, then type($\mathtt{X}$) $\overset{F_1}{\rightsquigarrow}_{SG}$ type($\mathtt{Z}$) and type($\mathtt{Z}$) $\overset{F_2}{\rightsquigarrow}_{SG}$ type($\mathtt{Y}$) follow by induction. We conclude that type($\mathtt{X}$) $\overset{F}{\rightsquigarrow}_{SG}$ type($\mathtt{Y}$). ◀

C.2 Proof for Theorem 16

We call node S a *descendant* of node R and R an *ancestor* of S in an acyclic schema graph SG. We write $R \overset{\varepsilon}{\rightsquigarrow}_{SG} S$, with ε denoting the empty labeling, for the case $R = S$. This means that a node is a descendant and ancestor of itself. When F is not relevant, we simply write $R \rightsquigarrow_{SG} S$.

Let c_R and c_S be two tuple-contexts for types R and S, respectively, such that S is a descendant of R in SG, witnessed by the path $R \overset{F}{\rightsquigarrow}_{SG} S$ in SG. We then say that c_S *is a tuple-subcontext of* c_R *witnessed by* F if $c_S(S \overset{F'}{\rightsquigarrow}_{SG} S') = c_R(R \overset{F \cdot F'}{\rightsquigarrow}_{SG} S')$ for every path $S \overset{F'}{\rightsquigarrow}_{SG} S'$ in SG. It should be noted that $R \overset{F \cdot F'}{\rightsquigarrow}_{SG} S'$ is indeed a valid path in SG, as it concatenates the paths $R \overset{F}{\rightsquigarrow}_{SG} S$ and $S \overset{F'}{\rightsquigarrow}_{SG} S'$. For a given tuple-context c for a type R in the schema graph SG, we will often write $c(F)$ as a shorthand notation for $c(R \overset{F}{\rightsquigarrow}_{SG} S)$.

Similar to Lemma 14 for sets of transaction templates admitting multi-tree bijectivity, Lemma 34 shows that we can represent a counterexample schedule based on a sequence of potentially conflicting quadruples D over an acyclic schema by assigning a tuple-context to each variable in Trans(D), taking special care when assigning contexts to variables connected in D to make sure that they are properly related to each other. For a set of tuple-contexts $\mathcal{A}$, we refer to a (partial) function $\varphi_{\mathcal{A}} : \mathbf{Var} \to \mathcal{A}$ mapping (a subset of) variables in Trans(D) to tuple-contexts in $\mathcal{A}$ as a *(partial) context assignment for* D *over* $\mathcal{A}$. We furthermore say that $\varphi_{\mathcal{A}}$ is a *total context assignment for* D *over* $\mathcal{A}$ if $\varphi_{\mathcal{A}}$ is defined for every variable in Trans(D).

Two variables $\mathtt{X}$ and $\mathtt{Y}$ occurring in Trans(D) are *equivalent in* D, denoted $\mathtt{X} \equiv_D \mathtt{Y}$ if

- $\mathtt{X} = \mathtt{Y}$;
- there exists a pair of variables $\mathtt{Z}$ and $\mathtt{W}$ and a sequence of function names F with $\mathtt{Z} \equiv_D \mathtt{W}$, $\mathtt{Z} \overset{F}{\rightsquigarrow}_D \mathtt{X}$ and $\mathtt{W} \overset{F}{\rightsquigarrow}_D \mathtt{Y}$; or
- there exists a variable $\mathtt{Z}$ with $\mathtt{X} \equiv_D \mathtt{Z}$ and $\mathtt{Z} \equiv_D \mathtt{Y}$.

Similarly, two variables $\mathtt{X}$ and $\mathtt{Y}$ occurring in a transaction template τ are *equivalent in* τ, denoted $\mathtt{X} \equiv_\tau \mathtt{Y}$ if

- $\mathtt{X} = \mathtt{Y}$;
- there exists a pair of variables $\mathtt{Z}$ and $\mathtt{W}$ in τ and a sequence of function names F with $\mathtt{Z} \equiv_\tau \mathtt{W}$, $\mathtt{Z} \overset{F}{\rightsquigarrow}_\tau \mathtt{X}$ and $\mathtt{W} \overset{F}{\rightsquigarrow}_\tau \mathtt{Y}$; or
- there exists a variable $\mathtt{Z}$ with $\mathtt{X} \equiv_\tau \mathtt{Z}$ and $\mathtt{Y} \equiv_\tau \mathtt{Z}$.

Intuitively, every variable mapping admissible for a given database will assign the same tuple to equivalent variables (see Lemma 32). Due to these equivalent variables, the assignment of a tuple to a variable $\mathtt{X}$ for a given database might imply the tuple assigned to a variable $\mathtt{Y}$, even if $\mathtt{X} \rightsquigarrow_D \mathtt{Y}$ does not hold. We capture this observation by introducing *variable determination*, which is stronger than the previously defined variable implication. Formally, a variable $\mathtt{X}$ *determines* a variable $\mathtt{Y}$ in D witnessed by a sequence of function names F, denoted $\mathtt{X} \overset{F}{\Rrightarrow}_D \mathtt{Y}$ if:

- $\mathtt{X} \overset{F}{\rightsquigarrow}_D \mathtt{Y}$;
- $F = \varepsilon$ and $\mathtt{X} \equiv_D \mathtt{Y}$; or
- there exists a variable $\mathtt{Z}$ with $\mathtt{X} \overset{F_1}{\Rrightarrow}_D \mathtt{Z}$, $\mathtt{Z} \overset{F_2}{\Rrightarrow}_D \mathtt{Y}$ and $F = F_1 \cdot F_2$.

For two variables $\mathtt{X}$ and $\mathtt{Y}$ in a template $\tau \in \text{Trans}(D)$ we furthermore say that $\mathtt{X}$ *determines* $\mathtt{Y}$ in τ witnessed by a sequence of function names F, denoted $\mathtt{X} \overset{F}{\Rightarrow}_\tau \mathtt{Y}$ if:

- $\mathtt{X} \overset{F}{\rightsquigarrow}_\tau \mathtt{Y}$;
- $F = \varepsilon$ and $\mathtt{X} \equiv_\tau \mathtt{Y}$; or
- there exists a variable $\mathtt{Z}$ with $\mathtt{X} \overset{F_1}{\Rightarrow}_\tau \mathtt{Z}$, $\mathtt{Z} \overset{F_2}{\Rightarrow}_\tau \mathtt{Y}$ and $F = F_1 \cdot F_2$.

▶ **Lemma 31.** *For a multiversion split schedule s based on a sequence of conflicting quadruples C over a set of transactions $\mathcal{T}$ consistent with a set of transaction templates $\mathcal{P}$ and a database $\mathbf{D}$, let $\bar{\mu}$ be the variable mapping for a sequence of potentially conflicting quadruples D over $\mathcal{P}$ with $\bar{\mu}(D) = C$. Then, for every combination of variables $\mathtt{W}, \mathtt{X}, \mathtt{Y}, \mathtt{Z}$ occurring in Trans(D), if $\mathtt{Z} \overset{F}{\rightsquigarrow}_D \mathtt{X}$, $\mathtt{W} \overset{F}{\rightsquigarrow}_D \mathtt{Y}$ and $\bar{\mu}(\mathtt{Z}) = \bar{\mu}(\mathtt{W})$, then $\bar{\mu}(\mathtt{X}) = \bar{\mu}(\mathtt{Y})$.*

Proof. By definition of $\mathtt{Z} \rightsquigarrow_D \mathtt{X}$, there is a sequence of variables $\mathtt{X}_1, \mathtt{X}_2 \ldots, \mathtt{X}_n$ with $\mathtt{X}_1 = \mathtt{Z}$ and $\mathtt{X}_n = \mathtt{X}$ such that for each pair of adjacent variables $\mathtt{X}_i$ and $\mathtt{X}_{i+1}$:

(†) $\mathtt{X}_i$ and $\mathtt{X}_{i+1}$ both occur in the same template $\tau \in \text{Trans}(D)$ and $\mathtt{X}_{i+1} = f(\mathtt{X}_i) \in \Gamma(\tau)$ for some function name f; or

(‡) $\text{type}(\mathtt{X}_i) = \text{type}(\mathtt{X}_{i+1})$ and there is a potentially conflicting quadruple $(\tau_j, o_j, p_k, \tau_k)$ in D with either $var(o_j) = \mathtt{X}_i$ and $var(p_k) = \mathtt{X}_{i+1}$ or $var(p_k) = \mathtt{X}_i$ and $var(o_j) = \mathtt{X}_{i+1}$.

Furthermore, the sequence F corresponds to the function names used in applications of (†). Analogously, $\mathtt{W} \rightsquigarrow_D \mathtt{Y}$, implies a sequence of variables $\mathtt{Y}_1, \mathtt{Y}_2 \ldots, \mathtt{Y}_n$ with $\mathtt{Y}_1 = \mathtt{W}$ and $\mathtt{Y}_m = \mathtt{Y}$ with the same properties. Notice that the lengths of these two sequences of variables, namely n and m, are not necessarily equal to each other and to the length of F due to possible applications of (‡). For a variable $\mathtt{X}_i$ in the sequence $\mathtt{X}_1, \mathtt{X}_2, \ldots, \mathtt{X}_n$, we denote the sequence of function names derived from applications of (†) in the subsequence $\mathtt{X}_i, \ldots, \mathtt{X}_n$ by $\mathit{suffix}_F(\mathtt{X}_i)$. Notice that $\mathit{suffix}_F(\mathtt{X}_i)$ is indeed always a suffix of F, and that $\mathit{suffix}_F(\mathtt{X}_1) = \mathit{suffix}_F(\mathtt{Y}_1) = F$ and $\mathit{suffix}_F(\mathtt{X}_n) = \mathit{suffix}_F(\mathtt{Y}_n) = \varepsilon$.

We argue by induction that for every $i \in [1, n]$ and $j \in [1, m]$, if $\mathit{suffix}_F(\mathtt{X}_i) = \mathit{suffix}_F(\mathtt{Y}_j)$ then $\bar{\mu}(\mathtt{X}_i) = \bar{\mu}(\mathtt{Y}_j)$. This then implies $\bar{\mu}(\mathtt{X}) = \bar{\mu}(\mathtt{X}_n) = \bar{\mu}(\mathtt{Y}_m) = \bar{\mu}(\mathtt{Y})$. *(base case)* Note that $\bar{\mu}(\mathtt{X}_1) = \bar{\mu}(\mathtt{Z}) = \bar{\mu}(\mathtt{W}) = \bar{\mu}(\mathtt{Y}_1)$ and $\mathit{suffix}_F(\mathtt{X}_1) = F = \mathit{suffix}_F(\mathtt{Y}_1)$. *(inductive case)* Let $\mathit{suffix}_F(\mathtt{X}_{i+1}) = \mathit{suffix}_F(\mathtt{Y}_{j+1})$. Then, we distinguish the following cases:

- $\mathit{suffix}_F(\mathtt{X}_i) = \mathit{suffix}_F(\mathtt{X}_{i+1})$: This means that (‡) applies to $\mathtt{X}_i$ and $\mathtt{X}_{i+1}$, and there is a potentially conflicting quadruple $(\tau_k, o_k, p_\ell, \tau_\ell)$ in D with either $var(o_k) = \mathtt{X}_i$ and $var(p_\ell) = \mathtt{X}_{i+1}$ or $var(p_\ell) = \mathtt{X}_i$ and $var(o_k) = \mathtt{X}_{i+1}$. By definition of $\bar{\mu}$, we have $\bar{\mu}(\mathtt{X}_{i+1}) = \bar{\mu}(\mathtt{X}_i)$ and by induction that $\bar{\mu}(\mathtt{X}_{i+1}) = \bar{\mu}(\mathtt{Y}_{j+1})$ implying that $\bar{\mu}(\mathtt{X}_{i+1}) = \bar{\mu}(\mathtt{X}_{j+1})$.
- $\mathit{suffix}_F(\mathtt{Y}_j) = \mathit{suffix}_F(\mathtt{Y}_{j+1})$: similar as previous argument;
- $\mathit{suffix}_F(\mathtt{X}_i) \neq \mathit{suffix}_F(\mathtt{X}_{i+1})$ and $\mathit{suffix}_F(\mathtt{Y}_j) \neq \mathit{suffix}_F(\mathtt{Y}_{j+1})$: Then, (†) applies to both $\mathtt{X}_i$ and $\mathtt{X}_{i+1}$, and $\mathtt{Y}_j$ and $\mathtt{Y}_{j+1}$. Furthermore, $\mathit{suffix}_F(\mathtt{X}_i) = \mathit{suffix}_F(\mathtt{Y}_j) = f \cdot F'$ for some f and F'. By induction, $\bar{\mu}(\mathtt{X}_i) = \bar{\mu}(\mathtt{Y}_j)$. Then, $\mathtt{X}_{i+1} = f(\mathtt{X}_i)$ is a constraint in some template $\tau_k \in \text{Trans}(D)$ and $\mathtt{Y}_{j+1} = f(\mathtt{Y}_j)$ is a constraint in some template $\tau_\ell \in \text{Trans}(D)$. Since $\bar{\mu}$ is admissible for database $\mathbf{D}$ and $\bar{\mu}(\mathtt{X}_i) = \bar{\mu}(\mathtt{Y}_j)$, it follows that $\bar{\mu}(\mathtt{X}_{i+1}) = f^{\mathbf{D}}(\bar{\mu}(\mathtt{X}_i)) = f^{\mathbf{D}}(\bar{\mu}(\mathtt{Y}_j)) = \bar{\mu}(\mathtt{Y}_{j+1})$.

