Verifying Serializability

No Institute Given

$$v \in \texttt{Variables} \qquad f \in \texttt{Fieldname} \qquad \mathcal{Q} \in \{\texttt{MIN}, \texttt{MAX}, \texttt{COUNT}\}$$
$$\oplus \in \{+, -, \times, /\} \qquad \odot \in \{<, \leq, =, >, \geq\} \qquad \circ \in \{\wedge, \vee\}$$

$$\begin{array}{lll}
e_d & \coloneqq & f \mid v \mid e_d \oplus e_d \mid \mathbb{Z} \\
\phi_d & \coloneqq & f \odot e_d \mid f \texttt{ IN } v \mid \neg\phi_d \mid \phi_d \circ \phi_d \\
e_c & \coloneqq & v \mid \texttt{CHOOSE } v \mid e_c \oplus e_c \mid \texttt{NULL} \mid \mathbb{Z} \\
\phi_c & \coloneqq & v \odot e_c \mid \neg\phi_c \mid \phi_c \circ \phi_c \\
c & \coloneqq & \texttt{SELECT } \bar{\texttt{f}} \texttt{ AS v WHERE } \phi_\texttt{d} \mid \texttt{SELECT } \mathcal{Q} \texttt{ f AS v WHERE } \phi_\texttt{d} \mid \texttt{UPDATE SET f} = \texttt{e}_\texttt{c} \texttt{ WHERE } \phi_\texttt{d} \mid \\
 & & \texttt{INSERT VALUES } \bar{\texttt{f}} = \bar{\texttt{e}}_\texttt{c} \mid \texttt{DELETE WHERE } \phi_\texttt{d} \mid \texttt{v} = \texttt{e}_\texttt{c} \mid \texttt{IF } \phi_\texttt{c} \texttt{ THEN c ELSE c} \mid \texttt{c ; c} \mid \\
 & & \texttt{FOREACH v}_1 \texttt{ IN v}_2 \texttt{ DO c END} \\
vlist & \coloneqq & v \mid vlist \\
t & \coloneqq & \texttt{Tname}(vlist)\{c\}
\end{array}$$

Let $\mathbb{T} = \{T_1, \ldots, T_n\}$ be the set of transactions to be verified. Every transaction T_i is generated using the grammar described above, and has a set of parameters and local variables $\{v_{i1}, \ldots, v_{il_i}\}$ (also represented as $\bar{v}_i$). The grammar is essentially a simplified version of standard SQL, allowing SQL statements which access the database to be combined with usual program connectives such as conditionals, sequencing and loops. The SQL statements use predicates ϕ_d to refer to the records that would be accessed/modified, where ϕ_d allows all boolean combinations of comparison predicates between fields and values. These predicates are evaluated individually on every record in the database and only those records which satisfy the predicate are operated upon by the SQL statement. When a predicate is being evaluated on a record, the field variables are replaced by the corresponding values of the record in the database. Note that this does not allow comparison between field values across different records. The only exceptions are the `MIN, MAX` qualifiers used with `SELECT`, which return the record with the minimum (resp. maximum) field value in the database. A transaction also has a set of parameter variables which are instantiated with values when the transaction is called, and a set of local variables which are used to store intermediate values from the database (typically as output of `SELECT` queries). For simplicity, we will assume that the transactions are written in Static Single Assignment (SSA) form, so that any local (or parameter) variable is never re-assigned. We assume that the database schema is known. For notational convenience, we will assume that there is only one table and hence all records have the same structure with fields $\{f_1, \ldots, f_m\}$.

To determine whether an execution is serializable, we can construct the dependency graph of the execution and then find out whether there are any cycles. In our encoding, we go in the reverse direction. That is, we build a dependency graph that contains a cycle, and then determine whether it is possible to have a valid execution with the constructed dependency graph. Hence, a majority of

our encoding rules are of the form $\mathcal{R}(t_1, t_2) \Rightarrow \phi(t_1, t_2)$, where $\mathcal{R}$ is a dependency relation (WR, RW or WW) and ϕ is some constraint on the transaction instances t_1 and t_2. The presence of a dependency edge between two transaction instances generally enforces some constraint on their parameters, since there must be some common record that must be accessed by both the transactions. We also use relations such as vis and ar to describe an abstract execution and to specify weak consistency and isolation specifications. A dependency edge also enforces constraints on vis and ar.

We now describe the encoding. We define an uninterpreted sort τ whose members are transaction instances. A finite sort $\mathbb{T} = \{T_1, \ldots, T_n\}$ contains the transaction types, and we define a function $\Gamma : \tau \to \mathbb{T}$ which associates each transaction instance with its type. We also define an uninterpreted sort R whose members will be records in the database. For each field f_i, we define the field projection function $f_i : R \to \mathbb{Z}$, which projects the value of the field in a record. For each variable v_{ij} of transaction type T_i, we define the variable projection function $v_{ij} : \tau \to \mathbb{V}$ which gives the value of the variable in a transaction instance. Here, $\mathbb{V}$ is the set of all possible values that can be stored in variables (in our case, $\bigcup_{i>0} \mathbb{Z}^i$). Note that while v_{ij} is defined for all types of transaction instances, it will only be applied on transaction instances of type T_i.

We define predicates $\mathsf{WR}, \mathsf{WW}, \mathsf{RW}$ all of type $\tau \times \tau \to \mathbb{B}$ which specify the read, write and anti-dependency relations respectively between transaction instances. We also define $\mathsf{WR}^R, \mathsf{RW}^R, \mathsf{WW}^R$ all of type $R \times \tau \times \tau \to \mathbb{B}$ which provide more context to the dependency relations, by also specifying the records causing the dependencies. They are related in the following fashion:

$$\forall (r : R)(t_1, t_2 : \tau)(\mathcal{R} \in \{\mathsf{WR}, \mathsf{WW}, \mathsf{RW}\}).\mathcal{R}^R(r, t_1, t_2) \Rightarrow \mathcal{R}(t_1, t_2) \tag{1}$$

$$\forall (t_1, t_2 : \tau)(\mathcal{R} \in \{\mathsf{WR}, \mathsf{WW}, \mathsf{RW}\}).\mathcal{R}(t_1, t_2) \Rightarrow \exists (r : R)\mathcal{R}^R(r, t_1, t_2) \tag{2}$$

We define predicates $\mathsf{vis}, \mathsf{ar}$ of type $\tau \times \tau \to \mathbb{B}$ which specify the visibility and arbitration relation between transaction instances. By placing constraints on these predicates we can specify various weak consistency and weak isolation models in a declarative fashion. Note that since we define these relations among transaction instances, we are already assuming that transactions execute atomically and in isolation. We relate them with the dependency relations :

$$\forall (t_1, t_2 : \tau).\mathsf{WR}(t_1, t_2) \Rightarrow \mathsf{vis}(t_1, t_2) \tag{3}$$

$$\forall (t_1, t_2 : \tau).\mathsf{WW}(t_1, t_2) \Rightarrow \mathsf{ar}(t_1, t_2) \tag{4}$$

$$\forall (t_1, t_2 : \tau).\mathsf{RW}(t_1, t_2) \Rightarrow \neg\mathsf{vis}(t_2, t_1) \tag{5}$$

We also add an auxiliary relation $\mathsf{D} : \tau \times \tau \to \mathbb{B}$ indicating the presence of any dependency edge between transaction instances.

$$\forall (t_1, t_2 : \tau).\mathsf{D}(t_1, t_2) \Leftrightarrow \mathsf{WR}(t_1, t_2) \wedge \mathsf{RW}(t_1, t_2) \wedge \mathsf{WW}(t_1, t_2) \tag{6}$$

A fundamental constraint relating WR, RW and WW relations is that if a transaction t_1 sees a record r written by t_2 (hence $\mathsf{WR}(t_2, t_1)$) and has an anti-dependency to another transaction t_3 due to the same record r, i.e. the value

read by t_1 would change if t_3 was made to be visible to t_1, then t_3 must install a later version of the record. This constraint can be encoded as follows :

$$\forall(t_1, t_2, t_3 : T)(r : R).\mathsf{WR}(r, t_2, t_1) \wedge \mathsf{RW}(r, t_1, t_3) \Rightarrow \mathsf{WW}(r, t_2, t_3) \tag{7}$$

Below, we show how some known weak consistency and isolation specifications can be encoded:

- Full Serializability :

$$\mathsf{vis} = \mathsf{ar} \tag{8}$$

- ~~Selective Serializability for transactional programs T_1, T_2 :~~

$$\forall(t_1, t_2 : \tau).((\Gamma(t_1) = T_1 \wedge \Gamma(t_2) = T_2) \vee (\Gamma(t_1) = T_2 \wedge \Gamma(t_2) = T_1)) \wedge \mathsf{ar}(t_1, t_2) \Rightarrow \mathsf{vis}(t_1, t_2) \tag{9}$$

- Causal Consistency :

$$\forall t_1, t_2, t_3.\mathsf{vis}(t_1, t_2) \wedge \mathsf{vis}(t_2, t_3) \Rightarrow \mathsf{vis}(t_1, t_3) \tag{10}$$

- Prefix Consistency (equivalent to Repeatable Read in centralized databases) :

$$\forall t_1, t_2, t_3.\mathsf{ar}(t_1, t_2) \wedge \mathsf{vis}(t_2, t_3) \Rightarrow \mathsf{vis}(t_1, t_3) \tag{11}$$

- Parallel Snapshot Isolation :

$$\forall t_1, t_2.\mathsf{WW}(t_1, t_2) \Rightarrow \mathsf{vis}(t_1, t_2) \tag{12}$$

Every path condition ϕ_c and SQL `WHERE` clauses ϕ_d contain variables and field names, but they can be specialized to a transaction instance and a record and directly encoded in our setting by replacing them with the corresponding variable projection and field projection functions. Note that since the field projection function only takes the record r as an argument, it can only be used for primary key fields which are accessed within ϕ_d. The other fields can be modified and hence the values contained within those fields are a function of both the record and the transaction instance, which we do not model. This does not affect the soundness of the approach, and ~~since majority of queries in database transactions only use the primary key fields~~, the effect on precision is minimal. As an optimization, we also find read-only fields (`RO`) which are never modified within any transaction, and treat fields in `RO` in the same manner as fields in `PK`. For a record r and a transaction instance t, we use $[\![\phi]\!]_{r,t}$ to denote this encoding, where each variable v is replaced by $v(t)$ and each field f is replaced by $f(r)$. Formally, $[\![\phi]\!]_{r,t}$ is computed as follows:

$$\begin{aligned}
[\![\phi_1 \circ \phi_2]\!]_{t,r} &= [\![\phi_1]\!]_{t,r} \circ [\![\phi_2]\!]_{t,r} \\
[\![\neg\phi]\!]_{t,r} &= \neg[\![\phi]\!]_{t,r} \\
[\![f \odot e]\!]_{t,r} &= \begin{cases} f(r) \odot [\![e]\!]_{t,r} & \text{if } f \cup \mathcal{F}(e) \subseteq \mathtt{PK} \cup \mathtt{RO} \\ true & \text{otherwise} \end{cases} \\
[\![e_1 \oplus e_2]\!]_{t,r} &= [\![e_1]\!]_{t,r} \oplus [\![e_2]\!]_{t,r} \\
[\![v]\!]_{t,r} &= v(t) \\
[\![n]\!]_{t,r} &= n
\end{aligned}$$

Here, $\mathcal{F}(e)$ indicates all the fields that are accessed within the expression e. With each SQL statement, we associate a path condition ψ which is simply the conjunction of the `IF` conditions on the path from the start of the transaction to the statement (in original form if the the statement is on the IF path, in negated form if the statement is on the $ELSE$ path).