◀

▶ **Lemma 32.** *For a multiversion split schedule s based on a sequence of conflicting quadruples C over a set of transactions $\mathcal{T}$ consistent with a set of transaction templates $\mathcal{P}$ and a database $\mathbf{D}$, let $\bar{\mu}$ be the variable mapping for a sequence of potentially conflicting quadruples D over $\mathcal{P}$ with $\bar{\mu}(D) = C$. Then, for every pair of variables $\mathtt{X}$ and $\mathtt{Y}$ occurring in templates τ_i and τ_j in Trans(D) respectively, if $\mathtt{X} \equiv_D \mathtt{Y}$, then $\bar{\mu}(\mathtt{X}) = \bar{\mu}(\mathtt{Y})$.*

Proof. The proof is by induction on the definition of $\mathtt{X} \equiv_D \mathtt{Y}$. *(base case)* If $\mathtt{X} = \mathtt{Y}$, then the result is immediate. *(inductive cases)* If there are two variables $\mathtt{Z}$ and $\mathtt{W}$ and a sequence of function names F such that $\texttt{Z} \equiv_D \texttt{W}$, $\texttt{Z} \overset{F}{\rightsquigarrow}_D \texttt{X}$ and $\texttt{W} \overset{F}{\rightsquigarrow}_D \texttt{Y}$, then by induction we have $\bar{\mu}(\texttt{Z}) = \bar{\mu}(\texttt{W})$. The proof that $\bar{\mu}(\texttt{X}) = \bar{\mu}(\texttt{Y})$ is now immediate by application of Lemma 31. If instead there is a variable $\texttt{Z}$ with $\texttt{X} \equiv_D \texttt{Z}$ and $\texttt{Y} \equiv_D \texttt{Z}$, then we can argue by induction that $\bar{\mu}(\texttt{X}) = \bar{\mu}(\texttt{Z})$ and $\bar{\mu}(\texttt{Y}) = \bar{\mu}(\texttt{Z})$, and hence $\bar{\mu}(\texttt{X}) = \bar{\mu}(\texttt{Y})$. ◀

▶ **Definition 33.** *Let D be a sequence of potentially conflicting quadruples, $\mathcal{A}$ a set of tuple-contexts and $\varphi_\mathcal{A}$ a partial context assignment for D over $\mathcal{A}$. We say that $\varphi_\mathcal{A}$* respects the constraints of *D if, for every two (not necessarily different) variables $\texttt{X}$ and $\texttt{Y}$ occurring in D that $\varphi_\mathcal{A}$ is defined for, the following conditions are true, where $c_\texttt{X} = \varphi_\mathcal{A}(\texttt{X})$ and $c_\texttt{Y} = \varphi_\mathcal{A}(\texttt{Y})$:*

1. *$c_\texttt{X}$ is a tuple-context for type$(\texttt{X})$;*
2. *for every $\texttt{X} \overset{F_1}{\Rightarrow}_D \texttt{Z}$ and $\texttt{Y} \overset{F_2}{\Rightarrow}_D \texttt{Z}$, $c_\texttt{X}(F_1) = c_\texttt{Y}(F_2)$;*
3. *for every $\texttt{X} \overset{F_1}{\Rightarrow}_D \texttt{W}$ and $\texttt{Y} \overset{F_2}{\Rightarrow}_D \texttt{Z}$ with $\texttt{W} \neq \texttt{Z}$ a constraint in a template τ, $c_\texttt{X}(F_1) \neq c_\texttt{Y}(F_2)$;*
4. *for every $\texttt{X} \overset{F_1}{\Rightarrow}_D \texttt{W}$ and $\texttt{Y} \overset{F_2}{\Rightarrow}_D \texttt{Z}$, if $c_\texttt{X}(F_1) = c_\texttt{Y}(F_2)$, then there is no constraint $\texttt{W} \neq \texttt{Z}$ in a template $\tau \in Trans(D)$;*
5. *if $\texttt{X} \overset{F}{\Rightarrow}_D \texttt{Y}$, then $c_\texttt{Y}$ is a tuple-subcontext of $c_\texttt{X}$ witnessed by F; and*
6. *for every pair of tuple-subcontexts $c'_\texttt{X}$ and $c'_\texttt{Y}$ of $c_\texttt{X}$ and $c_\texttt{Y}$ witnessed by respectively $F_\texttt{X}$ and $F_\texttt{Y}$, if $c_\texttt{X}(F_\texttt{X}) = c_\texttt{Y}(F_\texttt{Y})$, then $c'_\texttt{X} = c'_\texttt{Y}$.*

▶ **Lemma 34.** *For a multiversion split schedule s based on a sequence of conflicting quadruples C over a set of transactions $\mathcal{T}$ consistent with a set of transaction templates $\mathcal{P} \in$ **AcycTemp** and a database $\mathbf{D}$, let $\bar{\mu}$ be the variable mapping for a sequence of potentially conflicting quadruples D over $\mathcal{P}$ with $\bar{\mu}(D) = C$. Then a set $\mathcal{A}$ of tuple-contexts and a total context assignment $\varphi_\mathcal{A}$ for D over $\mathcal{A}$ exist with:*

- *$\varphi_\mathcal{A}$ respects the constraints of D; and*
- *$\bar{\mu}(\texttt{X}) = c_\texttt{X}(\varepsilon)$ for every variable $\texttt{X}$, with $c_\texttt{X} = \varphi_\mathcal{A}(\texttt{X})$.*

Proof. We first assign a tuple-context to each tuple in database $\mathbf{D}$, based on the functions in $\mathbf{D}$. Let $(\mathsf{Rels}, \mathsf{Funcs})$ be the schema over which $\mathcal{P}$ is defined. Since the schema graph $SG(\mathsf{Rels}, \mathsf{Funcs})$ is acyclic, a total order $<_{SG}$ over Rels exists such that there is no path from type R to type S in SG if $R <_{SG} S$. We now assign tuple-contexts to tuples based on the order implied by $<_{SG}$. That is, we first consider all tuples of the type that is ordered first by $<_{SG}$, then all tuples of the type that is ordered second, etc. If there are multiple tuples of the same type, the relative order in which we handle them is not important. For each tuple $\texttt{t}$, we construct a tuple-context $c_\texttt{t}$ with $c_\texttt{t}(\varepsilon) = \texttt{t}$, and for each path $F = f \cdot F'$ in SG starting in type$(\texttt{t})$, set $c_\texttt{t}(F) = c_\texttt{v}(F')$, with $\texttt{v} = f^\mathbf{D}(\texttt{t})$. Notice that $c_\texttt{v}$ is already defined for $\texttt{v}$, as there is a path from type$(\texttt{t})$ to type$(\texttt{v})$ in SG and, hence, type$(\texttt{v}) <_{SG}$ type$(\texttt{t})$. By construction, $c_\texttt{v}$ is a tuple-subcontext of $c_\texttt{t}$ witnessed by f.

Next, we construct $\varphi_\mathcal{A}$ as follows: $\varphi_\mathcal{A}(\texttt{X}) = c_\texttt{t}$ with $\bar{\mu}(\texttt{X}) = \texttt{t}$ for every variable $\texttt{X}$ occurring in Trans(D). We argue by induction on the definition of $\overset{F}{\Rightarrow}_D$ that

$$c_\texttt{t}(F) = \bar{\mu}(\texttt{Y}) \text{ for every } \texttt{X} \overset{F}{\Rightarrow}_D \texttt{Y} \text{ (with } c_\texttt{t} = \varphi_\mathcal{A}(\texttt{X})). \qquad (\dagger)$$

If $\texttt{X} \overset{F}{\rightsquigarrow}_D \texttt{Y}$, then by construction of $\varphi_\mathcal{A}$ and since $\bar{\mu}$ is admissible for $\mathbf{D}$, we have $c_\texttt{t}(F) = \bar{\mu}(\texttt{Y})$. If $F = \varepsilon$ and $\texttt{X} \equiv_D \texttt{Y}$, then $c_\texttt{t}(\varepsilon) = \bar{\mu}(\texttt{X}) = \bar{\mu}(\texttt{Y})$ by Lemma 32. Otherwise, if there exists a variable $\texttt{Z}$ with $\texttt{X} \overset{F_1}{\Rightarrow}_D \texttt{Z}$, $\texttt{Z} \overset{F_2}{\Rightarrow}_D \texttt{Y}$ and $F = F_1 \cdot F_2$, then by induction $c_\texttt{t}(F_1) = \bar{\mu}(\texttt{Z}) = \texttt{v}$ and $c_\texttt{v}(F_2) = \bar{\mu}(\texttt{Y})$, with $\varphi_\mathcal{A}(\texttt{Z}) = c_\texttt{v}$. By construction of $c_\texttt{t}$ and $c_\texttt{v}$, the desired $c_\texttt{t}(F) = \bar{\mu}(\texttt{Y})$ now follows.

It remains to verify that $\varphi_\mathcal{A}$ indeed satisfies all required properties. By construction, $\bar{\mu}(\texttt{X}) = c_\texttt{t}(\varepsilon)$ with $c_\texttt{t} = \varphi_\mathcal{A}(\texttt{X})$, so we only need to show that $\varphi_\mathcal{A}$ respects the constraints of D by verifying all properties in Definition 33. To this end, let $\texttt{X}$ and $\texttt{Y}$ be two variables occurring in Trans(D), and let $c_\texttt{t} = \varphi_\mathcal{A}(\texttt{X})$ and $c_\texttt{v} = \varphi_\mathcal{A}(\texttt{Y})$. *(1)* By construction, $c_\texttt{t}$ is a tuple-context for type($\mathtt{X}$). *(2)* $c_{\mathtt{t}}(F_1) = \bar{\mu}(\mathtt{Z}) = c_{\mathtt{v}}(F_2)$ by (†). *(3)* If $\mathtt{W} \neq \mathtt{Z}$ is a constraint, then $\bar{\mu}(\mathtt{W}) \neq \bar{\mu}(\mathtt{Z})$ as $\bar{\mu}$ is admissible for $\mathbf{D}$. By (†), it follows that $c_{\mathtt{t}}(F_1) = \bar{\mu}(\mathtt{W}) \neq \bar{\mu}(\mathtt{Z}) = c_{\mathtt{v}}(F_2)$. *(4)* If $c_{\mathtt{t}}(F_1) = c_{\mathtt{v}}(F_2)$, then $\bar{\mu}(\mathtt{W}) = c_{\mathtt{t}}(F_1) = c_{\mathtt{v}}(F_2) = \bar{\mu}(\mathtt{Z})$ by (†). Since $\bar{\mu}$ is admissible for $\mathbf{D}$, there cannot be a consraint $\mathtt{W} \neq \mathtt{Z}$. *(5)* If $\mathtt{X} \overset{F}{\Rightarrow}_D \mathtt{Y}$, then by construction of the tuple-contexts $c_{\mathtt{t}}$ and $c_{\mathtt{v}}$ it follows that $c_{\mathtt{v}}$ is a tuple-subcontext of $c_{\mathtt{t}}$ witnessed by F. *(6)* Let $c_{\mathtt{t}'}$ and $c_{\mathtt{v}'}$ be tuple-subcontexts of $c_{\mathtt{t}}$ and $c_{\mathtt{v}}$ witnessed by respectively $F_{\mathtt{X}}$ and $F_{\mathtt{Y}}$. If $c_{\mathtt{t}}(F_{\mathtt{X}}) = c_{\mathtt{v}}(F_{\mathtt{Y}}) = \mathtt{q}$ for some tuple $\mathtt{q}$, then by construction of $c_{\mathtt{t}}$ and $c_{\mathtt{v}}$ we have $c_{\mathtt{t}'} = c_{\mathtt{v}'} = c_{\mathtt{q}}$, with $c_{\mathtt{q}}$ the tuple-context assigned to this tuple $\mathtt{q}$. ◀

From $D = (\tau_1, o_1, p_2, \tau_2), \ldots, (\tau_m, o_m, p_1, \tau_1)$ and $\varphi_{\mathcal{A}}$ as in Lemma 34 we can derive a sequence of quintuples $E = (\tau_1, o_1, c_{o_1}, p_1, c_{p_1}), \ldots, (\tau_m, o_m, c_{o_m}, p_m, c_{p_m})$ such that $c_{o_i} = \varphi_{\mathcal{A}}(\mathtt{X}_i)$ (respectively $c_{p_i} = \varphi_{\mathcal{A}}(\mathtt{Y}_i)$) for $i \in [1, m]$ with o_i (respectively p_i) an operation over variable $\mathtt{X}_i$ (respectively $\mathtt{Y}_i$). Intuitively, this sequence of quintuples can be used to reconstruct the original multiversion split schedule s. To this end, notice that we can derive the original sequence of potentially conflicting quadruples D and a partial context assignment $\varphi'_{\mathcal{A}}$ from E that is defined for each variable $\mathtt{X}_i$ occurring in either an operation o_i or p_i in τ_i. We first show that we can extend this partial context assignment $\varphi'_{\mathcal{A}}$ to a total context assignment respecting the constraints in D (Lemma 35), and then prove that such a total context assignment respecting the constraints in D implies a variable assignment $\bar{\mu}$ such that the $C = \bar{\mu}(D)$ is a valid sequence of conflicting quadruples (Lemma 36).