For each local variable in a transaction which is also the output of a `SELECT` statement, we also define a predicate $v^{\texttt{NULL}} : T \to \mathbb{B}$. If the output of the `SELECT` query in a transaction instance t is `NULL`, i.e. there are no records which satisfy the `WHERE` clause of the statement, then $v^{\texttt{NULL}}(t)$ would be true. For every record r and transaction instance t, we define the predicate $\texttt{Alive}(r, t)$ which indicates whether r was alive in the database when t began its execution. A record becomes alive once a transaction inserts it into the database (using the `INSERT` statement), and becomes dead when it is deleted by a transaction (using the `DELETE` statement). The `Alive` predicate is useful in eliminating certain spurious dependencies (for example, if a `DELETE` of r is visible to a transaction t, then no update can be performed by t on r, and hence there can be no WR dependencies from t due to r).

WR-Update-Select

$$\frac{\begin{array}{c}T_1, \psi_1 \vdash \texttt{UPDATE SET f} = \texttt{e}_\texttt{c}\ \texttt{WHERE}\ \phi_1 \\ T_2, \psi_2 \vdash \texttt{SELECT f AS v WHERE}\ \phi_2\end{array}}{\begin{array}{c}\eta_{\mathsf{WR}}(T_1, T_2) = \eta_{\mathsf{WR}}(T_1, T_2) \vee (\exists r. [\![\psi_1]\!]_{t_1} \wedge [\![\phi_1]\!]_{t_1, r} \wedge [\![\psi_2]\!]_{t_2} \wedge [\![\phi_2]\!]_{t_2, r} \wedge \\ \texttt{Alive}(r, t_1) \wedge \neg v^{\texttt{NULL}}(t_2))\end{array}}$$

RW-Update-Select

$$\frac{\begin{array}{c}T_1, \psi_1 \vdash \texttt{UPDATE SET f} = \texttt{e}_\texttt{c}\ \texttt{WHERE}\ \phi_1 \\ T_2, \psi_2 \vdash \texttt{SELECT f AS v WHERE}\ \phi_2\end{array}}{\eta_{\mathsf{RW}}(T_2, T_1) = \eta_{\mathsf{RW}}(T_2, T_1) \vee (\exists r. [\![\psi_1]\!]_{t_1} \wedge [\![\phi_1]\!]_{t_1, r} \wedge [\![\psi_2]\!]_{t_2} \wedge [\![\phi_2]\!]_{t_2, r} \wedge \texttt{Alive}(r, t_1))}$$

WR-Insert-Select

$$\frac{\begin{array}{c}T_1, \psi_1 \vdash \texttt{INSERT VALUES}\ \bar{\texttt{f}} = \bar{\texttt{e}}_\texttt{c} \\ T_2, \psi_2 \vdash \texttt{SELECT f AS v WHERE}\ \phi_2\end{array}}{\begin{array}{c}\eta_{\mathsf{WR}}(T_1, T_2) = \eta_{\mathsf{WR}}(T_1, T_2) \vee (\exists r. [\![\psi_1]\!]_{t_1} \wedge \bigwedge_{i=1}^{len(\bar{f})} \bar{f}(i)(r) = [\![\bar{e}_c(i)]\!]_{t_1} \\ \wedge [\![\psi_2]\!]_{t_2} \wedge [\![\phi_2]\!]_{t_2, r} \wedge \texttt{Alive}(r, t_2) \wedge \neg v^{\texttt{NULL}}(t_2))\end{array}}$$

RW-Insert-Select

$$\frac{\begin{array}{c}T_1, \psi_1 \vdash \texttt{INSERT VALUES}\ \bar{\texttt{f}} = \bar{\texttt{e}}_\texttt{c} \\ T_2, \psi_2 \vdash \texttt{SELECT f AS v WHERE}\ \phi_2\end{array}}{\begin{array}{c}\eta_{\mathsf{RW}}(T_2, T_1) = \eta_{\mathsf{RW}}(T_2, T_1) \vee (\exists r. [\![\psi_1]\!]_{t_1} \wedge \bigwedge_{i=1}^{len(\bar{f})} \bar{f}(i)(r) = [\![e_c(i)]\!]_{t_1} \\ \wedge [\![\psi_2]\!]_{t_2} \wedge [\![\phi_2]\!]_{t_2, r} \wedge \neg\texttt{Alive}(r, t_2) \wedge PK(\phi_2) \Rightarrow v^{\texttt{NULL}}(t_2))\end{array}}$$

WR-Delete-Select

$$\frac{\begin{array}{c}T_1, \psi_1 \vdash \texttt{DELETE WHERE}\ \phi_1 \\ T_2, \psi_2 \vdash \texttt{SELECT f}_2\ \texttt{AS v}_2\ \texttt{WHERE}\ \phi_2\end{array}}{\begin{array}{c}\eta_{\mathsf{WR}}(T_1, T_2) = \eta_{\mathsf{WR}}(T_1, T_2) \vee (\exists r. [\![\psi_1]\!]_{t_1} \wedge [\![\phi_1]\!]_{t_1, r} \wedge [\![\psi_2]\!]_{t_2} \wedge [\![\phi_2]\!]_{t_2, r} \\ \wedge \texttt{Alive}(r, t_1) \wedge \neg\texttt{Alive}(r, t_2) \wedge PK(\phi_2) \Rightarrow \neg v_2^{NULL}(t_2))\end{array}}$$

RW-DELETE-SELECT

$$\frac{T_1, \psi_1 \vdash \texttt{DELETE WHERE } \phi_1 \quad T_2, \psi_2 \vdash \texttt{SELECT f}_2 \texttt{ AS v}_2 \texttt{ WHERE } \phi_2}{\eta_{\mathsf{RW}}(T_2, T_1) = \eta_{\mathsf{RW}}(T_2, T_1) \vee (\exists r. [\![\psi_1]\!]_{t_1} \wedge [\![\phi_1]\!]_{t_1,r} \wedge [\![\psi_2]\!]_{t_2} \wedge [\![\phi_2]\!]_{t_2,r} \wedge \texttt{Alive}(r, t_1) \wedge \neg v_2^{\texttt{NULL}}(t_2) \wedge \texttt{Alive}(r, t_2))}$$