▶ **Lemma 35.** *Let $D = (\tau_1, o_1, p_2, \tau_2), \ldots, (\tau_m, o_m, p_1, \tau_1)$ be a sequence of potentially conflicting quadruples over a set of transaction templates $\mathcal{P} \in$ **AcycTemp** and $\varphi_{\mathcal{A}}$ a partial context assignment defined for every variable $\mathtt{X}_i$ of o_i and $\mathtt{Y}_i$ of p_i in every τ_i. If*

- *$\varphi_{\mathcal{A}}$ respects the constraints of D; and*
- *for every pair of variables $\mathtt{X}$ and $\mathtt{Y}$ in a template τ_i with $\mathtt{X} \equiv_{\tau_i} \mathtt{Y}$, there is no constraint $\mathtt{X} \neq \mathtt{Y}$ in τ_i;*

then we can extend $\varphi_{\mathcal{A}}$ to a total context assignment $\varphi'_{\mathcal{A}}$ for D respecting the constraints of D.

Proof. By definition of equivalence, $\equiv_D$ partitions all variables occurring in Trans(D) in *equivalence classes*. That is, two variables $\mathtt{X}$ and $\mathtt{Y}$ are in the same equivalence class iff $\mathtt{X} \equiv_D \mathtt{Y}$. For a given variable $\mathtt{X}$, we denote the equivalence class $\mathtt{X}$ belongs to by $[\mathtt{X}]$. Note that for any pair of variables $\mathtt{X}$ and $\mathtt{Y}$ occurring in Trans(D), if $\mathtt{X} \overset{F}{\Rightarrow}_D \mathtt{Y}$, then $\mathtt{X}' \overset{F}{\Rightarrow}_D \mathtt{Y}'$ for any pair of variables $\mathtt{X}' \in [\mathtt{X}]$ and $\mathtt{Y}' \in [\mathtt{Y}]$. By slight abuse of notation, we use $\mathtt{X} \overset{F}{\Rightarrow}_D [\mathtt{Y}]$ and $[\mathtt{X}] \overset{F}{\Rightarrow}_D \mathtt{Y}$ to denote that $\mathtt{X} \overset{F}{\Rightarrow}_D \mathtt{Y}'$ for every $\mathtt{Y}' \in [\mathtt{Y}]$ and $\mathtt{X}' \overset{F}{\Rightarrow}_D \mathtt{Y}$ for every $\mathtt{X}' \in [\mathtt{X}]$, respectively.

Let $(\mathsf{Rels}, \mathsf{Funcs})$ be the schema over which $\mathcal{P}$ is defined. Since the schema graph $SG(\mathsf{Rels}, \mathsf{Funcs})$ is acyclic, a total order $<_{SG}$ over Rels exists such that there is no path from type R to type S in SG if $R <_{SG} S$. We now define $\varphi'_{\mathcal{A}}$ for variables in Trans(D) according to the order implied by $<_{SG}$. If there are multiple variables of the same type, the relative order in which we handle them is not important.

The proof is as follows. Assume $\varphi_{\mathcal{A}}$ respects the constraints of D and is at least defined for every variable $\mathtt{X}_i$ of o_i and $\mathtt{Y}_i$ of p_i in every τ_i. We extend $\varphi_{\mathcal{A}}$ towards $\varphi'_{\mathcal{A}}$ by defining $\varphi'_{\mathcal{A}}$ for the whole equivalence class $[\mathtt{X}]$ of the first (according to $<_{SG}$) variable $\mathtt{X}$ for which $\varphi_{\mathcal{A}}$ is not defined. The precise construction is by case. In the first case, the tuple-context that should be assigned to variables in $[\mathtt{X}]$ is already implied, as it is the tuple-subcontext of an existing tuple-context. In the second case, we construct a fresh tuple-context, including existing tuple-contexts as tuple-subcontexts where we need to make sure that $\varphi'_{\mathcal{A}}$ respects the constraints in D. In each case, we then argue that $\varphi'_{\mathcal{A}}$ still respects the constraints in

D. By repeating this argument, we can extend the context assignment to a total context assignment defined for all variables occurring in Trans(D).

(Case 1) If a variable $\mathtt{Y}$ exists with $\varphi_{\mathcal{A}}$ defined for $\mathtt{Y}$ and $\mathtt{Y} \overset{F}{\Rightarrow}_D [\mathtt{X}]$, then $\varphi'_{\mathcal{A}}(\mathtt{X}') = c_{\mathtt{X}'}$ for every variable $\mathtt{X}' \in [\mathtt{X}]$, with $c_{\mathtt{X}'}$ the tuple-subcontext of $c_{\mathtt{Y}} = \varphi_{\mathcal{A}}(\mathtt{Y})$ witnessed by F. Notice that this is well defined, even if there are multiple such $\mathtt{Y}$, as they all agree on $c_{\mathtt{X}'}$ by Definition 33 *(2, 6)*. Also note that the special case where $\varphi_{\mathcal{A}}$ is already defined for at least one variable $\mathtt{X}' \in [\mathtt{X}]$ is covered by this case as well, as $\mathtt{X}' \overset{\varepsilon}{\Rightarrow}_D [\mathtt{X}]$ follows from $\mathtt{X}' \in [\mathtt{X}]$. In this special case, the tuple-subcontext of $\varphi_{\mathcal{A}}(\mathtt{X}')$ witnessed by ε (i.e., $\varphi_{\mathcal{A}}(\mathtt{X}')$ itself) will be assigned to each variable in $[\mathtt{X}]$.

We show that $\varphi'_{\mathcal{A}}$ indeed respects the constraints in D according to the properties stated in Definition 33. To this end, let $\mathtt{X}'$ and $\mathtt{Y}'$ be two variables, with $c_{\mathtt{X}'} = \varphi'_{\mathcal{A}}(\mathtt{X}')$ and $c_{\mathtt{Y}'} = \varphi'_{\mathcal{A}}(\mathtt{Y}')$. *(1)* By construction, $c_{\mathtt{X}'}$ is a tuple-context for type($\mathtt{X}'$). *(2-4)* Note that if $\mathtt{X}'' \overset{F'}{\Rightarrow}_D \mathtt{Z}''$ with $\mathtt{X}'' \in [\mathtt{X}]$, then $\mathtt{Y} \overset{F \cdot F'}{\Rightarrow}_D \mathtt{Z}''$ and $c_{\mathtt{X}''} = \varphi'_{\mathcal{A}}(\mathtt{X}'')$ is the tuple-subcontext of $c_{\mathtt{Y}} = \varphi_{\mathcal{A}}(\mathtt{Y})$ witnessed by F', implying that $c_{\mathtt{X}''}(F') = c_{\mathtt{Y}}(F \cdot F')$. If $\mathtt{X}'$ and/or $\mathtt{Y}'$ are in $[\mathtt{X}]$, then we can apply this substition and use the fact that $\varphi_{\mathcal{A}}$ respects the constraints in τ to conclude that the desired properties hold for $\varphi'_{\mathcal{A}}$. *(5)* Assume $\mathtt{X}' \overset{F'}{\Rightarrow}_D \mathtt{Y}'$. If both $\mathtt{X}' \in [\mathtt{X}]$ and $\mathtt{Y}' \in [\mathtt{X}]$, then $F' = \varepsilon$ as otherwise the schema graph is not acyclic. Since $c_{\mathtt{Y}'} = c_{\mathtt{X}'}$, it follows that $c_{\mathtt{Y}'}$ is a tuple-subcontext of $c_{\mathtt{X}'}$ witnessed by ε. If $\mathtt{X}' \in [\mathtt{X}]$ and $\mathtt{Y}' \notin [\mathtt{X}]$, then $\mathtt{Y} \overset{F \cdot F'}{\Rightarrow}_\tau \mathtt{Y}'$ and $c_{\mathtt{Y}'}$ is a tuple-subcontext of $c_{\mathtt{Y}}$ witnessed by $F \cdot F'$ as $\varphi_{\mathcal{A}}$ respects the constraints of τ. Since $c_{\mathtt{X}'}$ is the tuple-subcontext of $c_{\mathtt{Y}}$ witnessed by F, it follows that $c_{\mathtt{Y}'}$ is a tuple-subcontext of $c_{\mathtt{X}'}$ witnessed by F'. If $\mathtt{X}' \notin [\mathtt{X}]$ and $\mathtt{Y}' \in [\mathtt{X}]$, then $\mathtt{Y} \overset{F}{\Rightarrow}_\tau \mathtt{Y}'$. Since $\varphi_{\mathcal{A}}$ respects the constraints in D, we apply Definition 33 *(2, 5, 6)* to conclude that $c_{\mathtt{Y}'}$ is a tuple-subcontext of $c_{\mathtt{X}'}$ witnessed by F'. *(6)* Assume $c_{\mathtt{X}'}(F_{\mathtt{X}'}) = c_{\mathtt{Y}'}(F_{\mathtt{Y}'})$, and let $c_{\mathtt{X}''}$ and $c_{\mathtt{Y}''}$ be the tuple-subcontexts of respectively $c_{\mathtt{X}'}$ witnessed by $F_{\mathtt{X}'}$ and $c_{\mathtt{Y}'}$ witnessed by $F_{\mathtt{Y}'}$. We argue that $c_{\mathtt{X}''} = c_{\mathtt{Y}''}$. Note that, if $\mathtt{X}' \in [\mathtt{X}]$, then $c_{\mathtt{X}''}$ is the tuple-subcontext of $c_{\mathtt{Y}}$ witnessed by $F \cdot F_{\mathtt{X}'}$. The reasoning for $\mathtt{Y}' \in [\mathtt{X}]$ is analogous. Since $\varphi_{\mathcal{A}}$ respects the constraints in D, it follows that $c_{\mathtt{X}''} = c_{\mathtt{Y}''}$.

(Case 2) Otherwise, we construct a fresh tuple-context $c_{\mathtt{X}}$ and define $\varphi'_{\mathcal{A}}(\mathtt{X}') = c_{\mathtt{X}}$ for every variable $\mathtt{X}' \in [\mathtt{X}]$. This tuple-context $c_{\mathtt{X}}$ is constructed as follows: $c_{\mathtt{X}}(\varepsilon) = \mathtt{t}_{\mathtt{X}}$, with $\mathtt{t}_{\mathtt{X}}$ a fresh tuple of the appropriate type. For every path $F = f \cdot F'$ in SG starting in type($\mathtt{X}$), if there is a variable $\mathtt{Y}$ with $[\mathtt{X}] \overset{f}{\Rightarrow}_D \mathtt{Y}$, then $c_{\mathtt{X}}(F) = c_{\mathtt{Y}}(F')$, with $c_{\mathtt{Y}} = \varphi_{\mathcal{A}}(\mathtt{Y})$. In other words, $c_{\mathtt{Y}}$ is the tuple-subcontext of $c_{\mathtt{X}}$ witnessed by f. Note that due to the order $<_{SG}$, $\varphi_{\mathcal{A}}(\mathtt{Y})$ has to be defined already. Also note that this is well defined, even if multiple such $\mathtt{Y}$ exist. In that case, all these $\mathtt{Y}$ are equivalent to each other by definition of $\equiv_D$, and by construction of $\varphi_{\mathcal{A}}$ they are assigned the same tuple-context. If instead no such variable $\mathtt{Y}$ exists, we define $c_{\mathtt{X}}(F) = \mathtt{t}_F$, with $\mathtt{t}_F$ a fresh tuple of the appropriate type.