WR-INSERT-SELECT-COUNT

$$\frac{T_1, \psi_1 \vdash \texttt{INSERT VALUES } \bar{\texttt{f}} = \bar{\texttt{e}}_\texttt{c} \quad T_2, \psi_2 \vdash \texttt{SELECT COUNT f AS v WHERE } \phi_2}{\eta_{\mathsf{WR}}(T_1, T_2) = \eta_{\mathsf{WR}}(T_1, T_2) \vee (\exists r. [\![\psi_1]\!]_{t_1} \wedge \bigwedge_{i=1}^{len(\bar{f})} \bar{f}(i)(r) = [\![\bar{e}_c(i)]\!]_{t_1} \wedge [\![\psi_2]\!]_{t_2} \wedge [\![\phi_2]\!]_{t_2,r} \wedge \texttt{Alive}(r, t_2) \wedge v(t_2) > 0)}$$

RW-INSERT-SELECT-COUNT

$$\frac{T_1, \psi_1 \vdash \texttt{INSERT VALUES } \bar{\texttt{f}} = \bar{\texttt{e}_\texttt{c}} \quad T_2, \psi_2 \vdash \texttt{SELECT COUNT f AS v WHERE } \phi_2}{\eta_{\mathsf{RW}}(T_2, T_1) = \eta_{\mathsf{RW}}(T_2, T_1) \vee (\exists r. [\![\psi_1]\!]_{t_1} \wedge \bigwedge_{i=1}^{len(\bar{f})} \bar{f}(i)(r) = [\![e_c(i)]\!]_{t_1} \wedge [\![\psi_2]\!]_{t_2} \wedge [\![\phi_2]\!]_{t_2,r} \wedge PK(\phi_2) \Rightarrow v(t_2) = 0 \wedge \neg \texttt{Alive}(r, t_2))}$$

WR-DELETE-SELECT-COUNT

$$\frac{T_1, \psi_1 \vdash \texttt{DELETE WHERE } \phi_1 \quad T_2, \psi_2 \vdash \texttt{SELECT COUNT f}_2 \texttt{ AS v}_2 \texttt{ WHERE } \phi_2}{\eta_{\mathsf{WR}}(T_1, T_2) = \eta_{\mathsf{WR}}(T_1, T_2) \vee (\exists r. [\![\psi_1]\!]_{t_1} \wedge [\![\phi_1]\!]_{t_1,r} \wedge [\![\psi_2]\!]_{t_2} \wedge [\![\phi_2]\!]_{t_2,r} \wedge \texttt{Alive}(r, t_1) \wedge \neg \texttt{Alive}(r, t_2) PK(\phi_2) \Rightarrow v_2(t_2) = 0)}$$

RW-DELETE-SELECT-COUNT

$$\frac{T_1, \psi_1 \vdash \texttt{DELETE WHERE } \phi_1 \quad T_2, \psi_2 \vdash \texttt{SELECT COUNT f}_2 \texttt{ AS v}_2 \texttt{ WHERE } \phi_2}{\eta_{\mathsf{RW}}(T_2, T_1) = \eta_{\mathsf{RW}}(T_2, T_1) \vee (\exists r. [\![\psi_1]\!]_{t_1} \wedge [\![\phi_1]\!]_{t_1,r} \wedge [\![\psi_2]\!]_{t_2} \wedge [\![\phi_2]\!]_{t_2,r} \wedge \texttt{Alive}(r, t_1) \wedge v_2(t_2) > 0 \wedge \texttt{Alive}(r, t_2))}$$

Here, $PK(\phi)$ implies that ϕ constrains all the primary key fields of the record. More formally, if $f_1, \ldots, f_k$ are the primary key fields, then $PK(\phi) \Leftrightarrow \phi \Rightarrow \bigwedge_{i=1}^{k} f_i = e_i$ for some expressions $e_1, \ldots, e_k$.

For every pair of transactions T_1, T_2, and all statements inside these transactions, we use the above rules to compute $\eta_{\mathcal{R}}(T_1, T_2)$ all $\mathcal{R} \in \{\mathsf{WR}, \mathsf{RW}, \mathsf{WW}\}$. We then add the following:

$$\forall t_1, t_2. (\Gamma(t_1) = T_1 \wedge \Gamma(t_2) = T_2 \wedge \mathcal{R}(t_1, t_2)) \Rightarrow \eta_{\mathcal{R}}(T_1, T_2) \tag{13}$$

For bounded anomaly detection of length k, we declare k instances of T $(c_1, \ldots, c_k)$ and add the following:

$$\forall i \in \{1, \ldots, k-1\}. \mathsf{D}(c_i, c_{i+1}) \tag{14}$$

$$\mathsf{D}(c_k, c_1) \tag{15}$$

To summarize, our encoding consists of equations (1)-(7), any of equations (8)-(12) depending upon the chosen consistency specifications, and equations (13)-(15).

1 Soundness

To explain the soundness of the approach, we first introduce the concept of abstract executions. Instead of using a state-based operational approach to express executions, we follow an event-based axiomatic approach. In this approach, the execution of a transaction instance consists of events. An event is just the execution of a database operation. Formally, an event is a tuple (e, o), where e is a unique event-ID and o is a database operation. We now describe the database operations. The database consists of a set of records, where each record is a set of values indexed using $\texttt{Fields}$. We assume a fixed non-empty subset of $\texttt{Fields}$ to be the primary key $\texttt{PK}$. Any two records must have distinct values in at least one of their $\texttt{PK}$ fields. Assume that there is a special boolean field called $\texttt{Alive} \in \texttt{Fields}$ which tracks whether the record is actually in the database. Initially, all records are not $\texttt{Alive}$. When a record is inserted into the database, it becomes $\texttt{Alive}$, and when the record is deleted, it again becomes not $\texttt{Alive}$. Assuming integer values, the database can be thought of as a function $(\mathbb{Z})^{|PK|} \rightarrow (\texttt{Fields} \setminus \texttt{PK}) \rightarrow \mathbb{Z}$. Let $\mathcal{R} = \texttt{PK} \rightarrow \mathbb{Z}$ be the set of all possible primary keys. Then, the set of all database operations is

$$\mathcal{O} = \{\texttt{wri}(r, f, n), \texttt{rd}(r, f, n) | r \in \mathcal{R}, f \in \texttt{Fields}, n \in \mathbb{Z}\}$$

Note that for write operations $\texttt{wri}(r, f)$, $f \in \texttt{Fields} \setminus \texttt{PK}$. Assume that every write operation writes distinct values. Insert and delete operations are simulated by writing to the $\texttt{Alive}$ field of the record. Also assume that when a transaction inserts a record, it immediately writes to all the non-$\texttt{PK}$-fields of the record to initialize the values. Finally, a record that is deleted can never be re-inserted into the database (in the same execution).

A transaction instance σ consists of a sequence of events. Formally, a transaction instance is a tuple $\sigma = (t, \varepsilon, \mathsf{po})$, where t is a unique transaction instance id, ε is the set of events and po is a total order on ε. We use the notation $\sigma \vdash o$ to specify that σ performs the database operation o. To simplify the presentation, we assume that a transaction does not read any record that it writes, inserts or deletes, and a transaction writes at most once to a field of a record. These assumptions are satisfied by all transactional programs that we have encountered, and our approach can be easily adapted if the assumptions are not satisfied.

An execution consists of a set of transaction instances Σ. Transaction instances will interact with each other through the database, and to specify these interactions, we define two binary relations on Σ: visibility (vis) and arbitration (ar). $\mathsf{vis} \subseteq \Sigma \times \Sigma$ is an anti-symmetric, irreflexive relation. Formally, $\forall t, s \in \Sigma. t \xrightarrow{\mathsf{vis}} s \Rightarrow \neg s(\xrightarrow{\mathsf{vis}} t)$. Intuitively, if $t \xrightarrow{\mathsf{vis}} s$, then all database operations performed by t are visible to s and hence will affect the output of the

reads performed by s. $\mathsf{ar} \subseteq \Sigma \times \Sigma$ is a total order (irreflexive, anti-symmetric, transitive, total relation) on all transaction instances of the execution, and essentially provides a fixed order in the case where multiple writes to the same record are visible to a transaction. Given a set of transaction instances Σ', we use the notation $[\Sigma']_{<\mathtt{wri}(r,f)>} = \{\sigma \in \Sigma' | \sigma \vdash \mathtt{wri}(r,f,n), n \in \mathbb{Z}\}$ to denote the set of transactions which are writing to field `f` of record `r`. We use the notation $\mathsf{MAX}_{\mathsf{ar}}(\Sigma')$ to denote $\sigma \in \Sigma'$ such that $\forall \sigma' \in \Sigma'.\sigma = \sigma' \vee \sigma' \xrightarrow{\mathsf{ar}} \sigma$. Given a transaction instance σ in an execution Σ, we use $\mathsf{vis}^{-1}(\sigma)$ to denote the set $\{\sigma' \in \Sigma | \sigma' \xrightarrow{\mathsf{vis}} \sigma\}$.

$$
\begin{aligned}
t \vdash \mathtt{rd}(r,f,n) \Rightarrow & (\exists s.s \vdash \mathtt{wri}(r,\mathtt{Alive},1) \wedge s \xrightarrow{\mathsf{vis}} t) \\
& \wedge (\forall t'.t' \xrightarrow{\mathsf{vis}} t \Rightarrow \neg(t' \vdash \mathtt{wri}(r,\mathtt{Alive},0))) \\
& \wedge f \notin \mathtt{PK} \Rightarrow \mathsf{MAX}_{\mathsf{ar}}([\mathsf{vis}^{-1}(t)]_{<\mathtt{wri}(r,f)>}) \vdash \mathtt{wri}(r,f,n) \\
& \wedge f \in \mathtt{PK} \Rightarrow r(f) = n
\end{aligned}
$$

The above condition specifies the last writer wins nature of the database.

Given an abstract execution $(\Sigma, \mathsf{vis}, \mathsf{ar})$, we now define its associated dependency graph G. The vertices of the graph are just the transaction instances, while the edges are obtained by taking a union of relations which are defined as follows:

- $\mathsf{WR}_{r,f} : t \xrightarrow{\mathsf{WR}_{r,f}} s$ if $s \vdash \mathtt{rd}(r,f,n)$ and $t = \mathsf{MAX}_{\mathsf{ar}}([\mathsf{vis}^{-1}(s)]_{<\mathtt{wri}(r,f)>})$
- $\mathsf{WW}_{r,f} : t \xrightarrow{\mathsf{WW}_{r,f}} s$ if $t \vdash \mathtt{wri}(r,f,n)$, $s \vdash \mathtt{wri}(r,f,m)$ and $t \xrightarrow{\mathsf{ar}} s$
- $\mathsf{RW}_{r,f} : t \xrightarrow{\mathsf{RW}_{r,f}} s$ if $t \vdash \mathtt{rd}(r,f,n)$, $s \vdash \mathtt{wri}(r,f,m)$ and there exists another transaction instance t' such that $t' \xrightarrow{\mathsf{WR}_{r,f}} t$ and $t' \xrightarrow{\mathsf{WW}_{r,f}} s$

In addition, we define $\mathsf{WR}, \mathsf{WW}, \mathsf{RW}$ respectively to be the union of $\mathsf{WR}_{r,f}, \mathsf{WW}_{r,f}, \mathsf{RW}_{r,f}$ for all r, f.

In our setting, transaction instances are generated by executing instantiations of transactional programs. A transactional program $\mathcal{T} \in \mathbb{T}$ is generated using the grammar specified earlier, and is associated with parameter variables $\bar{v}^p$ and local variables $\bar{v}^l$. Let $c(\mathcal{T})$ denote the program text of $\mathcal{T}$. An instantiation of $\mathcal{T}$ is an assignment of all the parameter variables of $\mathcal{T}$ to values. Given an abstract execution, every transaction instance t is associated with the transactional program $\mathcal{T}$ generating the instance, denoted as $\Gamma(t)$.