We show that $\varphi'_{\mathcal{A}}$ indeed respects the constraints in D according to the properties stated in Definition 33. To this end, let $\mathtt{X}'$ and $\mathtt{Y}'$ be two variables occurring in Trans(D), with $c_{\mathtt{X}'} = \varphi'_{\mathcal{A}}(\mathtt{X}')$ and $c_{\mathtt{Y}'} = \varphi'_{\mathcal{A}}(\mathtt{Y}')$. *(1)* By construction, $c_{\mathtt{X}'}$ is a tuple-context for type($\mathtt{X}'$). *(2)* Assume $\mathtt{X}' \overset{F_1}{\Rightarrow}_D \mathtt{Z}$ and $\mathtt{Y}' \overset{F_2}{\Rightarrow}_D \mathtt{Z}$ for some variable $\mathtt{Z}$. We argue that there exists a pair of variables $\mathtt{X}''$ and $\mathtt{Y}''$ and two sequences of function names F'_1 and F'_2 such that $c_{\mathtt{X}'}(F_1) = c_{\mathtt{X}''}(F'_1) = c_{\mathtt{Y}''}(F'_2) = c_{\mathtt{Y}'}(F_2)$, where $c_{\mathtt{X}''} = \varphi'_{\mathcal{A}}(\mathtt{X}'')$ and $c_{\mathtt{Y}''} = \varphi'_{\mathcal{A}}(\mathtt{Y}'')$. If $\mathtt{X}' \in [\mathtt{X}]$, then either $F_1 = f \cdot F'_1$ or $F_1 = \varepsilon$. In the former case there is a variable $\mathtt{X}''$ with $\mathtt{X}'' \overset{f}{\Rightarrow}_D \mathtt{Z}$ such that $c_{\mathtt{X}'}(F_1) = c_{\mathtt{X}''}(F'_1)$, where $c_{\mathtt{X}''} = \varphi_{\mathcal{A}}(\mathtt{X}'')$. In the later case, $\mathtt{Z} \in [\mathtt{X}]$, and we simply take $\mathtt{X}'' = \mathtt{X}'$ and $F'_1 = F_1$. If $\mathtt{X}' \notin [\mathtt{X}]$, we take $\mathtt{X}'' = \mathtt{X}'$ and $F'_1 = F_1$. For $\mathtt{Y}' \in [\mathtt{X}]$ and $\mathtt{Y}' \not\in [\mathtt{X}]$, the reasoning is analogous. It remains to argue that $c_{\mathtt{X}''}(F'_1) = c_{\mathtt{Y}''}(F'_2)$. By choice of $\mathtt{X}''$ and $\mathtt{Y}''$, either $\mathtt{X}'' \notin [\mathtt{X}]$ and $\mathtt{Y}'' \notin [\mathtt{X}]$; or $\mathtt{Z} \in [\mathtt{X}]$. In te former case, $c_{\mathtt{X}''}(F'_1) = c_{\mathtt{Y}''}(F'_2)$ follows by the fact that $\varphi_{\mathcal{A}}$ respects the constraints of D. In the latter case, both $\mathtt{X}'' \in [\mathtt{X}]$ and $\mathtt{Y}'' \in [\mathtt{X}]$, as otherwise (Case 1) would apply to $[\mathtt{X}]$ instead. Then, $c_{\mathtt{X}''}(F'_1) = c_{\mathtt{Y}''}(F'_2) = c_{\mathtt{X}}(\varepsilon) = \mathtt{t}_{\mathtt{X}}$. *(3, 4)*

The reasoning is analogous to the previous property. Note in particular that by construction of the new $c_{\mathtt{X}}$ we have $c_{\mathtt{X}'}(F_1) = c_{\mathtt{Y}'}(F_2)$ if $\mathtt{W} \equiv_D \mathtt{Z}$. Since $\mathtt{W} \equiv_D \mathtt{Z}$ implies that there is no constraint $\mathtt{W} \neq \mathtt{Z}$ by the assumptions on $\varphi_{\mathcal{A}}$ and on the disequality constraints in each template $\tau \in \text{Trans}(D)$, this does not lead to contradictions. *(5)* If $\mathtt{X}' \overset{F'}{\Rightarrow} \mathtt{Y}'$, then $\mathtt{Y}' \in [\mathtt{X}]$ only if $\mathtt{X}' \in [\mathtt{X}]$, as otherwise (Case 1) would apply to $[\mathtt{X}]$ instead. We argue by case that $c_{\mathtt{Y}'}$ is a tuple-subcontext of $c_{\mathtt{X}'}$ witnessed by F'. If $\mathtt{X}' \notin [\mathtt{X}]$ and $\mathtt{Y}' \notin [\mathtt{X}]$, the result is immediate by the fact that $\varphi_{\mathcal{A}}$ respects the constraints of D. If $\mathtt{X}' \in [\mathtt{X}]$ and $\mathtt{Y}' \notin [\mathtt{X}]$, then $c_{\mathtt{X}'} = c_{\mathtt{X}}$ and a variable $\mathtt{Z}$ exists such that $F' = f \cdot F''$, $\mathtt{X}' \overset{f}{\Rightarrow}_D \mathtt{Z}$, $\mathtt{Z} \overset{F''}{\Rightarrow}_D \mathtt{Y}'$, and, by construction of $c_{\mathtt{X}}$, $c_{\mathtt{Y}'}$ is a tuple-subcontext of $\varphi_{\mathcal{A}}(\mathtt{Z})$ witnessed by F''. It now follows that $c_{\mathtt{Y}'}$ is a tuple-subcontext of $c_{\mathtt{X}}$ witnessed by F'. Lastly, If both $\mathtt{X}' \in [\mathtt{X}]$ and $\mathtt{Y}' \in [\mathtt{X}]$, then $F' = \varepsilon$, as otherwise the schema graph is not acyclic. The result is immediate, as $c_{\mathtt{Y}'} = c_{\mathtt{X}'} = c_{\mathtt{X}}$ is by definition a tuple-subcontext of itself witnessed by ε. *(6)* Assume $c_{\mathtt{X}''}(F_1) = c_{\mathtt{Y}''}(F_1)$ for some pair of tuple-contexts $c_{\mathtt{X}''}$ and $c_{\mathtt{Y}''}$ that are tuple-subcontexts of respectively $c_{\mathtt{X}'}$ witnessed by F_1 and $c_{\mathtt{Y}'}$ witnessed by F_2. We argue that $c_{\mathtt{X}''} = c_{\mathtt{Y}''}$. If both $c_{\mathtt{X}'}$ and $c_{\mathtt{Y}'}$ are different from $c_{\mathtt{X}}$, the result is immediate as $\varphi_{\mathcal{A}}$ respects the constraints of D. Otherwise, since the construction of $c_{\mathtt{X}}$, either copies existing tuple-contexts as tuple-subcontexts, or introduces fresh variables. the result holds if $c_{\mathtt{X}'}$ and/or $c_{\mathtt{Y}'}$ are equal to $c_{\mathtt{X}}$. ◀

▶ **Lemma 36.** *Let $D = (\tau_1, o_1, p_2, \tau_2), \ldots, (\tau_m, o_m, p_1, \tau_1)$ be a sequence of potentially conflicting quadruples over a set of transaction templates $\mathcal{P}$ and $\varphi_{\mathcal{A}}$ a total context assignment for D respecting the constraints of D. The variable mapping $\bar{\mu}$ obtained by defining $\bar{\mu}(\mathtt{X}) = c_{\mathtt{X}}(\varepsilon)$ for every variable $\mathtt{X}$ in Trans(D) with $c_{\mathtt{X}} = \varphi_{\mathcal{A}}(\mathtt{X})$ then is a valid variable mapping admissible for some database $\mathbf{D}$.*

Proof. We first argue that $\bar{\mu}$ is valid by showing for each conflicting quadruple $(\tau_i, o_i, p_j, \tau_j)$ in D that $\bar{\mu}(\mathtt{X}) = \bar{\mu}(\mathtt{Y})$ with $\mathtt{X} = var(o_i)$ and $\mathtt{Y} = var(p_j)$. By definition, $\mathtt{X} \overset{\varepsilon}{\rightsquigarrow}_D \mathtt{Y}$, and hence $\mathtt{X} \overset{\varepsilon}{\Rightarrow}_D \mathtt{Y}$. Let $c_{\mathtt{X}} = \varphi_{\mathcal{A}}(\mathtt{X})$ and $c_{\mathtt{Y}} = \varphi_{\mathcal{A}}(\mathtt{Y})$. Since $\varphi_{\mathcal{A}}$ respects the constraints of D, $c_{\mathtt{Y}}$ is a tuple-subcontext of $c_{\mathtt{X}}$ witnessed by ε. By definition of tuple-subcontexts, $c_{\mathtt{X}}(\varepsilon) = c_{\mathtt{Y}}(\varepsilon)$, and, as a result, $\bar{\mu}(\mathtt{X}) = c_{\mathtt{X}}(\varepsilon) = c_{\mathtt{Y}}(\varepsilon) = \bar{\mu}(\mathtt{Y})$.

Next, we construct a database $\mathbf{D}$ and show that $\bar{\mu}$ is admissible for $\mathbf{D}$. To this end, we add the tuple $\bar{\mu}(\mathtt{X})$ to $\mathbf{D}$ for each variable $\mathtt{X}$ occurring in Trans(D). For each functional constraint $\mathtt{Y} = f(\mathtt{X})$ in a transaction template in Trans(D), we define $\bar{\mu}(\mathtt{Y}) = f^{\mathbf{D}}(\bar{\mu}(\mathtt{X}))$ for the corresponding function $f^{\mathbf{D}}$ in $\mathbf{D}$. Note that this is well defined. Towards a contradiction, assume that we have both $\bar{\mu}(\mathtt{Y}) = f^{\mathbf{D}}(\bar{\mu}(\mathtt{X}))$ and $\bar{\mu}(\mathtt{W}) = f^{\mathbf{D}}(\bar{\mu}(\mathtt{Z}))$, with $\bar{\mu}(\mathtt{X}) = \bar{\mu}(\mathtt{Z})$ but $\bar{\mu}(\mathtt{Y}) \neq \bar{\mu}(\mathtt{W})$. Let $c_{\mathtt{X}} = \varphi_{\mathcal{A}}(\mathtt{X})$, $c_{\mathtt{Y}} = \varphi_{\mathcal{A}}(\mathtt{Y})$, $c_{\mathtt{Z}} = \varphi_{\mathcal{A}}(\mathtt{Z})$ and $c_{\mathtt{W}} = \varphi_{\mathcal{A}}(\mathtt{W})$. By construction of $\bar{\mu}$, we have $c_{\mathtt{X}}(\varepsilon) = \bar{\mu}(\mathtt{X}) = \bar{\mu}(\mathtt{Z}) = c_{\mathtt{Z}}(\varepsilon)$ and $c_{\mathtt{Y}}(\varepsilon) = \bar{\mu}(\mathtt{Y}) \neq \bar{\mu}(\mathtt{W}) = c_{\mathtt{W}}(\varepsilon)$. By Definition 33 *(6)*, it now follows that $c_{\mathtt{X}} = c_{\mathtt{Z}}$, since $c_{\mathtt{X}}$ (respectively $c_{\mathtt{Z}}$) is a tuple-subcontext of itself witnessed by ε and $c_{\mathtt{X}}(\varepsilon) = c_{\mathtt{Z}}(\varepsilon)$. Since we defined $\bar{\mu}(\mathtt{Y}) = f^{\mathbf{D}}(\bar{\mu}(\mathtt{X}))$, there is a constraint $\mathtt{Y} = f(\mathtt{X})$ in some template in Trans(D), and hence $\mathtt{X} \overset{f}{\Rightarrow}_D \mathtt{Y}$. Analogously, $\mathtt{Z} \overset{f}{\Rightarrow}_D \mathtt{W}$. By Definition 33 *(5)*, $c_{\mathtt{Y}}$ and $c_{\mathtt{W}}$ are tuple-subcontexts of respectively $c_{\mathtt{X}}$ and $c_{\mathtt{Z}}$ witnessed by f. As $c_{\mathtt{X}} = c_{\mathtt{Z}}$, it immediately follows that $c_{\mathtt{Y}} = c_{\mathtt{W}}$, and in particular $c_{\mathtt{Y}}(\varepsilon) = c_{\mathtt{W}}(\varepsilon)$, leading to the desired contradiction.

To conclude the proof, we show that $\bar{\mu}$ is indeed admissible for $\mathbf{D}$. By construction of $\mathbf{D}$ based on $\bar{\mu}$, $\bar{\mu}(\mathtt{Y}) = f^{\mathbf{D}}(\bar{\mu}(\mathtt{X}))$ is immediate for each constraint $\mathtt{Y} = f(\mathtt{X})$ in a template $\tau \in$ Trans(D). We still need to argue that $\bar{\mu}(\mathtt{X}) \neq \bar{\mu}(\mathtt{Y})$ for each constraint $\mathtt{X} \neq \mathtt{Y}$ in a template $\tau \in$ Trans(D). Let $c_{\mathtt{X}} = \varphi_{\mathcal{A}}(\mathtt{X})$ and $c_{\mathtt{Y}} = \varphi_{\mathcal{A}}(\mathtt{Y})$. By construction of $\bar{\mu}$ we have $\bar{\mu}(\mathtt{X}) = c_{\mathtt{X}}(\varepsilon)$ and $\bar{\mu}(\mathtt{Y}) = c_{\mathtt{Y}}(\varepsilon)$. Note that $\mathtt{X} \overset{\varepsilon}{\Rightarrow}_D \mathtt{X}$ and $\mathtt{Y} \overset{\varepsilon}{\Rightarrow}_D \mathtt{Y}$. Therefore, we can apply Definition 33 *(3)* to conclude that $c_{\mathtt{X}}(\varepsilon) \neq c_{\mathtt{Y}}(\varepsilon)$, and hence $\bar{\mu}(\mathtt{X}) \neq \bar{\mu}(\mathtt{Y})$. ◀

In order to decide robustness against RC, one can now construct a sequence of quintuples

E and derive the sequence of potentially conflicting quadruples D and partial context assignment $\varphi_{\mathcal{A}}$ from it. If $\varphi_{\mathcal{A}}$ respects the constraints in D, then it follows from Lemma 35 and Lemma 36 that we can construct a variable assignment $\bar{\mu}$ such that $C = \bar{\mu}(D)$ is a valid sequence of conflicting quadruples. However, in this construction of E, care should be taken to guarantee that $\varphi_{\mathcal{A}}$ indeed respects the constraints in D, and that the resulting multiversion split schedule based on C indeed satisfies all properties in Definition 6.