Different weak consistency and weak isolation models can be expressed by placing constraint (which we denote by Ψ) on the vis and ar relations associated with an abstract execution. This gives rise to the notion of valid abstract executions under specific models, which are executions satisfying the constraints associated with those models. Below, we provide examples of several known weak consistency and weak isolation models:

- Full Serializability : $\Psi_{Ser} \triangleq \mathsf{vis} = \mathsf{ar}$

- Selective Serializability for transactional programs $\mathcal{T}_1, \mathcal{T}_2$: $\Psi_{Ser(\mathcal{T}_1,\mathcal{T}_2)} \triangleq \forall t_1, t_2.\ ((\Gamma(t_1) = \mathcal{T}_1 \wedge \Gamma(t_2) = \mathcal{T}_2) \vee (\Gamma(t_1) = \mathcal{T}_2 \wedge \Gamma(t_2) = \mathcal{T}_1) \wedge t_1 \xrightarrow{\mathsf{ar}} t_2) \Rightarrow t_1 \xrightarrow{\mathsf{vis}} t_2$
- Causal Consistency : $\Psi_{CC} \triangleq \forall t_1, t_2, t_3.\ t_1 \xrightarrow{\mathsf{vis}} t_2 \wedge t_2 \xrightarrow{\mathsf{vis}} t_3 \Rightarrow t_1 \xrightarrow{\mathsf{vis}} t_3$
- Prefix Consistency (equivalent to Repeatable Read in centralized databases) : $\Psi_{PC} \triangleq \forall t_1, t_2, t_3.\ t_1 \xrightarrow{\mathsf{ar}} t_2 \wedge t_2 \xrightarrow{\mathsf{vis}} t_3 \Rightarrow t_1 \xrightarrow{\mathsf{vis}} t_3$
- Parallel Snapshot Isolation : $\Psi_{PSI} \triangleq \forall t_1, t_2.\ t_1 \xrightarrow{\mathsf{WW}} t_2 \Rightarrow t_1 \xrightarrow{\mathsf{vis}} t_2$

In addition, we can also define selective parallel snapshot isolation for transactional programs just like selective serializability. Different models can be also be combined together to create a hybrid model. For example, $\Psi_{PSI} \wedge \Psi_{PC}$ is equivalent to Snapshot Isolation in centralized databases.

We now describe the operational semantics of the transactional programs which results in abstract executions. This is an interleaving semantics where we can non-deterministically decide to begin a new instantiation of a transactional program. The system state is stored in terms of the committed transaction instances, the vis and ar relations among them, and a running pool of transaction instances. When a new execution of a transactional program begins, a subset of the committed transaction instances is non-deterministically selected to be made visible to the new instance. A view of the database is reconstructed based on the set of visible transactions and the ar relation, and all queries of the newly executing transaction instance are answered on the basis of this view. The newly executing transaction instance is added to the pool of running transactions. At any point, any transaction instance from the running pool can be non-deterministically selected for execution. Any new event generated during the execution of a transaction instance is stored in the running pool but is not made visible to other transaction instances. Finally, when a transaction instance wants to commit, it is checked whether constraints of the weak consistency and weak isolation model (Ψ) are satisfied if the instance were to commit, and if yes then the instance is allowed to commit and is added to the set of committed transaction instances.

Let Σ denote the set of committed transaction instances. The state is maintained as a tuple $(\Sigma, \mathsf{vis}, \mathsf{ar}, R, \Gamma)$ where R is the set of running transaction instances and $\Gamma : \Sigma \cup R \to \mathbb{T}$ maps transaction instances to the transactional programs which generated them. A running transaction instance is maintained as a tuple $r = (t, \varepsilon, c, \Delta, \Sigma_r, \theta)$, where t is the unique instance ID, ε is the set of events generated by the instance, c is the program to be executed, Δ is the view of the database, Σ_r is the set of committed transaction instances visible to r, and $\theta : \bar{v}_p(\Gamma(r)) \cup \bar{v}_l(\Gamma(r)) \to \mathbb{D}$ provides valuations of the parameter and local variables of the transaction instance. $\mathbb{D}$ is the set of all possible values that can be stored in variables, which in our case is $\bigcup_{k\geq 1} \mathbb{Z}^k \cup \bigcup_{k\geq 1} \mathbb{P}(\mathbb{Z}^k)$. The view of the database Δ is constructed from the set of committed transaction instances visible to the running instance Σ_r (which was decided when the running instance began its execution).

E-START

$$\frac{\mathcal{T} \in \mathbb{T} \quad \Sigma' \subseteq \Sigma \quad \Delta = \varsigma(\Sigma', \mathsf{ar}) \quad \theta(\bar{v}_p(\mathcal{T})) \in \mathbb{Z} \quad t \in \texttt{TID} \quad \Gamma' = \Gamma \cup \{(r, \mathcal{T})\} \\ r = (t, \{\}, c(\mathcal{T}), \Delta, \Sigma', \theta)}{(\Sigma, \mathsf{vis}, \mathsf{ar}, R, \Gamma) \to (\Sigma, \mathsf{vis}, \mathsf{ar}, R \cup \{r\}, \Gamma')}$$

E-STEP

$$\frac{r \to r'}{(\Sigma, \mathsf{vis}, \mathsf{ar}, R \cup \{r\}, \Gamma) \to (\Sigma, \mathsf{vis}, \mathsf{ar}, R \cup \{r'\}, \Gamma)}$$

E-COMMIT

$$\frac{r = (t, \varepsilon, \texttt{SKIP}, \Delta, \Sigma_r, \theta) \quad \sigma = (t, \varepsilon) \quad \mathsf{vis}' = \mathsf{vis} \cup \{(\sigma', \sigma) \mid \sigma' \in \Sigma_r\} \\ \mathsf{ar}' = \mathsf{ar} \cup \{(\sigma', \sigma) \mid \sigma' \in \Sigma\} \quad \Sigma' = \Sigma \cup \{\sigma\} \\ \Gamma' = \Gamma \cup \{(\sigma, \Gamma(r))\} \setminus \{(r, \Gamma(r))\} \quad \Psi(\Sigma', \mathsf{vis}', \mathsf{ar}', \Gamma')}{(\Sigma, \mathsf{vis}, \mathsf{ar}, R \cup \{r\}, \Gamma) \to (\Sigma', \mathsf{vis}', \mathsf{ar}', R, \Gamma')}$$