In the algorithm that we are about to propose, we search for such a sequence of quintuples E, but without fixating all the tuples in each context. For this, we generalize our definition of tuple-contexts to allow variables: A *context for a type* R is a function from paths with source R in $SG(\mathsf{Rels}, \mathsf{Funcs})$ to variables in **Var** and tuples in **Tuples** of the appropriate type. The purpose of variables is to encode equalities and disequalities within each context, without being explicit about the precise tuples. That is, if two paths ending in the same node in SG are mapped on the same variable, then they will represent the same tuple; if they are mapped on different variables, then they represent a different tuple. We remark that a same variable occurring in different contexts can still represent different tuples. Analogous to tuple-subcontexts, for two types R and S with $R \overset{F}{\rightsquigarrow}_{SG} S$, we say that a context c_S for type S is a *subcontext* of a context c_R for type R witnessed by F if:

- for every path $S \overset{F'}{\rightsquigarrow}_{SG} S'$ in SG, if $c_R(F \cdot F')$ is a tuple, then $c_S(F') = c_R(F \cdot F')$; otherwise, $c_S(F')$ is a variable; and
- for every pair of paths $S \overset{F_1}{\rightsquigarrow}_{SG} S_1$ and $S \overset{F_2}{\rightsquigarrow}_{SG} S_2$ in SG with $c_R(F \cdot F_1)$ and $c_R(F \cdot F_2)$ variables, $c_S(F_1) = c_S(F_2)$ iff $c_R(F \cdot F_1) = c_R(F \cdot F_2)$.

We call a context a *variable-context* if all paths are mapped on variables.

For a transaction template τ, tuple-context c_p for p and c_o for o in τ, we consider the set $\mathsf{Contexts}(SG, \tau, p, c_p, o, c_o)$ of all different (not-necessarily tuple-) contexts c (up to isomorphisms over the variables in c) that can be obtained, starting from a variable-context c', by performing substitutions of subcontexts of c' with subcontexts of c_p and/or c_o. More formally, these substitutions are of the form: For a path $R_c \overset{F}{\rightsquigarrow}_{SG} S$ (here R_c is the type that c is for) and $R_p \overset{F'}{\rightsquigarrow}_{SG} S$ (with R_p the type that c_p is for) then $c(F \cdot F'') = c_p(F' \cdot F'')$ for every path $S \overset{F''}{\rightsquigarrow}_{SG} S'$ in SG and with $c(F \cdot F'') = c'(F \cdot F'')$ otherwise. (The substitution rule can be applied for c_p as well as for c_o.)

▶ **Lemma 37.** *Let $\mathcal{P}$ be a set of transaction templates over an acyclic schema. Then, $\mathcal{P}$ is not robust against RC if, and only if, there is a sequence of quintuples $E = (\tau_1, o_1, c_{o_1}, p_1, c_{p_1})$, $\ldots, (\tau_m, o_m, c_{o_m}, p_m, c_{p_m})$ with $m \geq 2$ such that for each quintuple $(\tau_i, o_i, c_{o_i}, p_i, c_{p_i})$ in E, with q_i and r_i two (not necessarily different) operations in $\{o_i, p_i\}$,*

1. *if $i = 1$, then c_{o_1} and c_{p_1} are tuple-contexts for type$(var(o_1))$ and type$(var(p_1))$. Furthermore, for every pair of tuple-subcontexts c'_{o_1} and c'_{p_1} of c_{o_1} and c_{p_1} witnessed by respectively F and F', if $c_{o_1}(F) = c_{p_1}(F')$, then $c'_{o_1} = c'_{p_1}$;*
2. *if $i \neq 1$, then $c_{o_i}, c_{p_i} \in \mathsf{Contexts}(SG, \tau_1, p_1, c_{p_1}, o_1, c_{o_1})$ are contexts for type$(var(o_i))$ and type$(var(p_i))$;*
3. *for every pair of variables $\mathtt{W}_i$ and $\mathtt{Z}_i$ in τ_i with $\mathtt{W}_i \equiv_{\tau_i} \mathtt{Z}_i$, there is no constraint $\mathtt{W}_i \neq \mathtt{Z}_i$ in τ_i;*
4. *for every $var(q_i) \overset{F_1}{\Rightarrow}_{\tau_i} \mathtt{Z}_i$ and $var(r_i) \overset{F_2}{\Rightarrow}_{\tau_i} \mathtt{Z}_i$, the subcontext of c_{q_i} witnessed by F_1 is equal (up to isomorphisms over variables) to the subcontext of c_{r_i} witnessed by F_2.*
5. *for every $var(q_i) \overset{F_1}{\Rightarrow}_{\tau_i} \mathtt{W}_i$ and $var(r_i) \overset{F_2}{\Rightarrow}_{\tau_i} \mathtt{Z}_i$ with $\mathtt{W}_i \neq \mathtt{Z}_i$ a constraint in τ_i, $c_{q_i}(F_1) \neq c_{r_i}(F_2)$ or $c_{q_i}(F_1)$ and $c_{r_i}(F_2)$ are both variables;*
6. *for every $var(q_i) \overset{F_1}{\rightsquigarrow}_{\tau_i} \mathtt{W}_i$ and $var(r_i) \overset{F_2}{\rightsquigarrow}_{\tau_i} \mathtt{Z}_i$, if $c_{q_i}(F_1)$ and $c_{r_i}(F_2)$ are the same tuple, then there is no constraint $\mathtt{W}_i \neq \mathtt{Z}_i$ in τ_i;*
7. *if $var(q_i) \overset{F}{\Rightarrow}_{\tau_i} var(r_i)$, then c_{r_i} is a subcontext of c_{q_i} witnessed by F; and*

8. *If $i \neq 1$ and $c_{q_i}(F) = c_{q_1}(F')$ is a tuple for some $q_1 \in \{o_1, p_1\}$ and some sequence of function names F and F', then there is no operation $o'_i \in \tau_i$ potentially ww-conflicting with an operation $o'_1 \in \mathsf{prefix}_{o_1}(\tau_1)$ with $var(q_i) \overset{F}{\Rightarrow}_{\tau_i} var(o'_i)$ and $var(q_1) \overset{F'}{\Rightarrow}_{\tau_1} var(o'_1)$.*

Furthermore, for each pair of adjacent quintuples $(\tau_i, o_i, c_{o_i}, p_i, c_{p_i})$ and $(\tau_j, o_j, c_{o_j}, p_j, c_{p_j})$ in E with $j = i + 1$, or $i = m$ and $j = 1$:

9. *o_i is potentially conflicting with p_j and $c_{o_i} = c_{p_j}$;*

10. *if $i = 1$ and $j = 2$, then o_1 is potentially rw-conflicting with p_2; and*

11. *if $i = m$ and $j = 1$, then $o_1 <_{\tau_1} p_1$ or o_m is potentially rw-conflicting with p_1.*

Proof. *(if)* Let $D = (\tau_1, o_1, p_2, \tau_2), \ldots, (\tau_m, o_m, p_1, \tau_1)$ be the sequence of potentially conflicting quadruples derived from E. Note that each $(\tau_i, o_i, p_j, \tau_j) \in D$ is indeed a valid sequence of potentially conflicting quadruples, as o_i is potentially conflicting with p_j by *(9)*. We show in Lemma 38 that a partial context assignment $\varphi_{\mathcal{A}}$ over a set of tuple-contexts $\mathcal{A}$ exists such that

- for every pair of operations o_i and p_i occurring in D, $\varphi_{\mathcal{A}}$ is defined for $var(o_i)$ and $var(p_i)$;
- $\varphi_{\mathcal{A}}$ respects the constraints in D; and
- for every template τ_i in D with $i \neq 1$ and for every $q_i \in \{o_i, p_i\}$ and $q_1 \in \{o_1, p_1\}$, let $c_{q_i} = \varphi_{\mathcal{A}}(var(q_i))$ and $c_{q_1} = \varphi_{\mathcal{A}}(var(q_1))$. If $c_{q_i}(F) = c_{q_1}(F')$ for some sequence of function names F and F', then there is no operation $o'_i \in \tau_i$ potentially ww-conflicting with an operation $o'_1 \in \mathsf{prefix}_{o_1}(\tau_1)$ with $var(q_i) \overset{F}{\Rightarrow}_{\tau_i} var(o'_i)$ and $var(q_1) \overset{F'}{\Rightarrow}_{\tau_1} var(o'_1)$.

Because of *(3)*, we can now apply Lemma 35 extending $\varphi_{\mathcal{A}}$ to a total context assignment defined for all variables occurring in Trans(D), without losing the property that $\varphi_{\mathcal{A}}$ respects all constraints in D. Let $\bar{\mu}$ be the variable mapping obtained by defining $\bar{\mu}(\mathtt{X}) = c_{\mathtt{X}}(\varepsilon)$ for every variable $\mathtt{X}$ in Trans(D) with $c_{\mathtt{X}} = \varphi_{\mathcal{A}}(\mathtt{X})$. By Lemma 36, $\bar{\mu}$ is a valid variable mapping and a database $\mathbf{D}$ exists such that $\bar{\mu}$ is admissible for $\mathbf{D}$.

We now prove that the sequence of conflicting quadruples $C = \bar{\mu}(D)$ satisfies the conditions stated in Definition 6 to show that $\mathcal{P}$ is indeed not robust against RC. Condition (2) and Condition (3) are immediate by respectively *(11)* and *(10)*. Towards a contradiction, assume Condition (1) does not hold. Then, there is an operation o'_i in a template τ_i potentially ww-conflicting with an operation $o'_1 \in \mathsf{prefix}_{o_1}(\tau_1)$, and $\bar{\mu}(var(o'_i)) = \bar{\mu}(var(o'_1))$. Let $c_{o'_i} = \varphi_{\mathcal{A}}(var(o'_i))$ and $c_{o'_1} = \varphi_{\mathcal{A}}(var(o'_1))$. By construction of $\bar{\mu}$, we have $c_{o'_i}(\varepsilon) = c_{o'_1}(\varepsilon)$. By construction of the total context assignment in Lemma 35, this is only the case if for some $q_i \in \{o_i, p_i\}$ and $q_1 \in \{o_1, p_1\}$ it holds that $var(q_i) \overset{F}{\Rightarrow}_{\tau_i} var(o'_i)$ with $c_{q_i}(F) = c_{o'_i}(\varepsilon)$ and $var(q_1) \overset{F'}{\Rightarrow}_{\tau_1} var(o'_1)$ with $c_{q_1}(F') = c_{o'_1}(\varepsilon)$. Consequently, $c_{q_i}(F) = c_{q_1}(F')$, contradicting Lemma 38.

(only if) Since $\mathcal{P}$ is not robust against RC, a multiversion split schedule s exists based on a sequence of conflicting quadruples C over a set of transactions $\mathcal{T}$ consistent with a set of transaction templates $\mathcal{P} \in$ **AcycTemp** and a database $\mathbf{D}$. Let $\bar{\mu}$ be the variable mapping for a sequence of potentially conflicting quadruples $D = (\tau_1, o_1, p_2, \tau_2), \ldots, (\tau_m, o_m, p_1, \tau_1)$ with $\bar{\mu}(D) = C$. By Lemma 34 a set $\mathcal{A}$ of tuple-contexts and a total context assignment $\varphi_{\mathcal{A}}$ for D over $\mathcal{A}$ exist with:

- $\varphi_{\mathcal{A}}$ respects the constraints of D; and
- $\bar{\mu}(\mathtt{X}) = c_{\mathtt{X}}(\varepsilon)$ for every variable $\mathtt{X}$, with $c_{\mathtt{X}} = \varphi_{\mathcal{A}}(\mathtt{X})$.

Let $\lambda : \mathbf{Tuples} \to \mathbf{Tuples} \cup \mathbf{Var}$ with $\lambda(\mathtt{t}) = \mathtt{t}$ if $\mathtt{t}$ occurs in $\varphi_{\mathcal{A}}(var(o_1))$ or $\varphi_{\mathcal{A}}(var(p_1))$; and $\lambda(\mathtt{t}) \in \mathbf{Var}$ otherwise, such that $\lambda(\mathtt{t}) \neq \lambda(\mathtt{v})$ if $\mathtt{t} \neq \mathtt{v}$. From D and $\varphi_{\mathcal{A}}$ we derive the sequence of quintuples $E = (\tau_1, o_1, c_{o_1}, p_1, c_{p_1}), \ldots, (\tau_m, o_m, c_{o_m}, p_m, c_{p_m})$ with $c_{o_i} = \lambda \circ c'_{o_i}$ and $c_{p_i} = \lambda \circ c'_{p_i}$ for each o_i and p_i, where $c'_{o_i} = \varphi_{\mathcal{A}}(var(o_i))$ and $c'_{p_i} = \varphi_{\mathcal{A}}(var(p_i))$. Intuitively, we modify the tuple-contexts for each $var(o_i)$ and $var(p_i)$ as defined by $\varphi_{\mathcal{A}}$ into contexts over tuples and variables by replacing all tuples that do not occur in $\varphi_{\mathcal{A}}(var(o_1))$ and $\varphi_{\mathcal{A}}(var(p_1))$ with unique variables. Note that by construction $c_{o_1} = \varphi_{\mathcal{A}}(var(o_1))$ and $c_{p_1} = \varphi_{\mathcal{A}}(var(p_1))$.

Next, we show that E satisfies all properties. In the argumentation below, we denote $\varphi_{\mathcal{A}}(var(o_i))$ by c'_{o_i} and $\varphi_{\mathcal{A}}(var(p_i))$ by c'_{p_i} for each o_i and p_i in E. *(1)* By construction, $c_{o_1} = c'_{o_1}$ is a tuple-context for type($var(o_1)$), and $c_{p_1} = c'_{p_1}$ is a tuple-context for type($var(p_1)$). By Condition (6) of Definition 33, we have $c'_{o_1} = c'_{p_1}$ if $c_{o_1}(F) = c_{p_1}(F')$ for every pair of tuple-subcontexts c'_{o_1} and c'_{p_1} of c_{o_1} and c_{p_1} witnessed by respectively F and F'. *(2)* This property follows by construction of c_{o_i} and c_{p_i} based on λ and $\varphi_{\mathcal{A}}$. For completeness sake, one should note that for each group of contexts up to isomorphisms over variables, $\mathsf{Contexts}(SG, \tau_1, p_1, c_{p_1}, o_1, c_{o_1})$ contains only one context. W.l.o.g. we can implicitly assume that each c_{o_i} and c_{p_i} in E is replaced by the same context in $\mathsf{Contexts}(SG, \tau_1, p_1, c_{p_1}, o_1, c_{o_1})$ up to isomorphisms, as we will never directly test for equality or disequality between two variables of different contexts. *(3)* Assume $\mathtt{W}_i \equiv_{\tau_i} \mathtt{Z}_i$, then $\mathtt{W}_i \overset{\varepsilon}{\Rightarrow}_D \mathtt{Z}_i$. Since $\varphi_{\mathcal{A}}$ respects the constraints in D, $c_{\mathtt{Z}_i}$ is a tuple-subcontext of $c_{\mathtt{W}_i}$ witnessed by ε, with $c_{\mathtt{Z}_i} = \varphi_{\mathcal{A}}(\mathtt{Z}_i)$ and $c_{\mathtt{W}_i} = \varphi_{\mathcal{A}}(\mathtt{W}_i)$. Then, $c_{\mathtt{Z}_i} = c_{\mathtt{W}_i}$, and in particular $c_{\mathtt{Z}_i}(\varepsilon) = c_{\mathtt{W}_i}(\varepsilon)$. By Lemma 34, $\bar{\mu}(\mathtt{Z}_i) = c_{\mathtt{Z}_i}(\varepsilon) = c_{\mathtt{W}_i}(\varepsilon) = \bar{\mu}(\mathtt{W}_i)$. Since $\bar{\mu}$ is admissible for $\mathbf{D}$, the constraint $\mathtt{W}_i \neq \mathtt{Z}_i$ cannot exist. *(4)* Since $\varphi_{\mathcal{A}}$ respects the constraints in D, it follows from Conditions (2) and (6) in Definition 33 that the tuple-subcontext of c'_{q_i} witnessed by F_1 is equal to the tuple-subcontext of c'_{r_i} witnessed by F_2. As a result, the subcontext of $c_{q_i} = \lambda \circ c'_{r_i}$ witnessed by F_1 is equal (up to isomorphisms over variables) to the subcontext of $c_{r_i} = \lambda \circ c'_{r_i}$ witnessed by F_2. *(5)* Analogous to the previous case, we can conclude that $c_{q_i}(F_1) = \lambda \circ c'_{q_i}(F_1)$ and $c_{r_i}(F_2) = \lambda \circ c'_{r_i}(F_2)$ are either two different tuples, or both a variable. *(6)* If $c_{q_i}(F_1) = \lambda \circ c'_{q_i}(F_1) = \lambda \circ c'_{r_i}(F_2) = c_{r_i}(F_2)$ is a tuple, then $c'_{q_i}(F_1) = c'_{r_i}(F_2)$. Since $\varphi_{\mathcal{A}}$ respects the constraints in D, it follows that there is no constraint $\mathtt{W}_i \neq vz_i$. *(7)* If $var(q_i) \overset{F}{\Rightarrow}_{\tau_i} var(r_i)$, then c'_{r_i} is a tuple-subcontext of c'_{q_i}, as $\varphi_{\mathcal{A}}$ respects the constraints of D. By construction of c_{q_i} and c_{r_i} based on respectively c'_{q_i} and c'_{r_i}, it immediately follows that c_{r_i} is a subcontext of c_{q_i}. The case for $\mathtt{Y}_i \overset{F}{\Rightarrow}_{\tau_i} \mathtt{X}_i$ is analogous. *(8)* Assume towards a contradiction that $c_{q_i}(F) = c_{q_1}(F')$ is a tuple for some $q_1 \in \{o_1, p_1\}$ and some sequence of function names F and F', and there is an operation $o'_i \in \tau_i$ potentially ww-conflicting with an operation $o'_1 \in \mathsf{prefix}_{o_1}(\tau_1)$ with $var(q_i) \overset{F}{\Rightarrow}_{\tau_i} var(o'_i)$ and $var(q_1) \overset{F'}{\Rightarrow}_{\tau_1} var(o'_1)$. Since $c_{q_i}(F) = c_{q_1}(F')$ is a tuple, $c'_{q_i}(F) = c_{q_i}(F) = c_{q_1}(F') = c'_{q_1}(F')$. By definion of $\bar{\mu}$ and since $\varphi_{\mathcal{A}}$ respects the constraints in D, we conclude that $\bar{\mu}(o'_i)$ in $\bar{\mu}(\tau_i)$ is ww-conflicting with $\bar{\mu}(o'_1)$ in $\mathsf{prefix}_{\bar{\mu}(o_1)}(\bar{\mu}(\tau_1))$, thereby contradicting Condition (1) of Definition 6. *(9)* Since E is based on C, the operation o_i is potentially conflicting with p_j. Furthermore, since $var(o_i) \equiv_D var(p_j)$ and since $\varphi_{\mathcal{A}}$ respects the constraints of D, $c'_{o_i} = c'_{p_j}$, and hence $c_{o_i} = c_{p_j}$. *(10)* Immediate by Condition (3) of Definition 6. *(11)* Immediate by Condition (2) of Definition 6. ◀

Central to the correctness of the if-direction of Lemma 37 is the observation that for any sequence E satisfying the stated conditions, we can assign tuples to variables in each context, thereby replacing contexts with tuple-contexts, in such a way that the resulting context assignment over tuple-contexts respects the constraints of the corresponding sequence of potentially conflicting quadruples D. This observation is formalized in the following Lemma:

▶ **Lemma 38.** *Let $E = (\tau_1, o_1, c_{o_1}, p_1, c_{p_1}), \ldots, (\tau_m, o_m, c_{o_m}, p_m, c_{p_m})$ be a sequence of quintuples satisfying the conditions stated in Lemma 37, and let $D = (\tau_1, o_1, p_2, \tau_2), \ldots, (\tau_m, o_m, p_1, \tau_1)$ be the sequence of potentially conflicting quadruples derived from E. Then a partial context assignment $\varphi_{\mathcal{A}}$ over a set of tuple-contexts $\mathcal{A}$ exists such that*

- *for every pair of operations o_i and p_i occurring in D, $\varphi_{\mathcal{A}}$ is defined for $var(o_i)$ and*

$var(p_i)$;

- *$\varphi_{\mathcal{A}}$ respects the constraints in D; and*
- *for every template τ_i in D with $i \neq 1$ and for every $q_i \in \{o_i, p_i\}$ and $q_1 \in \{o_1, p_1\}$, let $c_{q_i} = \varphi_{\mathcal{A}}(var(q_i))$ and $c_{q_1} = \varphi_{\mathcal{A}}(var(q_1))$. If $c_{q_i}(F) = c_{q_1}(F')$ for some sequence of function names F and F', then there is no operation $o'_i \in \tau_i$ potentially ww-conflicting with an operation $o'_1 \in \mathsf{prefix}_{o_1}(\tau_1)$ with $var(q_i) \overset{F}{\Rightarrow}_{\tau_i} var(o'_i)$ and $var(q_1) \overset{F'}{\Rightarrow}_{\tau_1} var(o'_1)$.*

Proof. The general proof idea is as follows. We iteratively extend $\varphi_{\mathcal{A}}$ by deriving $\varphi_{\mathcal{A}}(var(o_i))$ and $\varphi_{\mathcal{A}}(var(p_i))$ from contexts c_{o_i} and c_{p_i} for each quintuple $(\tau_i, o_i, c_{o_i}, p_i, c_{p_i})$ in the order that they appear in E. Afterwards, we argue that $\varphi_{\mathcal{A}}$ respects the constraints in D and for every $q_i \in \{o_i, p_i\}$ with $i \neq 1$ and $q_1 \in \{o_1, p_1\}$ with $c_{q_i} = \varphi_{\mathcal{A}}(var(q_i))$ and $c_{q_1} = \varphi_{\mathcal{A}}(var(q_1))$, and for every pair of sequences of function names F and F' with $c_{q_i}(F) = c_{q_1}(F')$, there is no operation $o'_i \in \tau_i$ potentially ww-conflicting with an operation $o'_1 \in \mathsf{prefix}_{o_1}(\tau_1)$ with $var(q_i) \overset{F}{\Rightarrow}_{\tau_i} var(o'_i)$ and $var(q_1) \overset{F'}{\Rightarrow}_{\tau_1} var(o'_1)$.

Let $(\tau_1, o_1, c_{o_1}, p_1, c_{p_1})$ be the first quintuple in E. We initiate $\varphi_{\mathcal{A}}$ by defining $\varphi_{\mathcal{A}}(var(o_1)) = c_{o_1}$ and $\varphi_{\mathcal{A}}(var(p_1)) = c_{p_1}$. Next, we iteratively extend $\varphi_{\mathcal{A}}$ by considering the remaining quintuples in E in order. To this end, let $(\tau_{i-1}, o_{i-1}, c_{o_{i-1}}, p_{i-1}, c_{p_{i-1}})$ be the last considered quintuple and $(\tau_i, o_i, c_{o_i}, p_i, c_{p_i})$ the next quintuple in E. We define $\varphi_{\mathcal{A}}(var(p_i)) = c'_{p_i} = \varphi_{\mathcal{A}}(var(o_{i-1}))$ and $\varphi_{\mathcal{A}}(var(o_i)) = c'_{o_i} = \lambda_i \circ c_{o_i}$, where $\lambda_i : \mathbf{Tuples} \cup \mathbf{Var} \to \mathbf{Tuples}$ is a function mapping tuples and variables occurring in c_{o_i} to tuples such that[5]

- $\lambda_i(\mathtt{t}) = \mathtt{t}$ for each tuple $\mathtt{t}$ occurring in c_{o_i};
- $\lambda_i(\mathtt{Q}) = \mathtt{v}$ for each variable $\mathtt{Q}$ occurring in c_{o_i} for which there is a variable $\mathtt{Z}$ in τ_i and sequences of function names F, F' and F'' with $var(p_i) \overset{F}{\Rightarrow}_{\tau_i} \mathtt{Z}$, $var(o_i) \overset{F'}{\Rightarrow}_{\tau_i} \mathtt{Z}$, $c'_{p_i}(F \cdot F'') = \mathtt{v}$ and $c_{o_i}(F' \cdot F'') = \mathtt{Q}$; and
- $\lambda_i(\mathtt{Q}) = \mathtt{t}_{i,\mathtt{Q}}$, for the remaining variables $\mathtt{Q}$ in c_{o_i} where $\mathtt{t}_{i,\mathtt{Q}}$ is a fresh tuple.

Note that this λ_i is well defined. In particular, the second rule intuitively states that the tuple-subcontext of the resulting c'_{o_i} witnessed by F' is equal to the tuple-subcontext of c'_{p_i} witnessed by F, given that there is a variable $\mathtt{Z}$ with $var(o_i) \overset{F'}{\Rightarrow}_{\tau_i} \mathtt{Z}$ and $var(p_i) \overset{F}{\Rightarrow}_{\tau_i} \mathtt{Z}$. This substitution is well defined since in this case the subcontext of c_{o_i} witnessed by F' is equal (up to isomorphisms over variables) to the subcontext of c_{p_i} witnessed by F according to Condition 4 of Lemma 37.

It remains to show that $\varphi_{\mathcal{A}}$ indeed satisfies the conditions stated in Lemma 38. To this end, note that by definition of variable determination, if $\mathtt{X} \overset{F}{\Rightarrow}_D \mathtt{Y}$ with $\mathtt{X}$ in a template τ_i and $\mathtt{Y}$ in a template τ_i in Trans(D), then a sequence of variables $\mathtt{X}_{k_1}, \mathtt{Y}_{k_1}, \ldots, \mathtt{X}_{k_m}, \mathtt{Y}_{k_m}$ exists such that (†):

- $\mathtt{X}_{k_1} = \mathtt{X}$ and $\mathtt{Y}_{k_m} = \mathtt{Y}$;
- each pair of (not necessarily different) variables $\mathtt{X}_{k_i}, \mathtt{Y}_{k_i}$ occur in the same template τ_{k_i} in Trans(D) and $\mathtt{X}_{k_i}, \overset{F_i}{\Rightarrow}_{\tau_{k_i}} \mathtt{Y}_{k_i}$ with $F = F_1 \cdot \ldots, F_m$;
- in the implied sequence of templates $\tau_{k_1}, \ldots, \tau_{k_m}$, these $\tau_{k_i}, \ldots, \tau_{k_{i+1}}$ are neighbouring in E (where we assume that τ_1 is neighbouring to τ_n in E); and
- for each pair of variables $\mathtt{Y}_{k_i}, \mathtt{X}_{k_{i+1}}$, there is a sequence of function names F' such that $var(o_{k_i}) \overset{F'}{\Rightarrow}_{\tau_{k_i}} \mathtt{Y}_{k_i}$ and $var(p_{k_{i+1}}) \overset{F'}{\Rightarrow}_{\tau_{k_{i+1}}} \mathtt{X}_{k_{i+1}}$ (i.e., equivalence of $\mathtt{Y}_{k_i}$ and $\mathtt{X}_{k_{i+1}}$ in D is implied by equivalence of $var(o_{k_i})$ and $var(p_{k_{i+1}})$).