E-SELECT

$$\frac{\varepsilon' = \varepsilon \cup \{\texttt{rd}(r, f', n) \mid r \in \mathcal{R} \wedge f' \in \mathcal{F}(\phi) \wedge [\![f']\!]_{r,\Delta,\theta} = n\} \\ \cup \{\texttt{rd}(r, \bar{f}(i), n) \mid [\![\phi]\!]_{r,\Delta,\theta} \wedge [\![\bar{f}(i)]\!]_{r,\Delta,\theta} = n \wedge 1 \leq i \leq \texttt{len}(\bar{f})\} \\ \cup \{\texttt{rd}(r, \texttt{Alive}, n) \mid [\![\phi]\!]_{r,\Delta,\theta} \wedge \Delta(r)(\texttt{Alive}) = n\} \\ s = \{\Delta(r)(\bar{f}) \mid [\![\phi]\!]_{\Delta,\theta}(r) \wedge \Delta(r)(\texttt{Alive}) = 1\} \quad \theta' = \theta[v \to s]}{(t, \varepsilon, \texttt{SELECT } \bar{f} \texttt{ AS } v \texttt{ WHERE } \phi, \Delta, \Sigma_r, \theta) \to (t, \varepsilon', \texttt{SKIP}, \Delta, \Sigma_r, \theta')}$$

E-INSERT

$$\frac{\varepsilon' = \varepsilon \cup \{\texttt{wri}(r, \texttt{Alive}, 1) \mid \forall i. 1 \leq i \leq \texttt{len}(f) \Rightarrow r(\bar{f}(i)) = [\![\bar{e}_c(i)]\!]_\theta\}}{(t, \varepsilon, \texttt{INSERT VALUES } \bar{f} = \bar{e}_c, \Delta, \Sigma_r, \theta) \to (t, \varepsilon', \texttt{SKIP}, \Delta, \Sigma_r, \theta)}$$

E-UPDATE

$$\frac{\varepsilon' = \varepsilon \cup \{\texttt{rd}(r, f', n) \mid f' \in \mathcal{F}(\phi) \wedge [\![f']\!]_{r,\Delta,\theta} = n\} \\ \cup \{\texttt{wri}(r, f, n) \mid [\![\phi]\!]_{r,\Delta,\theta} \wedge [\![e_c]\!]_\theta = n \wedge \Delta(r)(\texttt{Alive}) = 1\} \cup \\ \{\texttt{rd}(r, \texttt{Alive}, n) \mid [\![\phi]\!]_{r,\Delta,\theta} \wedge \Delta(r)(\texttt{Alive}) = n\}}{(t, \varepsilon, \texttt{UPDATE SET } f = e_c \texttt{ WHERE } \phi, \Delta, \Sigma_r, \theta) \to (t, \varepsilon', \texttt{SKIP}, \Delta, \Sigma_r, \theta)}$$

E-DELETE

$$\frac{\varepsilon' = \varepsilon \cup \{\texttt{rd}(r, f', n) \mid f' \in \mathcal{F}(\phi) \wedge [\![f']\!]_{r,\Delta,\theta} = n\} \cup \{\texttt{wri}(r, \texttt{Alive}, 0) \mid [\![\phi]\!]_{r,\Delta,\theta}\}}{(t, \varepsilon, \texttt{DELETE WHERE } \phi, \Delta, \Sigma_r, \theta) \to (t, \varepsilon', \texttt{SKIP}, \Delta, \Sigma_r, \theta)}$$

E-SELECT-COUNT

$$\frac{\varepsilon' = \varepsilon \cup \{\texttt{rd}(r, f', n) \mid f' \in \mathcal{F}(\phi) \wedge [\![f']\!]_{r,\Delta,\theta} = n\} \\ \cup \{\texttt{rd}(r, \texttt{Alive}, n) \mid [\![\phi]\!]_{r,\Delta,\theta} \wedge \Delta(r)(\texttt{Alive}) = n\} \\ s = |\{r \mid [\![\phi]\!]_{r,\Delta,\theta} \wedge \Delta(r)(\texttt{Alive}) = 1\}| \quad \theta' = \theta[v \to s]}{(t, \varepsilon, \texttt{SELECT COUNT } f \texttt{ AS } v \texttt{ WHERE } \phi, \Delta, \Sigma_r, \theta) \to (t, \varepsilon', \texttt{SKIP}, \Delta, \Sigma_r, \theta')}$$

E-Select-Max

$$\frac{\begin{array}{c}\varepsilon' = \varepsilon \cup \{\mathtt{rd}(r, f', n) \mid f' \in \mathcal{F}(\phi) \wedge [\![f']\!]_{r,\Delta,\theta} = n\} \\ \cup\{\mathtt{rd}(r, f, n) \mid [\![\phi]\!]_{r,\Delta,\theta} \wedge [\![f]\!]_{r,\Delta,\theta} = n\} \\ \cup\{\mathtt{rd}(r, \mathtt{Alive}, n) \mid [\![\phi]\!]_{r,\Delta,\theta} \wedge \Delta(r)(\mathtt{Alive}) = n\} \\ s = \mathsf{MAX}\{[\![f]\!]_{r,\Delta,\theta} \mid [\![\phi]\!]_{r,\Delta,\theta} \wedge \Delta(r)(\mathtt{Alive}) = 1\} \quad \theta' = \theta[v \to s]\end{array}}{(t, \varepsilon, \mathtt{SELECT\ MAX}\ f\ \mathtt{AS}\ v\ \mathtt{WHERE}\ \phi, \Delta, \Sigma_r, \theta) \to (t, \varepsilon', \mathtt{SKIP}, \Delta, \Sigma_r, \theta')}$$

E-Select-Min

$$\frac{\begin{array}{c}\varepsilon' = \varepsilon \cup \{\mathtt{rd}(r, f', n) \mid f' \in \mathcal{F}(\phi) \wedge [\![f']\!]_{r,\Delta,\theta} = n\} \\ \cup\{\mathtt{rd}(r, f, n) \mid [\![\phi]\!]_{r,\Delta,\theta} \wedge [\![f]\!]_{r,\Delta,\theta} = n\} \\ \cup\{\mathtt{rd}(r, \mathtt{Alive}, n) \mid [\![\phi]\!]_{r,\Delta,\theta} \wedge [\![f]\!]_{r,\Delta,\theta} \leq s \wedge \Delta(r)(\mathtt{Alive}) = n\} \\ s = \mathsf{MIN}\{[\![f]\!]_{r,\Delta,\theta} \mid [\![\phi]\!]_{r,\Delta,\theta} \wedge \Delta(r)(\mathtt{Alive}) = 1\} \quad \theta' = \theta[v \to s]\end{array}}{(t, \varepsilon, \mathtt{SELECT\ MIN}\ f\ \mathtt{AS}\ v\ \mathtt{WHERE}\ \phi, \Delta, \Sigma_r, \theta) \to (t, \varepsilon', \mathtt{SKIP}, \Delta, \Sigma_r, \theta')}$$

E-Sequence

$$\frac{(t, \varepsilon, c1, \Delta, \Sigma_r, \theta) \to (t, \varepsilon', \mathtt{SKIP}, \Delta, \Sigma_r, \theta')}{(t, \varepsilon, c1; c2, \Delta, \Sigma_r, \theta) \to (t, \varepsilon', c2, \Delta, \Sigma_r, \theta')}$$

E-If-True

$$\frac{[\![\phi_c]\!]_\theta \quad (t, \varepsilon, c1, \Delta, \Sigma_r, \theta) \to (t, \varepsilon', \mathtt{SKIP}, \Delta, \Sigma_r, \theta')}{(t, \varepsilon, \mathtt{IF}\ \phi_c\ \mathtt{THEN}\ c_1\ \mathtt{ELSE}\ c_2, \Delta, \Sigma_r, \theta) \to (t, \varepsilon', \mathtt{SKIP}, \Delta, \Sigma_r, \theta')}$$

E-If-False

$$\frac{\neg[\![\phi_c]\!]_\theta \quad (t, \varepsilon, c2, \Delta, \Sigma_r, \theta) \to (t, \varepsilon', \mathtt{SKIP}, \Delta, \Sigma_r, \theta')}{(t, \varepsilon, \mathtt{IF}\ \phi_c\ \mathtt{THEN}\ c_1\ \mathtt{ELSE}\ c_2, \Delta, \Sigma_r, \theta) \to (t, \varepsilon', \mathtt{SKIP}, \Delta, \Sigma_r, \theta')}$$

$$\Delta = \varsigma(\varepsilon, ar)$$

$$r \in \mathbb{Z}^{PK}, \Delta(r)(\mathtt{f}) = n \Leftrightarrow \mathsf{MAX}_{\mathsf{ar}}([\varepsilon]_{<\mathtt{wri}(r,f)>}) \vdash \mathtt{wri}(r, f, n)$$

$$\begin{array}{ll}\mathcal{F}(\phi_1 \circ \phi_2) & = \mathcal{F}(\phi_1) \cup \mathcal{F}(\phi_2) \\ \mathcal{F}(\neg\phi) & = \mathcal{F}(\phi) \\ \mathcal{F}(f \odot e) & = \{f\} \cup \mathcal{F}(e) \\ \mathcal{F}(e1 \oplus e2) & = \mathcal{F}(e1) \cup \mathcal{F}(e2) \\ \mathcal{F}(f) & = \{f\} \\ \mathcal{F}(v) & = \phi \\ \mathcal{F}(n) & = \phi\end{array}$$

$$\begin{array}{lll}[\![\phi_1 \circ \phi_2]\!]_{r,\Delta,\theta} & = [\![\phi_1]\!]_{r,\Delta,\theta} \circ [\![\phi_2]\!]_{r,\Delta,\theta} & \\ [\![\neg\phi]\!]_{r,\Delta,\theta} & = \neg[\![\phi]\!]_{r,\Delta,\theta} & \\ [\![f \odot e]\!]_{r,\Delta,\theta} & = [\![f]\!]_{r,\Delta,\theta} \odot [\![e]\!]_{r,\Delta,\theta} & \\ [\![e_1 \oplus e_2]\!]_{r,\Delta,\theta} & = [\![e_1]\!]_{r,\Delta,\theta} \oplus [\![e_2]\!]_{r,\Delta,\theta} & \\ [\![f]\!]_{r,\Delta,\theta} & = \Delta(r)(f) & \mathsf{IF}\ f \notin \mathtt{PK} \\ [\![f]\!]_{r,\Delta,\theta} & = r(f) & \mathsf{IF}\ f \in \mathtt{PK} \\ [\![v]\!]_{r,\Delta,\theta} & = \theta(v) & \\ [\![n]\!]_{r,\Delta,\theta} & = n & \end{array}$$

$$
\begin{aligned}
[\![\phi_1 \circ \phi_2]\!]_\theta &= [\![\phi_1]\!]_\theta \circ [\![\phi_2]\!]_\theta \\
[\![\neg\phi]\!]_\theta &= \neg[\![\phi]\!]_\theta \\
[\![v \odot e]\!]_\theta &= \theta(v) \odot [\![e]\!]_\theta \\
[\![v]\!]_\theta &= \theta(v) \\
[\![e_1 \oplus e_2]\!]_\theta &= [\![e_1]\!]_\theta \oplus [\![e_2]\!]_\theta \\
[\![n]\!]_\theta &= n \\
[\![\texttt{NULL}]\!]_\theta &= \{\}
\end{aligned}
$$