In other words, $\mathtt{X} \overset{F}{\Rightarrow}_D \mathtt{Y}$ can be broken down into a sequence of $\mathtt{X}_{k_i} \overset{F_i}{\Rightarrow}_{\tau_{k_i}} \mathtt{Y}_{k_i}$ through a sequence of neighbouring templates, where equivalence between each $\mathtt{Y}_{k_i}$ and $\mathtt{X}_{k_{i+1}}$ is implied

[5] Note that λ_i is defined over variables in the context c_{o_i}. These variables are unrelated to the variables occurring in Trans(D). We therefore denote these variables by $\mathtt{Q}$ instead of the usual $\mathtt{W}, \mathtt{X}, \mathtt{Y}, \mathtt{Z}$ to avoid confusion.

by the variables in the potentially conflicting operations o_{k_i} and $p_{k_{i+1}}$. For ease of exposition, we implicitly assumed that $\tau_{k_1}, \ldots, \tau_{k_m}$ agrees with the order in E. If the order is opposite to the order in E instead, the above still holds, but the occurrences of o_{k_i} and $p_{k_{i+1}}$ should be replaced with p_{k_i} and $o_{k_{i+1}}$.

We argue by construction of $\varphi_{\mathcal{A}}$ that for every pair of variables $\mathtt{X}$ and $\mathtt{Y}$ for which $\varphi_{\mathcal{A}}$ is defined with $c_{\mathtt{X}} = \varphi_{\mathcal{A}}(\mathtt{X})$ and $c_{\mathtt{Y}} = \varphi_{\mathcal{A}}(\mathtt{Y})$, if $c_{\mathtt{X}}(F) = c_{\mathtt{Y}}(F')$ for some sequence of function names F and F', then $c'_{\mathtt{X}} = c'_{\mathtt{Y}}$, where $c'_{\mathtt{X}}$ and $c'_{\mathtt{Y}}$ are the tuple-subcontexts of $c_{\mathtt{X}}$ witnessed by F and $c_{\mathtt{Y}}$ witnessed by F', respectively (‡). If $\mathtt{X} = var(o_1)$ and $\mathtt{Y} = var(p_1)$ (or the other way around), the result is immediate by Lemma 37 (1). Otherwise, let $\mathtt{X}$ be the variable in a template τ_i and $\mathtt{Y}$ the variable in a template τ_j such that $j \leq i$ (i.e., τ_i does not occur before τ_j in E). W.l.o.g., we assume that $\mathtt{X} = var(o_i)$ with $i \neq 1$ (the case where $\mathtt{X} = var(p_i)$ is analogous, as $\varphi_{\mathcal{A}}(var(p_i)) = \varphi_{\mathcal{A}}(var(o_{i-1}))$ by construction). By construction of each c'_{o_i} based on c'_{p_i} and λ_i, if $c'_{o_i}(F_i) = c'_{p_i}(F'_i)$, then the whole tuple-subcontext of c'_{o_i} witnessed by F_i is copied over from the tuple-subcontext of c'_{p_i} witnessed by F'_i. Indeed λ_i introduces fresh tuples whenever the tuple for $c'_{o_i}(F_i)$ is not implied by c'_{p_i}.

The desired properties now follow from (†) and (‡) as well as the conditions in Lemma 37. In particular, $\varphi_{\mathcal{A}}$ respecting the constraints of D can now be derived from Conditions (1, 2, 4-7) in Lemma 37, and the last condition of Lemma 38 follows from Condition (8) in Lemma 37. ◀

A NEXPSPACE algorithm proving the correctness of Theorem 16 is now immediate by Lemma 37, as we can iteratively guess and verify quintuples in E while only keeping track of the very first quintuple and the previous quintuple. Since in an acyclic schema graph the number of paths starting in a given type is at most exponential in the total number of types, each context is defined over at most an exponential number of paths. However, to formally argue that these contexts can be encoded in exponential space, we still need to show that each tuple or variable used in a context can be encoded in at most exponential space. Since the only tuples used are those mentioned in c_{o_1} and c_{p_1}, and since we can reuse the same variables over all contexts, both the maximal number of tuples and the maximal number of variables needed are exponential in the total number of types.

C.3 Proof for Theorem 18

The PSPACE result for workloads over a schema (Rels, Funcs) where the number of paths between any two nodes in the schema graph is bounded by a constant k is immediate by the nondeterministic algorithm based on Lemma 37 presented for Theorem 16. Indeed, in this case, the total number of paths starting in a given type is at most $k.|\mathsf{Rels}|$ and therefore each context is defined over at most a polynomial number of paths, instead of an exponential number of paths for the general case in Theorem 16.

The EXPTIME result for workloads in **AcycResTemp** follows from a deterministic algorithm based on Lemma 37. In the remainder of this section, we first present the algorithm, and then discuss its complexity.

C.3.1 A deterministic algorithm

Towards a deterministic algorithm, assume the first quintuple $(\tau_1, o_1, c_{o_1}, p_1, c_{p_1})$ of E is fixed. We now translate the problem of deciding whether we can extend E such that it satisfies all properties to a graph problem over a graph $G(\tau_1, o_1, c_{o_1}, p_1, c_{p_1})$. This graph is constructed as follows:

- each quintuple $(\tau_i, o_i, c_{o_i}, p_i, c_{p_i})$ satisfying Conditions (2-8) of Lemma 37 is added as a node to $G(\tau_1, o_1, c_{o_1}, p_1, c_{p_1})$; and

- there is an edge from a node $(\tau_i, o_i, c_{o_i}, p_i, c_{p_i})$ to a node $(\tau_j, o_j, c_{o_j}, p_j, c_{p_j})$ if o_i is potentially conflicting with p_j and $c_{o_i} = c_{p_j}$ (c.f. Condition (9) of Lemma 37).

By construction, it is now easy to see that there is a sequence E satisfying Lemma 37 if there is a path from a quintuple $(\tau_2, o_2, c_{o_2}, p_2, c_{p_2})$ to a quintuple $(\tau_m, o_m, c_{o_m}, p_m, c_{p_m})$ in $G(\tau_1, o_1, c_{o_1}, p_1, c_{p_1})$ (where we allow a zero-length path with $2 = m$), such that (†)

- $c_{o_1} = c_{p_2}$ and $c_{o_m} = c_{p_1}$ (c.f. Condition (9) of Lemma 37);
- o_1 is potentially rw-conflicting with p_2 (c.f. Condition (10) of Lemma 37); and
- $o_1 <_{\tau_1} p_1$ or o_m is potentially rw-conflicting with p_1 (c.f. Condition (11) of Lemma 37).

(Algorithm) Given a set of transaction templates $\mathcal{P}$ over a schema $(\mathsf{Rels}, \mathsf{Funcs})$, the algorithm iterates over all possible quintuples $(\tau_1, o_1, c_{o_1}, p_1, c_{p_1})$ satisfying Condition (1, 3-7) of Lemma 37, where we consider all possible tuple-contexts c_{o_1} and c_{p_1} up to isomorphisms. For each such quintuple, the graph $G(\tau_1, o_1, c_{o_1}, p_1, c_{p_1})$ is constructed. Let TC be the reflexive-transitive closure of G. If there is a pair of quintuples $(\tau_2, o_2, c_{o_2}, p_2, c_{p_2})$ and $(\tau_m, o_m, c_{o_m}, p_m, c_{p_m})$ in TC satisfying (†), the algorithm emits a reject, indicating that $\mathcal{P}$ is not robust against RC. Otherwise, it proceeds with a new choice for $(\tau_1, o_1, c_{o_1}, p_1, c_{p_1})$. If, the algorithm didn't reject after considering all such quintuples, it accepts, indicating that $\mathcal{P}$ is indeed robust against RC. The correctness of this algorithm is immediate by Lemma 37.

C.3.2 Complexity analysis

We show the complexity of the presented algorithm. For this, first, notice that we have defined contexts c based on a type S (with S not necessarily a root of SG). For encoding purposes it makes sense to encode these as contexts for a root type R in combination with the intended type S. The context as defined in the previous section can then be derived by taking the left-most subtree with root S. Notice that this is purely an encoding choice that will simplify the analysis.

For a schema graph $SG(\mathsf{Rels}, \mathsf{Funcs})$ the total number of non-isomorphic tuple-contexts can be expressed using Bell's number $B(n)$, denoting the number of partitions for a set of size n, and the set $\text{Paths}_{\text{SG}}(R, S) = \{(R, F, S) \mid R \overset{F}{\rightsquigarrow}_{SG} S\}$ expressing the different paths from one node R to another node S in SG. Concretely,

$$|\mathsf{TupleContexts}(SG)| \leq \sum_{R \in \mathsf{roots}(SG)} \prod_{S \in \mathsf{Rels}} B(|\text{Paths}_{\text{SG}}(R, S)|) = B^*$$

where $\mathsf{TupleContexts}(SG)$ denotes the set containing all different tuple-contexts (up to isomorphisms).

Now let c_1 and c_2 be two fixed contexts for types that are descendants of roots R_1 and root R_2, respectively, in SG, and let c be a context for a type descending from root R. To express a bound on the number of substitutions in c from (parts of) c_1 and c_2, we need some additional terminology: Let $\text{Paths}_{\text{SG}}(R, *) = \bigcup_{S \in \mathsf{Rels}} \text{Paths}_{\text{SG}}(R, S)$. We say that a path $R \overset{F_1}{\rightsquigarrow}_{SG} S$ is a *prefix* of a path $R \overset{F}{\rightsquigarrow}_{SG} S'$ in SG if there is a (possibly empty) sequence of function names F_2 with $F = F_1 \cdot F_2$. The number of substitutions in c from (parts of) c_1 and c_2 is now bounded by

$$\sum_{\text{Part} \subseteq \text{Paths}_{\text{SG}}(R, *)} \mathbf{1}_{PFP} \prod_{R \overset{F}{\rightsquigarrow}_{SG} S \in \text{Part}} (|\text{Paths}_{\text{SG}}(R_1, S)| + |\text{Paths}_{\text{SG}}(R_2, S)|) \leq T^{\ell}.(2P)^{\ell} \leq (2TP)^{\ell}$$

In the above expression, $\mathbf{1}_{PFP}$ is an indicator variable that equals 1 if no path in Part is a prefix of another path in Part and that equals 0 otherwise. Further: P denotes the maximum number of different paths between a particular root and a particular node in SG, Balance:

$\mathtt{R}[\mathtt{X} : \text{Account}\{\text{N, C}\}]$
$\mathtt{R}[\mathtt{Y} : \text{Savings}\{\text{C, B}\}]$
$\mathtt{R}[\mathtt{Z} : \text{Checking}\{\text{C, B}\}]$
$\mathtt{Y} = f_{A\to S}(\mathtt{X}),\ \mathtt{X} = f_{S\to A}(\mathtt{Y})$
$\mathtt{Z} = f_{A\to C}(\mathtt{X}),\ \mathtt{X} = f_{C\to A}(\mathtt{Z})$

DepositChecking:

$\mathtt{R}[\mathtt{X} : \text{Account}\{\text{N, C}\}]$
$\mathtt{U}[\mathtt{Z} : \text{Checking}\{\text{C, B}\}\{\text{B}\}]$
$\mathtt{Z} = f_{A\to C}(\mathtt{X}),\ \mathtt{X} = f_{C\to A}(\mathtt{Z})$

TransactSavings:

$\mathtt{R}[\mathtt{X} : \text{Account}\{\text{N, C}\}]$
$\mathtt{U}[\mathtt{Y} : \text{Savings}\{\text{C, B}\}\{\text{B}\}]$
$\mathtt{Y} = f_{A\to S}(\mathtt{X}),\ \mathtt{X} = f_{S\to A}(\mathtt{Y})$

Amalgamate:

$\mathtt{R}[\mathtt{X}_1 : \text{Account}\{\text{N, C}\}]$
$\mathtt{R}[\mathtt{X}_2 : \text{Account}\{\text{N, C}\}]$
$\mathtt{U}[\mathtt{Y}_1 : \text{Savings}\{\text{C, B}\}\{\text{B}\}]$
$\mathtt{U}[\mathtt{Z}_1 : \text{Checking}\{\text{C, B}\}\{\text{B}\}]$
$\mathtt{U}[\mathtt{Z}_2 : \text{Checking}\{\text{C, B}\}\{\text{B}\}]$
$\mathtt{X}_1 \neq \mathtt{X}_2,$
$\mathtt{Y}_1 = f_{A\to S}(\mathtt{X}_1),\ \mathtt{X}_1 = f_{S\to A}(\mathtt{Y}_1)$
$\mathtt{Y}_2 = f_{A\to S}(\mathtt{X}_2),\ \mathtt{X}_2 = f_{S\to A}(\mathtt{Y}_2)$
$\mathtt{Z}_1 = f_{A\to C}(\mathtt{X}_1),\ \mathtt{X}_1 = f_{C\to A}(\mathtt{Z}_1)$
$\mathtt{Z}_2 = f_{A\to C}(\mathtt{X}_2),\ \mathtt{X}_2 = f_{C\to A}(\mathtt{Z}_2)$

WriteCheck:

$\mathtt{R}[\mathtt{X} : \text{Account}\{\text{N, C}\}]$
$\mathtt{R}[\mathtt{Y} : \text{Savings}\{\text{C, B}\}]$
$\mathtt{R}[\mathtt{Z} : \text{Checking}\{\text{C, B}\}]$
$\mathtt{U}[\mathtt{Z} : \text{Checking}\{\text{C, B}\}\{\text{B}\}]$
$\mathtt{Y} = f_{A\to S}(\mathtt{X}),\ \mathtt{X} = f_{S\to A}(\mathtt{Y})$
$\mathtt{Z} = f_{A\to C}(\mathtt{X}),\ \mathtt{X} = f_{C\to A}(\mathtt{Z})$

GoPremium:

$\mathtt{U}[\mathtt{X} : \text{Account}\{\text{N, C}\}\{\text{I}\}]$
$\mathtt{R}[\mathtt{Y} : \text{Savings}\{\text{C, I}\}]$
$\mathtt{U}[\mathtt{Y} : \text{Savings}\{\text{C}\}\{\text{I}\}]$
$\mathtt{Y} = f_{A\to S}(\mathtt{X}),\ \mathtt{X} = f_{S\to A}(\mathtt{Y})$

Figure 8 Transaction templates for SmallBank.

Account(<u>Name</u>, CustomerID, IsPremium)
Savings(<u>CustomerID</u>, Balance, InterestRate)
Checking(<u>CustomerID</u>, Balance)

Figure 9 Tables of the SmallBank benchmark. Underlined attributes are primary keys.

T denotes the maximum number of different paths from a particular root to nodes in SG, and ℓ denotes the maximal size of a set in which no path is a prefix of another path in the set. The latter is trivially bounded by T.

A special cases exists if all templates τ in $\mathcal{P}$ are restricted. In that case, the size of sets Part is bounded by 2, hence $\ell \leq 2$.

With the above bounds, the complexity of the presented algorithm is rather straightforward. The iteration over all possible quintuples $(\tau_1, o_1, c_{o_1}, p_1, c_{p_1})$ requires at most $|\mathcal{P}|.t^2.\,(B^*)^2$ iterations, with t denoting the maximal number of operations in a transaction template of $\mathcal{P}$. The remainder of the computation is dominated by the transitive-closure computation. Since the constructed graph $G(\tau_1, o_1, c_{o_1}, p_1, c_{p_1})$ has at most $|\mathcal{P}|.t^2.\,\left(B^*.(2TP)^\ell\right)^2$ nodes, the transitive closure computation requires $\left(|\mathcal{P}|.t^2.\,\left(B^*.(2TP)^\ell\right)^2\right)^3$ steps. Putting these numbers together, we obtain:

$$\mathcal{O}(|\mathcal{P}|^4.t^8.\,(B^*)^8\,.(2TP)^{6\ell}).$$

Since ℓ is bounded by a constant if all template are restricted, and since B^*, T and P can be exponential in the size of the input, the presented algorithm indeed decides T-ROBUSTNESS(**AcycResTemp**,RC) in EXPTIME.

D SmallBank and TPC-C benchmarks

D.1 SmallBank Benchmark

The SmallBank schema consists of three tables as given in Figure 9. The Account table associates customer names with IDs and keeps track of the premium status (Boolean);

CustomerID is a `UNIQUE` attribute. The other tables contain the balance (numeric value) of the savings and checking accounts of customers identified by their ID. Account (CustomerID) is a foreign key referencing both the columns Savings (CustomerID) and Checking (CustomerID). The interest rate on a savings account is based on a number of parameters, including the account status (premium or not). The application code can interact with the database only through the following transaction programs:

- Balance(N): returns the total balance (savings & checking) for a customer with name N.
- DepositChecking(N,V): makes a deposit of amount V on the checking account of the customer with name N.
- TransactSavings(N,V): makes a deposit or withdrawal V on the savings account of the customer with name N.
- Amalgamate(N_1,N_2): transfers all the funds from N_1 to N_2.
- WriteCheck(N,V): writes a check V against the account of the customer with name N, penalizing if overdrawing.
- GoPremium(N): converts the account of the customer with name N to a premium account and updates the interest rate of the corresponding savings account. This transaction program is an extension w.r.t. [2].

Figure 10 contains the SQL code for the SmallBank transaction templates presented in Figure 8.

D.2 TPC-C Benchmark

This benchmark is based on the TPC-C benchmark [17]. We modified the schema and templates to turn all predicate reads into key-based accesses. The schema consists of six relations:

- Warehouse(<u>WarehouseID</u>, Info, YTD),
- District(<u>WarehouseID</u>, <u>DistrictID</u>, Info, YTD, NextOrderID),
- Customer(<u>WarehouseID</u>, <u>DistrictID</u>, <u>CustID</u>, Info, Balance),
- Order(<u>WarehouseID</u>, <u>DistrictID</u>, <u>OrderID</u>, CustID, Status),
- OrderLine(<u>WarehouseID</u>, <u>DistrictID</u>, <u>OrderID</u>, <u>OrderLineID</u>, ItemID, DeliveryInfo, Quantity), and
- Stock(<u>WarehouseID</u>, <u>ItemID</u>, Quantity).

The function names belonging to this schema are given in Table 1.

We focus on five different transaction templates:

1. NewOrder(W, D, C, I_1, Q_1, I_2, Q_2, $\dots$): creates a new order for the customer identified by (W, D, C). The id for this order is obtained by increasing the NextOrderID attribute of the District tuple identified by (W, D) by one. Each order consists of a number of items $I_1, I_2, \dots$ with respectively quantities $Q_1, Q_2, \dots$. For each of these items, a new OrderLine tuple is created and the related stock quantity is decreased.
2. Payment(W, D, C, A): represents a customer identified by (W, D, C) paying an amount A. This payment is reflected in the database by increasing the balance of this customer by A. This amount is furthermore added to the YearToDate (YTD) income of both the related warehouse and district.
3. OrderStatus(W, D, C, O): requests information about the current status of the order identified by (W, D, O). This transaction template collects information of the customer identified by (W, D, C) who created the order, the order itself, and the different OrderLine tuples related to this order.
4. Delivery(W, D, C, O): delivers the order represented by (W, D, O). The status of the order is updated, as well as the DeliveryInfo attribute of each OrderLine tuple related to

```
Balance(N):
    SELECT CustomerId INTO :x
      FROM Account
     WHERE Name=:N;

    SELECT Balance INTO :a
      FROM Savings
     WHERE CustomerId=:x;

    SELECT Balance + :a
      FROM Checking
     WHERE CustomerId=:x;
    COMMIT;

Amalgamate(N1,N2):
    SELECT CustomerId INTO :x1
      FROM Account
     WHERE Name=:N1;

    SELECT CustomerId INTO :x2
      FROM Account
     WHERE Name=:N2;

    UPDATE Savings AS new
       SET Balance = 0
      FROM Savings AS old
     WHERE new.CustomerId=:x1
           AND old.CustomerId
           = new.CustomerId
    RETURNING old.Balance INTO :a;

    UPDATE Checking AS new
       SET Balance = 0
      FROM Checking AS old
     WHERE new.CustomerId=:x1
           AND old.CustomerId
           = new.CustomerId
    RETURNING old.Balance INTO :b;

    UPDATE Checking
       SET Balance = Balance + :a + :b
     WHERE CustomerId=:x2;

DepositChecking(N,V):
    SELECT CustomerId INTO :x
      FROM Account
     WHERE Name=:N;

    UPDATE Checking
       SET Balance = Balance + :V
     WHERE CustomerId=:x;
    COMMIT;

TransactSavings(N,V):
    SELECT CustomerId INTO :x
      FROM Account
     WHERE Name=:N;

    UPDATE Savings
       SET Balance = Balance + :V
     WHERE CustomerId=:x;
    COMMIT;

WriteCheck(N,V):
    SELECT CustomerId INTO :x
      FROM Account
     WHERE Name=:N;

    SELECT Balance INTO :a
      FROM Savings
     WHERE CustomerId=:x;

    SELECT Balance INTO :b
      FROM Checking
     WHERE CustomerId=:x;

    IF (:a + :b) < :V THEN
        UPDATE Checking
           SET Balance = Balance - (:V + 1)
         WHERE CustomerId=:x;
    ELSE
        UPDATE Checking
           SET Balance = Balance - :V
         WHERE CustomerId=:x;
    END IF;
    COMMIT;

GoPremium(N):
    UPDATE Account
       SET IsPremium = TRUE
     WHERE Name=:N
    RETURNING CustomerId INTO :x;

    SELECT InterestRate INTO :a
      FROM Savings
     WHERE CustomerId=:x;

    :rate = computePremiumRate(:x,:a);

    UPDATE Savings
       SET InterestRate = :rate
     WHERE CustomerId=:x;
    COMMIT;
```

Figure 10 SmallBank SQL Transaction Templates.

f	$dom(f)$	$range(f)$
$f_{D\to W}$	District	Warehouse
$f_{C\to D}$	Customer	District
$f_{O\to C}$	Order	Customer
$f_{L\to O}$	OrderLine	Order
$f_{L\to S}$	OrderLine	Stock
$f_{S\to W}$	Stock	Warehouse

Table 1 Function names for the TPC-C benchmark schema.

NewOrder:

$\mathtt{R}[\mathtt{X} : \text{Warehouse}\{\text{W, Inf}\}]$
$\mathtt{U}[\mathtt{Y} : \text{District}\{\text{W, D, Inf, N}\}\{\text{N}\}]$
$\mathtt{R}[\mathtt{Z} : \text{Customer}\{\text{W, D, C, Inf}\}]$
$\mathtt{W}[\mathtt{S} : \text{Order}\{\text{W, D O, C, Sta}\}]$
$\mathtt{U}[\mathtt{T}_1 : \text{Stock}\{\text{W, I, Qua}\}\{\text{Qua}\}]$
$\mathtt{W}[\mathtt{V}_1 : \text{OrderLine}\{\text{W, D, O, OL, I, Del, Qua}\}]$
$\mathtt{U}[\mathtt{T}_2 : \text{Stock}\{\text{W, I, Qua}\}\{\text{Qua}\}]$
$\mathtt{W}[\mathtt{V}_2 : \text{OrderLine}\{\text{W, D, O, OL, I, Del, Qua}\}]$
$\mathtt{X} = f_{D\to W}(\mathtt{Y}),\ \mathtt{Y} = f_{C\to D}(\mathtt{Z}),\ \mathtt{Z} = f_{O\to C}(S)$
$S = f_{L\to O}(\mathtt{V}_1),\ S = f_{L\to O}(\mathtt{V}_2)$
$\mathtt{T}_1 = f_{L\to S}(\mathtt{V}_1),\ \mathtt{T}_2 = f_{L\to S}(\mathtt{V}_2)$
$\mathtt{X} = f_{S\to W}(\mathtt{T}_1),\ \mathtt{X} = f_{S\to W}(\mathtt{T}_2)$

Delivery:

$\mathtt{U}[\mathtt{S} : \text{Order}\{\text{W, D, O}\}\{\text{Sta}\}]$
$\mathtt{U}[\mathtt{V}_1 : \text{OrderLine}\{\text{W, D, O, OL, Del}\}\{\text{Del}\}]$
$\mathtt{U}[\mathtt{V}_2 : \text{OrderLine}\{\text{W, D, O, OL, Del}\}\{\text{Del}\}]$
$\mathtt{U}[\mathtt{Z} : \text{Customer}\{\text{W, D, C, Bal}\}\{\text{Bal}\}]$
$\mathtt{Z} = f_{O\to C}(S),\ S = f_{L\to O}(\mathtt{V}_1),\ S = f_{L\to O}(\mathtt{V}_2)$

Payment:

$\mathtt{U}[\mathtt{X} : \text{Warehouse}\{\text{W, YTD}\}\{\text{YTD}\}]$
$\mathtt{U}[\mathtt{Y} : \text{District}\{\text{W, D, YTD}\}\{\text{YTD}\}]$
$\mathtt{U}[\mathtt{Z} : \text{Customer}\{\text{W, D, C, Bal}\}\{\text{Bal}\}]$
$\mathtt{X} = f_{D\to W}(\mathtt{Y}),\ \mathtt{Y} = f_{C\to D}(\mathtt{Z})$

OrderStatus:

$\mathtt{R}[\mathtt{Z} : \text{Customer}\{\text{W, D, C, Inf, Bal}\}]$
$\mathtt{R}[\mathtt{S} : \text{Order}\{\text{W, D, O, C, Sta}\}]$
$\mathtt{R}[\mathtt{V}_1 : \text{OrderLine}\{\text{W, D, O, OL, I, Del, Qua}\}]$
$\mathtt{R}[\mathtt{V}_2 : \text{OrderLine}\{\text{W, D, O, OL, I, Del, Qua}\}]$
$\mathtt{Z} = f_{O\to C}(S),\ S = f_{L\to O}(\mathtt{V}_1),\ S = f_{L\to O}(\mathtt{V}_2)$

StockLevel:

$\mathtt{R}[\mathtt{T} : \text{Stock}\{\text{W, I, Qua}\}]$

Figure 11 Abstraction for the TPC-C transaction templates. Attribute names are abbreviated.

this order. The total price of the order is deduced from the balance of the customer who made this order, identified by (W, D, C).

5. StockLevel(W, I): returns the current stock level of item I in warehouse W.

A detailed abstraction of each transaction template is given in Figure 11. To shorten the presentation, we only show two orderlines per order.